Molecular Dynamics modeling of single asperity contact

by

Filip Henrik Larsen

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

May 2017

Abstract

We perform Molecular Dynamics simulations using LAMMPS, and model a single spherical asperity indenting a flat substrate. Both bodies are made by β-cristobalite, and we apply the Vashishta potential. The LAMMPS distribution is extended by a custom class, in order to compute contact forces acting on individual atoms. Radial distribution of the contact forces is computed and compared to Hertz' theory [1], and the results of Robbins et al. [2]. We find behavioral differences from the continuum theory. However, despite the fact that we allow both bodies to be elastic and include frictional forces, we experience a very similar distribution to the one obtained by Robbins. The static friction force is measured to be proportional to the load, in accordance with Amontons' law. The coefficient of static friction is estimated for several substrate thicknesses. It's shown that for thin substrates the coefficient increases linearly with the substrate thickness, in agreement with recent experiments conducted by Julien Scheibert [3].

Acknowledgements

Upon the completion of this work I have a small group of people who greatly deserve some appreciation. First of, I would like to thank my supervisor Anders Malthe-Sørenssen for insightful talks, the goals you have set for me, and the encouragement I am left with after every conversation.

Anders Hafreager has practically been a second supervisor to me, and been of tremendous help. Thanks for the technical support related to LAMMPS, Abel and Atomify, but also for the conversations, tips related to workflow, writing topics and life in general. You have made yourself available, and initiated video conferences (also with Anders Malthe-Sørenssen). I feel lucky to have become a friend of yours.

Henrik Sveinsson has helped me with Abel related Issues, and shown genuine interest in my progress. All the above have supported me through relevant and constructive discussions, which has benefited me greatly.

Lastly, I would like to thank my wife, Flora Joelle Mbuebue Larsen, for her patience and love throughout my years of studies. You, and our son to come, are my greatest source of motivation.

Oslo, 15th of May 2017
Filip Henrik Larsen

Contents

Chapter 1

Introduction

The past decade, there has been an increasing interest in the behavior of materials at nano-scale [4]. Scientists have been able to alter material structures at nanometer scales, improving their properties, even macroscopically. This enables us to construct materials, machines and medicine with new capabilities. In order to advance in this field of research, we need to be able to characterize material properties at nano-scale.

Today, experimental and computational technology has progressed to the point where we can probe and model surfaces at atomic level. A common experimental approach for measuring local surface properties is to use an AFM[1], which presses a tip with radii 10 to 1000nm into a surface and "feels" the force acting on the probe. Mechanical properties are then extracted from these results using continuum mechanics.

It has recently been demonstrated that for contact mechanics, the nano-scale structure of a contact affects the interaction. Using atomic-scale models, Mark Robbins and coworkers [5] demonstrated that for a small asperity - a bump on a surface - the details of the atomic ordering on the surface had a first order impact on the force distribution at the contact. Thus, the continuum approach breaks down for very small contacts. In another paper, Robbins [2] uses a Molecular Dynamics model to analyze the pressure distribution in a single asperity contact, which he compares to well known continuum theory. He keeps the asperity rigid - rendering its atoms unable to move relative to each other - in order to see how different surface structures affects the distribution. His results show that the differently constructed asperities exhibit very different pressure distributions.

Very recently, Julien Scheibert [3] has experimentally studied the coefficient of static friction acting between a polymer probe and a flat polymer substrate, laying on top of glass. His results indicate that for very thin substrates on the nanometer scale, the coefficient of static friction is linearly dependent on the substrate thickness. This is in contrast to current theory, where the thickness of

[1] Atomic force microscope

the substrate is unimportant.

In this project we will develop a Molecular Dynamics model for dry single asperity contact of silica, and address the force distribution and coefficient of static friction present. Our model will be similar to that of Robbins, but with the perturbation of having an elastic asperity as well as substrate. We want all bodies to remain elastic because this is its state in reality. A rigid asperity implies an infinitely large stiffness, which is physically impossible. Thus, we will simulate a more realistic situation than Robbins did.

1.1 Goals

a) Develop a Molecular Dynamics model

The model should consist of a spherical cap - representing the asperity - and a flat substrate. We must decide upon a material to use, and choose a potential model suitable to this type of simulation.

b) Compute contact force distribution

We wish to compute the spatial force distribution in the area of contact acting between the two bodies, preferably component wise. In order to achieve this, we will need to be able to recognize the contact forces. Due to the simple geometry of the system, we may define contact forces as any force acting on atoms of one body from the atoms of the other. Secondly, we need to know the normal vector of the surface, in order to define its normal and shear components. We must be careful to recognize that we operate at the discrete atomic level. Therefore, we should partition the system into a grid and for each cell in the grid, compute the sum of contact forces[2].

1) Develop a method for computing per-atom forces

As of now, LAMMPS does not contain the functionality to retrieve the net force acting on every individual atom of one group from all atoms of another group. A goal is therefor to develop such a method and integrate it in the LAMMPS distribution.

2) Develop a method to approximate the contact surface

To distinguish the components of the contact forces, we must approximate the surface, in order to determine its normal vector.

c) Compute the coefficient of static friction

The coefficient of static friction is defined as $\mu = dF_S/dN$, where F_S is the maximum static friction force, and N is the normal force. Thus, we need to implement a method for computing dF_S/dN, and apply it in several simulations using different substrate thicknesses.

d) Compare results to theory and to results of Robbins and Scheibert

At last, we wish to compare the resulting force distribution to Robbins' results and to Hertzian theory. We also wish to study the behavior of the coefficient of static friction as a function of substrate thickness. Based on this study our goal is to support or object to Scheibert's claim.

[2] Acting on one body from the other.

1.2 My contributions

During this thesis we have been using LAMMPS - a library of Molecular Dynamics code - to perform simulations. It is therefore important to state that the predominant majority of the code we have been using was not self-written. The author has experience with writing Molecular Dynamics code, and decided that there's no reason to reinvent the wheel in this case. Of course, writing the code from scratch provides great insight in the numerical and physical aspects of the simulations, but it's very time consuming and will not be as efficient as using a code that has be improved, refined and expanded for many years. Nevertheless, during this thesis we have:

- Expanded the LAMMPS library with a custom class (section 6.1), in order to compute per-atom forces. This class is fully functional and will be submitted to Steve Plimpton at the Sandia National Laboratories, for revision. If approved, it can be included in future distributions and be available for all users.

- Developed a method that partitions a domain in a two-dimensional grid, and applies plane regression on points within each cell of the grid (section 6.2). This method was used to approximate the normal vector of the substrate's surface.

- Modeled all simulations and written all algorithms in the post-processing scripts. We utilize open-source python packages in these, like: NumPy, Matplotlib, SciPy and Pickle.

1.3 Structure of the thesis

This thesis consists of two parts. In the first part, we present the fundamental theory and methods relevant to the problem. This includes an introduction to friction, elasticity and contact mechanics. We will then introduce theory, concepts and methods used in Molecular Dynamics. The software LAMMPS, which we have used extensively, will be presented, and is meant as a quick introduction for readers not yet familiar with it. We the describe how we modeled the system.

In the second part we describe our methods, post and discuss our results. Here we will show how we modeled the system, what processes we induce with LAMMPS, and how we measured and analyzed the results. The results will be posted immediately after the method's description, but the main discussion is found in the final chapter. Finally, we meet a conclusion and briefly summarize what we have done, what we have achieved and whether our initial goals has been met.

Part I

Theory and methods

Chapter 2

Friction, elasticity and contact mechanics

Friction can be defined as *the force that resists relative motion between two bodies in contact* [6]. It's a well-known phenomenon, though still not completely understood. Despite the fact that we learn about friction in introductory courses to physics, it is not at all easy to comprehend. It's very complex and still a field of research. The effects of friction is affected by surface roughness, material properties, ambient conditions, the geometry of contact area, and many other factors. In this chapter we will study some elementary aspects of dry friction between two solid bodies, elasticity and contact mechanics.

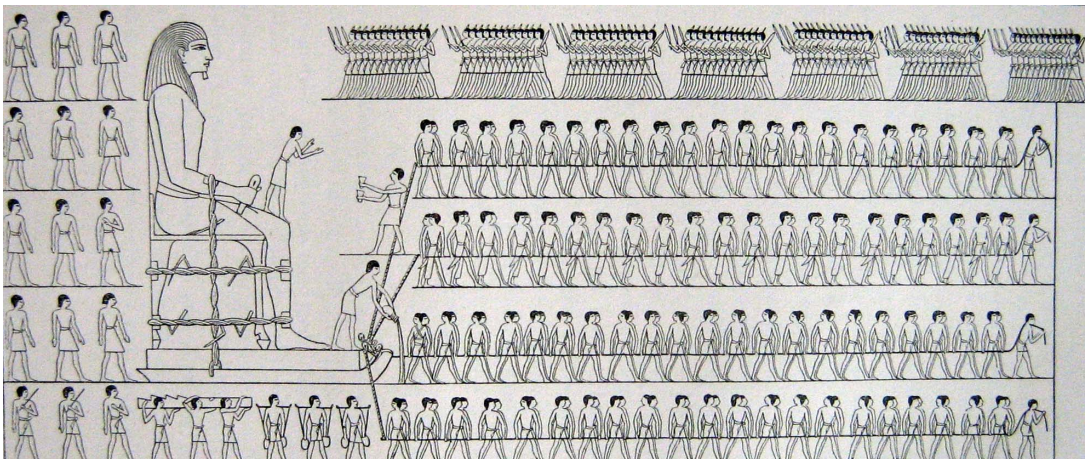


Figure 2.1: Painting from the tomb of Tehuti-Hetep, showing the transportation of an Egytian colossus.

2.1 Historical note

On the macroscopic level, friction has been observed since the dawn of man. Early actions that show awareness of the effects of friction was to chip stone in order to make tools, or rubbing wood in order to create fire. In the tomb of Tehuti-Hetep there was found the painting shown in figure 2.1, which show the moving of an Egyptian colossus. The statue is depicted pulled on a sledge, with officers standing on the front end of the sledge pouring water on the ground. This is evidence that they perceived the effect of lubrication even then, 1900 B.C..

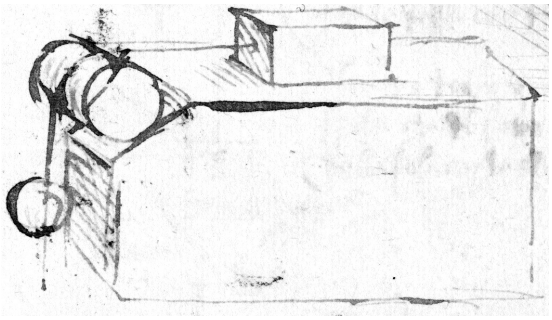


Figure 2.2: Drawing from Leonardo da Vinci's notebook. From Codex Arundel, British Library, London (Arundel folio 41r).

The first documented studies of friction were carried out during the renaissance by Leonardo da Vinci (1452-1519) [7]. Among his experiments he attached a block of wood to another object with a cord, and placed the block on a smooth surface. He was able to adjust the weight of the second object, which he let hang over a fixed cylinder, free to rotate. A sketch of this experiment's setup was found in his notebooks, and is shown in figure 2.2. He experimented with tilted planes and placed the block on different sides, changing the apparent area of contact. His results made him deduce the following statements, also found in his notes:

- Friction produces double the amount of effort if the weight be doubled.
- Friction made by the same weight will be of equal resistance at the beginning of the movement though the contact may be of different breadths or lengths.

This is now essentially known as the two first laws of friction. They were not recognized at the time, but were rediscovered later by Guillaume Amontons, and have since been commonly known as *Amontons' laws*. The linear relation described in the first postulate is commonly known as the coefficient of friction μ, defined as

$$F = \mu L. \tag{2.1}$$

Derjaguin proposed a modification to Amontons' law for adhering surfaces

$$F = F_0 + \mu L \tag{2.2}$$

where F_0 is the force present at zero load, due to attractive forces between the surfaces (adhesive force). Ultimately it changes the definition of μ to

$$\mu = \frac{dF}{dL}, \tag{2.3}$$

which express how the static friction force F increases with increasing load L. The difference between (2.1) and (2.3) is that for the first, $\mu = 0$ at $F = 0$ and $\mu = \infty$ at $L = 0$, while in the later, μ can be non-zero at $F = 0$ and μ is finite at $L = 0$. The latter is the common convention today. To account for the coefficient of friction, Amontons argued that the asperities in contact need to be pushed up a typical slope $\tan\theta$, which constitute an energy barrier. Soon after Leslie argued that the energy expended on pushing an asperity over another, is recovered when it descends on the other side. Thus there is no energy loss, and the surfaces should continually move without the presence of a driving force. He argued that friction had to be due some other mechanism than the pure geometrical.

Amontons' second law was long a mystery, until Bowden and Tabor concluded that the *real area of contact* is proportional to the load ($A \propto L$), as the electrical conductivity at a metals interface is proportional to the load [8].

The concept of force was not commonly recognized until Isaac Newton published *Principia*, which paved the way for studies of friction, there among the most comprehensive study carried out by Charles Augustin Coulomb (1736-1806). He investigated the influence of several factors on friction, namely [9]:

- The nature of the materials in contact and their surface coatings.
- The extent of the surface area.
- The normal force.
- The amount of time that the surfaces remained in contact prior to the applied force.
- Ambient conditions.

Among his results he found that the static friction force increased with the time the surfaces were in contact before applying shear force. Today this relation is known as

$$F_s(t) = A + B\ln(t). \tag{2.4}$$

Coulomb explained this behavior by regarding the materials as fibrous. Basically like a hairbrush, having fibers which get entangled into a mesh when in contact.

If the materials were in contact over a short period of time, fewer fibers would have gotten into their position of minimum energy in the mesh. When a shear force is applied, the fibers get tilted until they loosen from the mesh, at which point sliding occurs. This impression of the sliding mechanism is quite similar to the one we have today.

2.2 Macroscopic observations

2.2.1 Coefficient of friction

The coefficient of friction μ is a dimensionless scalar describing how the force of friction increases with the normal force, as given in equation (2.3). The value of μ is dependent on the material of the objects that are in contact, ambient conditions, presence of lubrication, etc.. For most materials the coefficient is higher if the objects are stationary, than if they are moving relative to each other (sliding). We may refer to them separately as the coefficient of static friction μ_s, and the coefficient of kinetic friction μ_k. Thus, $\mu_s > \mu_k$ for most materials[1].

2.2.2 Steady sliding

If we attach a spring to a block and pull on the spring with a constant velocity, we may get the evolution of friction force as shown in figure 2.3. We assume the block to have a constant normal force from the surface it rests upon. We observe that the friction force increases linearly during the loading of the spring, as expected from Hooke's law. Once the force from the spring overcomes the maximum static friction F_s, the block begins to move (slide) and the force of friction is reduced due to the change in coefficient of friction. The coefficient of static friction is found by using the maximum static friction force, F_s, in equation (2.3). Similarly the coefficient of sliding friction is found by using the value of the friction force in the sliding domain, F_k. In this scenario, the motion of the block resulted in a steady motion. As we shall see, that is not always the case.

Keep in mind that the forces we refer to are considered to be working on the center of mass of the block. The coefficient of friction is a macroscopic observation. It wouldn't make any sense to discuss a coefficient of friction between let's say two "blocks" containing 10 atoms each. How would one even define a normal force?

Steady sliding motion can be observed for instance if a car driver pulls on the break at high speed, so that the wheels are unable to turn. The wheels will then slide along the road until the car comes to a stop. At cites of traffic accidents it is common to measure the length of skid marks, in order to approximate a lower limit of a vehicles velocity.

[1] Teflon-on-teflon seems to have the same value for static- and kinetic friction.

2.2.3 Stick-slip motion

If we pull on a body with a spring with constant and reasonably low velocity, it will be subject to a linearly increasing force (Hooke's law). Because the object does not slide until we reach the maximum static friction, it will remain its position until it does. Once the force from the spring overcomes the maximum static friction, the body will begin to slide and the coefficient of friction changes to its kinetic state. It will accelerate the most in the beginning, and continue increasing its velocity until it passes an equilibrium position. Once it passes the equilibrium position, the friction force will overcome the spring force, and the velocity will decrease. When the velocity is sufficiently low, the object will stop abruptly. As the spring is still being moved, the spring force on the object will rise again (load), and we have a complete cycle. The time evolution of the force applied by the spring will be similar to the sketch shown in figure 2.4. If the surface of the block or substrate is non-homogeneous, the stick-slip motion may be more chaotic, than illustrated. In fact, it may be difficult to find any periodicity in the motion at all. We are not going to concern our self with chaotic stick-slip motion, since we will have perfectly smooth, dry and pure surfaces when doing Molecular Dynamics simulation. Stick-slip motion can be observed for instance when a bow slides over the strings of a violin, thus making them vibrate. By changing the pressure and speed of the bow, the performer can modify the sound produced. Earth quakes is another example, though this is of a more chaotic nature.

Whether the motion is of the steady type or the stick-slip type, is found experimentally to be depending on the velocity at which we pull the spring and stiffness of the spring. At sufficiently high speeds, we would always get a steady motion. Similarly, with a stiff enough spring we would also get a steady motion. While with a velocity, $v < v_s$, and stiffness $k < k_s$ we may observe a stick-slip motion. This can be illustrated by a phase diagram of velocity and spring stiffness (v, k), as in figure 2.5. The change in behavior from stick-slip to steady motion can be observed in our daily lives. The creaking of a door is due to stick-slip motion at the hinges. However, if we open the door quickly, the door will not creak.

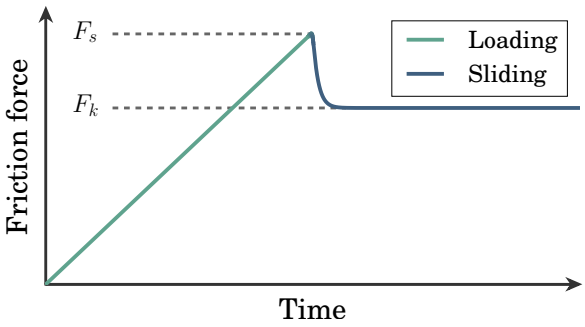


Figure 2.3: Behavior of friction force as function of time, for a block pulled at constant velocity and under constant load, resulting in a steady sliding motion.

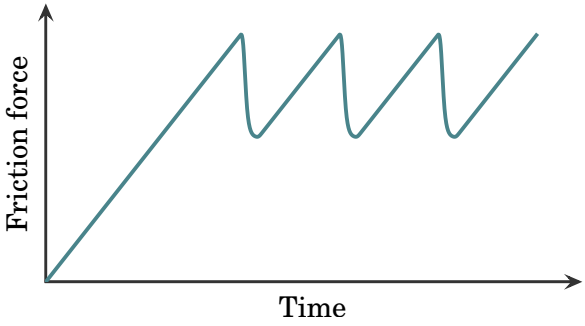


Figure 2.4: Behavior of friction force as function of time, for a block pulled at constant velocity and under constant load, resulting in a stick-slip motion.

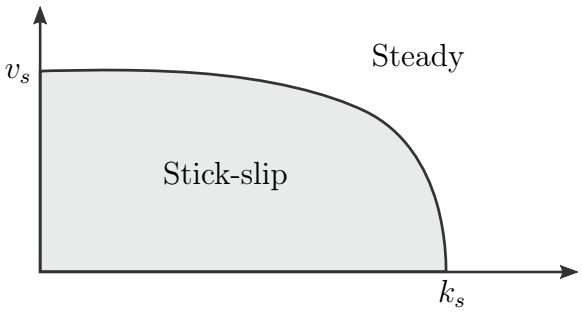


Figure 2.5: Phase diagram of velocity and spring stiffness (v, k). All combinations within the gray area will result in a stick-slip motion. All outside will result in a steady sliding motion.

2.3 Microscopic origin of friction

Friction is caused by bond formations between atoms of the two bodies, at the area of contact. This includes chemical bonds, hydrogen bonds, dipole-dipole interactions, and so on. The range of these interactions are typically quite short; at 10Å separation they are negligible. These bonds oppose relative motion, since the bonds must be broken for sliding to occur. As bonds are formed, they have associated energies required for the bonds to be broken. Thus, there is an energy barrier that must be overcome in order for relative motion to initiate, which is precisely our observation on the macroscopic level. The bonds can brake either by applying an external force or by thermal excitation[2]. The energy released by braking bonds can be converted into excitation of electrons, creation of phonons, mechanical energy in the form of elastic or plastic deformation, or heat [10].

Since the force of friction is due to an activation energy for breaking the bonds, it is intuitive that the force is proportional to the number of bonds

$$F \propto N_{\text{bonds}}. \tag{2.5}$$

As the formation of bonds only occur at the area of contact, one would expect $F \propto N_{\text{bonds}} \propto A$, which violates Amontons' second law. As we shall see in section 2.5, there is a difference in the *real* area of contact and the *apparent* area of contact.

2.4 Elasticity

We distinguish between elastic and plastic deformations. If a body is deformed by an external force, but recovers its initial structure after the force causing the deformation is removed, it is referred to as an *elastic* deformation. If it does not, it is regarded a *plastic* deformation. Obviously, there is a limit to the applied stress that results in elastic deformation; the so-called yield stress. In this section we will only consider elastic deformations, and review fundamental concepts of linear elasticity.

2.4.1 Strain and stress

Strain ϵ, is a measure of deformation. It's defined as an elongation from equilibrium $\Delta l = l - l_0$, relative to the equilibrium length l_0, and can be expressed as

$$\epsilon = \frac{\Delta l}{l_0} \tag{2.6}$$

For *normal* strain, Δl represents an elongation of the distance between two facing end planes of an infinitesimal cubic element. For *shear* strain, it represents a

[2]This is known as *creep*.

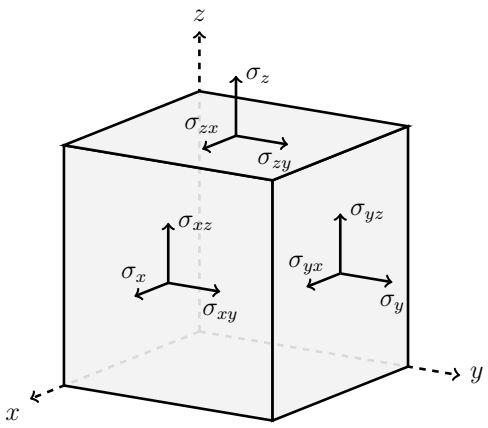


Figure 2.6: Graphical illustration of the components of the Cauchy stress tensor.

component of an in-plane displacement of the midpoints of said plains. It is common [11, 12] to discern the components of stress by using two indices, e.g. ϵ_{xy}. The first index indicates that we consider the planes normal to the x-axis, while the second that we consider the shear stress in the y-direction of these planes. Double indices, such as ϵ_{xx}, refer to the normal strain in the x-direction, and is commonly abbreviated to ϵ_x.

Stress σ, is a force density acting on a body due to internal forces, caused by strain. For stress on a plane, the *normal* and *shear* stress is defined as

$$\sigma_\perp = \frac{F_\perp}{A} \qquad \text{and} \qquad \sigma_\parallel = \frac{F_\parallel}{A} \tag{2.7}$$

respectively, where $F_\perp$ and $F_\parallel$ are the force components perpendicular and parallel to the plane. We use an equivalent notation for stress as for strain, with two indices distinguishing the components. This is illustrated in figure 2.6. The collection of stress and strain components are often composed into a stress tensor and a strain tensor. The stress tensor is commonly referred to as the Cauchy stress tensor.

$$\sigma = \begin{pmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z \end{pmatrix} \qquad \epsilon = \begin{pmatrix} \epsilon_x & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_y & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_z \end{pmatrix} \tag{2.8}$$

2.4.2 Hooke's law

As mentioned, stress is caused by strain. However, strain is also caused by stress. The relation between the two is known as Hooke's law. If we consider an infinitesimal cubical body subject to a tensile force in the x-direction, Hooke's

law states that the relation between stress and strain is given as

$$\epsilon_x = \frac{\sigma_x}{E}, \tag{2.9}$$

where E is a material property known as Young's modulus[3], which express the stiffness of the material. Here we assume that the deformation is elastic.

Often the faces of the infinitesimal cubic body are subject to forces in several direction. The general form of Hooke's law is expressed as

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl} \tag{2.10}$$

which relates the Cauchy stress tensor σ_{ij} to the strain tensor ϵ_{kl}, through the linear dependence of the fourth-order stiffness tensor c_{ijkl}. Note that each component of the Cauchy stress tensor is dependent on all elements of the strain tensor. The stress and strain tensors are both of rank 2 and have $3^2 = 9$ elements, while the stiffness tensor has rank 4 and contains $3^4 = 81$ elements. However, since both the Cauchy stress tensor and the strain tensor are symmetric,

$$\begin{aligned} \sigma_{ij} = \sigma_{ji} \Rightarrow C_{ijkl} = C_{jikl} \\ \epsilon_{kl} = \epsilon_{lk} \Rightarrow C_{ijkl} = C_{ijlk} \end{aligned} \tag{2.11}$$

In total there will be only 6 independent components for the stress and strain tensors, and 36 in the stiffness tensor. We may choose to express the stress and strain as vectors by using Voigts method for rank reduction [13]. This takes advantage of symmetry of the stress and strain tensors to express them as six-dimensional vectors [12].

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{xz} \\ \sigma_{yz} \end{pmatrix} \equiv \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{xy} \\ 2\epsilon_{xz} \\ 2\epsilon_{yz} \end{pmatrix} \equiv \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{pmatrix} \tag{2.12}$$

Similarly the stiffness tensor reduces to a 6×6 matrix (rank 2).

$$\hat{\mathbf{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{pmatrix} \tag{2.13}$$

[3] Young's modulus is also known as the elastic modulus.

The structure of the elements in the stiffness tensor are dependent on the isotropy[4] of the material of consideration. The general Hooke's law can now be expressed as a matrix product

$$\boldsymbol{\sigma} = \hat{\mathbf{C}}\boldsymbol{\epsilon} \tag{2.14}$$

2.4.3 Poisson's ratio

If we strain a body in the x-direction, the stress in the x-direction will be as given in equation (2.9). However, as the body is strained in this direction, it will contract in the other two directions. The relation between the applied strain in the x-direction and resulting strain in the other directions can be expressed as

$$\epsilon_y = -\nu\frac{\sigma_x}{E}, \qquad \epsilon_z = -\nu\frac{\sigma_x}{E}, \tag{2.15}$$

where ν is a constant known as Poisson's ratio. If an element is subject to uniformly distributed normal stress on all faces $\sigma_x, \sigma_y, \sigma_z$, we can use equation (2.9) and (2.15) in order to compute the resulting strains. We superpose the strain components produced by each of the stresses to get

$$\epsilon_x = \frac{1}{E}[\sigma_x - \nu(\sigma_y + \sigma_z)] \tag{2.16}$$

$$\epsilon_y = \frac{1}{E}[\sigma_y - \nu(\sigma_x + \sigma_z)] \tag{2.17}$$

$$\epsilon_z = \frac{1}{E}[\sigma_z - \nu(\sigma_x + \sigma_y)]. \tag{2.18}$$

2.5 Contact mechanics

2.5.1 Real area of contact

Today we know that most surfaces are rough, at least at a microscopic level, having some degree of asperities. A *contact* is defined as a point where two surfaces meet and where bonds can be formed [10]. It is actually the asperities that make up the contact areas between surfaces. The real area of contact is therefore the sum of individual contact area of the asperities. We can approximate the real area macroscopically. Imagine two surfaces initially apart, having no area of contact, as in figure 2.7. The top body has a load L due to its mass. It will move downwards, and at some point a single asperity will come in contact with the substrate. The pressure at the point of contact will be huge, since a single

[4]Directionally dependent physical properties.

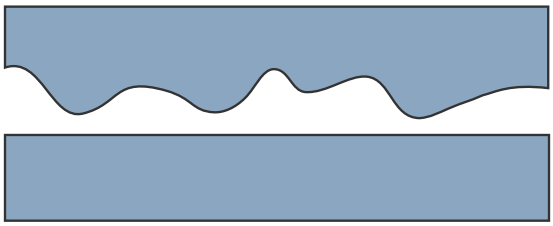


Figure 2.7: Sketch of two surfaces with different roughness. As they are forced towards each other, the entire load will rest on small sections of the surface.

asperity carries the entire load with a very small contact area[5] A_r.

$$p = \frac{L}{A_r} \tag{2.19}$$

The pressure will surely be much greater than the yield stress σ_c, and so the asperity will deform plastically. As the surface continues to descend, other asperities will come in contact with the substrate, increasing the contact area and share the load. They will all deform plastically until the area of contact is large enough so that the pressure at the points of contact is lower than the yield stress. At this point the top body is resting on the substrate. Thus, we should be able to relate the real area of contact to the yield stress as

$$A_r = \frac{L}{\sigma_c}. \tag{2.20}$$

At this point it is clear that Amontons' second law is preserved, since

$$F \propto N_{\text{bonds}} \propto A_r \propto \frac{L}{\sigma_c} \tag{2.21}$$

is not dependent on the apparent area of contact. In fact we just found Amontons' first law!

As an example[6] of the difference between apparent and real area of contact, consider a steel plate of size $10cm \times 10cm \times 1cm$. The apparent area is then $A_a = 100cm^2$. Assuming steel to have a density $\rho = 10g/cm^3$ and yield stress at $\sigma_c = 10^9 N/m^2$, the real area of contact is $A_r = 10^{-8}m^2 = 10^{-4}cm^2$. Thus, there is a huge difference in the apparent and real area of contact, a factor of a million in this case, but only the real area contributes to the effects of friction.

2.5.2 Hertz theory

Hertz' theory describe non-adhesive contact, and assumes that solids interact with a purely repulsive "hard-wall" interaction. It is assumed that the contacting

[5]Typical radius of the contact area on an asperities is $\sim 10\mu m$.
[6]This example is also found in reference [10].

solids are described by continuum elasticity, and discrete atomic structure is ignored [2]. Often it is also assumed that the two solids are isotropic, with Young's modulus E_1 and E_2 and Poissson's ratio ν_1 and ν_2. One can then compute an *effective modulus* E^* defined as

$$\frac{1}{E^*} = \frac{1-{\nu_1}^2}{E_1} + \frac{1-{\nu_2}^2}{E_2} \tag{2.22}$$

Heinrich Hertz predicted that when a sphere with radius of curvature R indents a flat surface, the area of contact is circular with a radii [1]

$$a = \left(\frac{3NR}{4E^*}\right)^{1/3}, \tag{2.23}$$

and the normal pressure p is related to it as

$$p(r) = p_0\sqrt{1-\frac{r^2}{a^2}}, \tag{2.24}$$

where r is the radial distance to the center of the indentation, and p_0 is the maximum value of the normal pressure, at the center of the indentation. The maximum pressure is defined as

$$p_0 = \frac{2aE^*}{\pi R}. \tag{2.25}$$

Also, the normal displacement δ_H of the spheres tip is related to a by

$$\delta_H = a^2/R, \tag{2.26}$$

where $\delta_H = 0$ is the vertical position of the spheres at first contact. The subscript H indicates that this is Hertz prediction, and follows the notation used in [1]. The assumptions made in this theory are that:

- Strains are small and within the elastic limit.
- Surfaces are continuous and non-conforming[7].
- Each body can be considered an elastic half-space.
- The surfaces are frictionless.

[7] A non-conforming contact is when dissimilar surfaces (which fit badly together) are in contact. When under zero load, they would only touch at a single point. The real area of contact is small compared to the sizes of the objects and the stress is highly concentrated.

2.6 Previously conducted studies

The field of friction is not new to science in any regard. However, thanks to advances in technology and computational power, we are now able to simulate physical systems at an atomic level. Continuum mechanics does not consider the nature of the atomic structure and interaction within solids. This has recently been proven to be an important detail when describing mechanics at the discrete atomic level. Mark O. Robbins, has written several articles on this matter [2, 5, 14, 15], one of which is very relevant to this thesis [2]. In collaboration with Binquan Luan, he used LAMMPS to do Molecular Dynamics simulations of a single spherical asperity indenting a flat substrate. They measured pressure distribution in the area of contact, without adhesion, and compared it to the solution predicted by Hertz' theory. In order to simulate nonadhesive contact, they used the Lennard-Jones[8] potential truncated at $2^{1/6}\sigma$, which assures only repulsive forces between the two bodies. They did this for differently constructed crystalline spheres: bent, amorphous and stepped. Examples of such surfaces are shown in figure 2.8. Also, for the bent tips, they used three different densities. They kept the sphere rigid throughout the simulation, fixing the atoms of the sphere relative to each other. Their results are shown in figure 2.9, and demonstrate that atomic structure clearly affects the distribution of pressure. The stepped tip deviated the most from the theory, having a behavior similar to that of a "flat punch"! They also noticed a dip in the pressure right at the center, both for the amorphous and bent incommensurate tips, obviously deviating from Hertz' theory. Thus, atomic structure is of importance at nano-scale.

Robbins and Luan makes two decisions that obviously renders their results unrealistic. First, they assume nonadhesive contact, that there are only repulsive forces between the two bodies. This is obviously not the case in reality. Secondly, they keep the sphere rigid. This implies an infinitely large stiffness in the material, which is unphysical.

In this thesis, we will perform a similar simulation, but with these important differences: We will allow both the sphere and the substrate to be elastic. Both these bodies will be constructed by a silica named β-cristobalite. We will apply the Vashishta potential[9], which includes attractive forces, and even allow construction of covalent bonds. We will compare our results to theirs and to Hertz theory in chapter 8.1, conscious about these differences.

JULIEN SCHEIBERT!

[8] The Lennard-Jones potential is discussed in section 3.2.1.

[9] The computed forces will only reflect the two-body term of the potential.

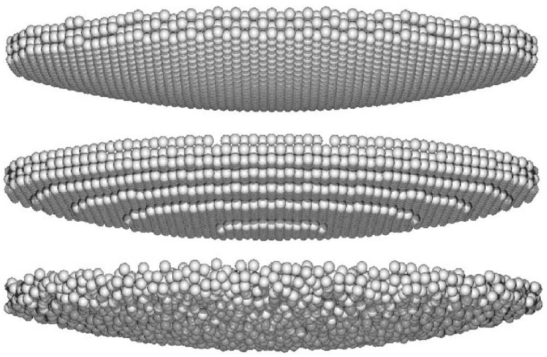


Figure 2.8: Snapshots of atoms near the center regions of spherical tips. From top to bottom, tips are made by bending a crystal, cutting a crystal, or cutting an amorphous solid. Three ratios ν of the atomic spacing in bent crystals to that in the substrate are considered; a dense case $\nu = 0.05$, a commensurate case $\nu = 1$, and an incommensurate case $\nu = 0.944$. The step structure of cut crystalline tips is not unique, leading to variations in their behavior. This figure is copied from [2].

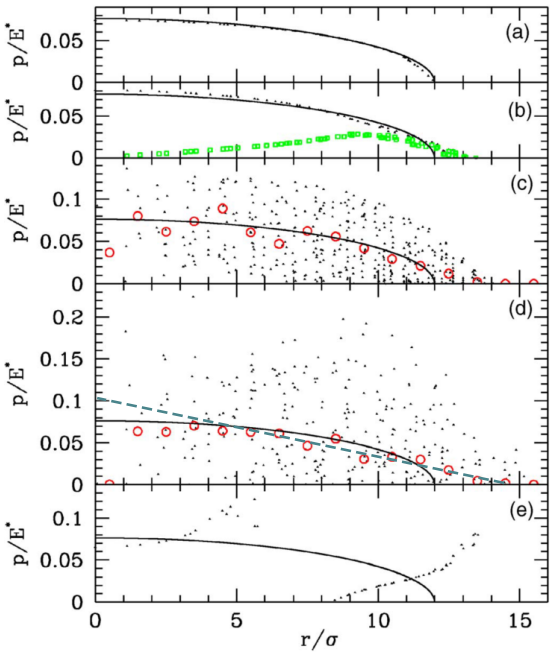


Figure 2.9: Local normal pressure vs radius for five different tip geometries (a) dense tip, (b) bent commensurate crystal, (c) bent incommensurate crystal, (d) amorphous, and (e) stepped crystal. All are nonadhesive, and have the same nominal radius. Solid lines show the prediction of Hertz theory, dots show the pressure on each surface atom, and circles in (c) and (d) show the mean pressure in radial bins. Squares in (b) show the component of the tangential force directed radially from the center of the contact. This figure is copied from [2]. However, we have inserted the dashed line in (d) to prove a point later.

Chapter 3

Molecular dynamics

Molecular dynamics is a method that uses Newton's equations of motion to simulate the movement of interacting atoms. Each atom is treated as a point particle with mass and charge, and the interaction between them is described by a force field. Thus, the force acting on a particle is computed as

$$\mathbf{F_i} = -\nabla U_i(\mathbf{r}^N), \tag{3.1}$$

where $\mathbf{r}^N = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ is the position of all atoms in the system. All the detail of the simulation lay in the interaction potential U. In order to obtain realistic simulations, the potentials often doesn't only consist of a pair-interaction, but many-body interactions as well. Also, methods for controlling the thermodynamic properties and boundary conditions pose additional complications. Lastly, the computational efficiency aspect of the simulation has to be evaluated. The computation time is predominantly spent in computing the forces. Thus, our methods must be able to ignore negligible parts of the force contributions, but still maintain a satisfactory level of accuracy.

Molecular dynamics aid our understanding of macroscopic phenomena by giving us insight in microscopic effects and behavior. It can be used if an experiment is too difficult, dangerous or expensive to do otherwise, for instance experiments at extreme temperatures or pressure. The areas of application are wide, and it has become a very popular field in material science, biochemistry and biophysics.

In this chapter we will present the basic principles of molecular dynamics, including essential algorithms, common methods and efficiency improvements.

3.1 Time-integration

The evolution of particles position and velocity is carried out through time integration. In molecular dynamics, atoms' movement is governed by Newton's

second law

$$\sum_{i=1}^{N} \mathbf{F_i} = m_i \mathbf{a_i} \tag{3.2}$$

Theoretically, the computation of particles movement is easy to assimilate. We know that acceleration is the time-derivative of velocity, which in turn is the time-derivative of position. Thus, we should be able to retrieve the velocities and positions by time-integrating the acceleration once and twice, respectively

$$\mathbf{v_i}(t) = \mathbf{v_i}(0) + \int_0^t \mathbf{a_i}(t)dt \tag{3.3}$$

$$\mathbf{r_i}(t) = \mathbf{r_i}(0) + \int_0^t \mathbf{v_i}(t)dt. \tag{3.4}$$

When solving an integral numerically, we're forced to discretize the problem and use an approximation method. This is done by dividing the domain into steps of size Δt and approximate the area of the function inside each such partition. Typically, the error of the approximation decreases when reducing the time step Δt. However, a smaller step length leads to higher computational expense, and may limit the size or time-frame we are able to compute. These are common considerations when doing computational research. One has to balance precision, size/time-frame, time and expense. It is therefore very important to choose a numerical scheme that has a satisfying level of precision as well as speed.

There are numerous numerical integration methods, but not all these have properties that are desirable for molecular dynamics simulation. The most fundamental features the scheme should have are high precision, computationally cheap, good energy conservation, reversibility and to be deterministic. Conservation of energy is fundamental in physics, and in order to simulate the microcanonical ensemble (NVE), the integration method cannot have an energy drift. The dynamics should be reversible, meaning one should be able to simulate the system in the opposite direction in time, and arrive at past states. It should be deterministic in the sense that by the information about a specific state $(r(t_0), v(t_0))$ one should be able to compute the state at any other time $(r(t), v(t))$, be it future of past. Molecular dynamics simulations are computational expensive, and practically all the work is spent computing forces. More precisely, computing the sum of forces on each atom. The choice of integration scheme is therefore very important. As it turns out, probably the best scheme for the task is also one of the simplest.

Velocity Verlet

In molecular dynamics the most common scheme for time-integration is the *Velocity Verlet* algorithm, which is a specific type of Verlet integration. It is derived

by taylor expanding the position both one time step forward and one backwards as follows:

$$r(t+\Delta t) = r(t) + \frac{\Delta t}{1!}r'(t) + \frac{\Delta t^2}{2!}r''(t) + \frac{\Delta t^3}{3!}r'''(t) + \mathcal{O}(\Delta t^4) \tag{3.5}$$

$$r(t-\Delta t) = r(t) - \frac{\Delta t}{1!}r'(t) + \frac{\Delta t^2}{2!}r''(t) - \frac{\Delta t^3}{3!}r'''(t) + \mathcal{O}(\Delta t^4) \tag{3.6}$$

If we add these two together, the odd terms cancel. After rearranging, we are left with the very simple verlet algorithm.

$$r(t+\Delta t) = 2r(t) - r(t-\Delta t) + \Delta t^2 r''(t) + \mathcal{O}(\Delta t^4) \tag{3.7}$$

Or written in a compact notation where

$$\mathbf{r}^n = r(t) \qquad \mathbf{r}^{n\pm1} = r(t \pm \Delta t) \qquad \mathbf{a}^n = \dot{\mathbf{v}}^n = \ddot{\mathbf{r}}^n \tag{3.8}$$

this can be expressed as

$$\mathbf{r}^{n+1} = 2\mathbf{r}^n - \mathbf{r}^{n-1} + \Delta t^2 \mathbf{a}^n \ . \tag{3.9}$$

Notice that we do not rely on information about the velocity in order to predict the next position. The algorithm in equation (3.9) is known as Strömer-Verlet. It's not self-starting, since you need to know the previous and current steps to predict the next. Thus, it needs to be initialized in a way, e.g. by using algorithm (3.11) for the first step. It has a high order truncation error, proportional to $\mathcal{O}(\Delta t^4)$, however it's vulnerable to round-off errors. This is because the last term is proportional to Δt^2, and is potentially much smaller than the other terms. When trying to add it with the other terms, this may be a source of round-off error [16]. As a remedy to this problem, the Velocity Verlet algorithm incorporates a second order differential approximation of the velocity

$$\mathbf{v}^n = \frac{\mathbf{r}^{n+1} - \mathbf{r}^{n-1}}{2\Delta t} + \mathcal{O}(\Delta t^2). \tag{3.10}$$

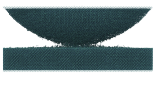
Resulting in

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \Delta t \mathbf{v}^n + \frac{\Delta t^2}{2}\mathbf{a}^n \tag{3.11}$$

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \Delta t \left(\mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}^n \right) \tag{3.12}$$

We may recognize the parenthesis as $\mathbf{v}^{n+1/2}$, leaving us with

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \Delta t \mathbf{v}^{n+1/2} \ . \tag{3.13}$$



Next, we taylor expand the velocity and use a first order approximation of $\dot{\mathbf{a}}^n$.

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t \mathbf{a}^n + \frac{\Delta t^2}{2}\dot{\mathbf{a}}^n + \mathcal{O}(\Delta t^3) \tag{3.14}$$

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t \mathbf{a}^n + \frac{\Delta t^2}{2}\frac{\mathbf{a}^{n+1} - \mathbf{a}^n}{\Delta t} \tag{3.15}$$

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{\Delta t}{2}\left(\mathbf{a}^{n+1} + \mathbf{a}^n\right) \tag{3.16}$$

$$\mathbf{v}^{n+1} = \mathbf{v}^{n+1/2} + \frac{\Delta t}{2}\mathbf{a}^{n+1} \tag{3.17}$$

Equations (3.13) and (3.17) define the Velocity Verlet algorithms. The final Velocity Verlet algorithm is thus:

$$\mathbf{v}^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}^n \tag{3.18}$$

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \Delta t \mathbf{v}^{n+1/2} \tag{3.19}$$

$$\mathbf{a}^{n+1} = -\nabla U(\mathbf{r}^{n+1})/m \tag{3.20}$$

$$\mathbf{v}^{n+1} = \mathbf{v}^{n+1/2} + \frac{\Delta t}{2}\mathbf{a}^{n+1} \tag{3.21}$$

The Velocity Verlet algorithm has the same truncation error as the original, but a lower round-off error. This algorithm has good energy conservation, and is symplectic.

3.2 Potentials

As mentioned, the force acting on each atom is computed using a potential model. In order to obtain realistic dynamics, it is therefore crucial to have an accurate potential model. The potentials are usually created for a particular kind of compound, like clay or hydrocarbonds [17, 18], and the parameters are derived from quantum chemistry calculations in combination with empirical methods. Some are more general than others, allowing other sets of parameters to be applied when using different materials. Different potential models may account for different forces. As we will see, the Lennard-Jones potential only account for steric repulsion and the Van der Waal force, rendering its applicability very limited. Besides the degrees of freedom of the potential, there are other aspects to evaluate as well. Like how we deal with bonds, for instance. The choice of potential model should always be dependent on the material and problem at hand.

3.2.1 Lennard-Jones

One of the simplest and most known potentials is the Lennard-Jones potential. It was first proposed in 1924 by John Edward Lennard-Jones. It takes the form

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], \tag{3.22}$$

where the first term represents Pauli repulsion due to overlapping electron orbitals, and the last represents the van der Waal force. The constant σ expresses the distance at which the inter-atomic potential is zero, while ϵ is the depth of the well, also regarded as the strength of the potential. r is of course the inter-atomic distance. A plot of the potential is shown in figure 3.1. The inter-atomic distance at which the potential is at its minimum can easily be shown to be $r = 2^{1/6}\sigma$.

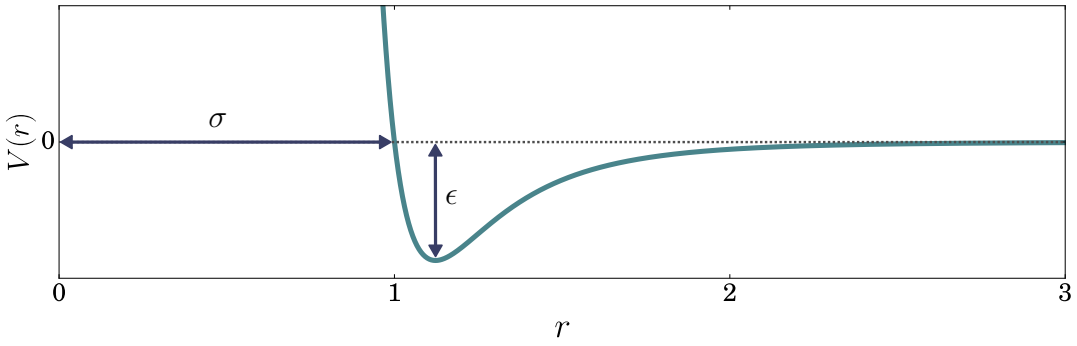


Figure 3.1: Lennard-Jones potential as function of inter-atomic distance, r, in units of σ.

Since the potential only account for the Pauli repulsion and van der Waal forces, its applicability is limited to systems consisting of neutral atoms or molecules, where there are no bonds present. It would suffice for studying noble gases, for instance, but that quickly becomes quite dull. In order to obtain realistic results, we need more degrees of freedom, accounting for more of the present forces.

3.2.2 Vashishta

The Vashishta potential consists of two terms that represent two- and three-body interactions respectively. These incorporate physical effects such as steric repulsion, coulomb interaction, dipole interaction, Van der Waal interaction and energy related to covalent bonds [19]. It can be expressed as

$$V = \sum_{i<j} V_{ij}^{(2)}(r_{ij}) + \sum_{i<j<k} V_{ijk}^{(3)}(\mathbf{r}_{ij}, \mathbf{r}_{ik}), \tag{3.23}$$

where $\mathbf{r}_i$ represents the position of the i-th atom, $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, and $r_{ij} = |\mathbf{r}_{ij}|$ is the distance between atom i and atom j. The two-body term, denoted by superscript (2), sums over all atoms j within a cutoff range r_c from atom i, and is given as

$$V_{ij}^{(2)}(r) = \frac{H_{ij}}{r^{\eta_{ij}}} + \frac{Z_i Z_j}{r} e^{-r/r_{1s}} - \frac{D_{ij}}{2r^4} e^{-r/r_{4s}} - \frac{W_{ij}}{r^6}. \tag{3.24}$$

where H_{ij} and η_{ij} are the strength and exponent of the steric repulsion respectively, Z_i is the effective charge of atom i, r_{1s} and r_{4s} are screening lengths for the coulomb and dipole interaction respectively, D_{ij} is the strength of the dipole interaction, and W_{ij} is the strength of the Van der Waal interaction.

The three-body term of the potential, denoted by superscript (3), sums over all atoms j and k , that are covalently bonded to atom i, within a cutoff range of r_0, and is expressed as

$$V_{ijk}^{(3)}(\mathbf{r}_{ij}, \mathbf{r}_{ik}) = B_{ijk} \exp\left(\frac{\xi}{r_{ij} - r_0} + \frac{\xi}{r_{ik} - r_0}\right) \frac{(\cos\theta_{ijk} - \cos\theta_0)^2}{1 + C_{ijk}(\cos\theta_{ijk} - \cos\theta_0)^2} \tag{3.25}$$

where B_{ijk} is the strength of the three-body term, ξ and C_{ijk} are constant parameters, and θ_{ijk} is the angle between $\mathbf{r}_{ij}$ and $\mathbf{r}_{ik}$. The values of all these parameters depend on the atom types. For instance, the strength of the steric repulsion, H_{ij}, for Si-Si interaction is different from that of Si-O interactions. For the case of silica, these metrics have been computed already, and when using the Vashishta potential in LAMMPS there is a file containing them. As for the Lennard-Jones potential, the two-body potential term is truncated and shifted to prevent unphysical jerks, and retain stability. This is done the same way as in equation (3.33). The three-body term mimics the forces due to bond bending. However, it saturates as the angle becomes too large, which allow bonds to be broken. The Vashishta potential is not *bonded*, which means that bonds can be created and broken. A non-bonded potential is very important in contact simulations. Not only because of effects like adhesion, but also to allow the surface atoms to organize freely, which would not be the case if atoms were fixed to each other. This renders the Vashishta potential superior to bonded[1] potentials like the ClayFF potential, in this particular problem.

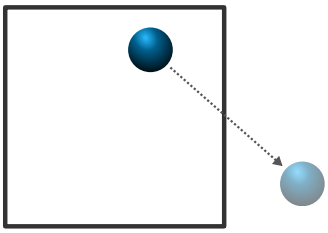
3.3 Boundary conditions

Boundary conditions is a crucial detail to decide upon when setting up a molecular dynamic experiment. The way we treat atoms at the boundary can have an immense effect on their behavior and how physically reasonable the results will

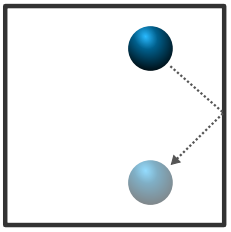
[1]In a bonded potential, bonds are not allowed to be broken and is not suited for studying diffusion, adhesion and similar problems.

be. There are several types of boundary conditions one may desire, and one may define custom conditions. A few examples are listed below.

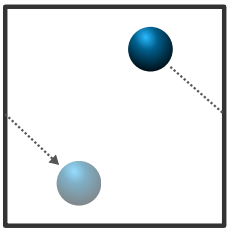
No boundary conditions. Particles are not subject to any special rules at any boundary. The system size may be regarded as infinite. This might be a reasonable choice when studying explosions for instance, or in experiments that is on a really short time scale.



Reflecting boundary conditions, which act as hard walls. Instead of passing through the boundary, the velocity component in the direction normal to the face of the boundary changes sign, thus causing the atoms to be confined within the simulation box.



Periodic boundary conditions, which allow atoms on separate sides of a boundary to interact through the boundary as if they were neighbors, and atoms crossing the boundary reappears on the other side of the simulation box. This is useful when studying bulk atoms of a material or materials that has a periodic structure.



The boundary conditions may include more details than only how to handle the trajectory of atoms intersecting the boundaries. For instance, with periodic boundaries atoms that are far apart are effectively close. These atoms should therefore interact, since their motion would seem artificial otherwise. Thus, the interpretation of atoms positions must in some cases be included in the boundary conditions. In order to interpret atoms positions in the case of periodic boundaries, one has to remember that an atom should not interact with multiple images of another atom. Which image, if not the original, must therefore be decided. Also, an atom should not interact with an image of itself. That could potentially cause strange correlations. The standard way of determining atoms positions is by using the *minimum image convention.*

3.3.1 Minimum image convention

The minimum image convention simply states that the position of the periodic image, or the original image, that is closest to the reference atom should be considered the atoms position. Example cases of 1D and 2D systems are shown in figure 3.2 and figure 3.3 respectively. In the examples we look at systems consisting of two atoms and we try to determine the minimum image of the smaller, blue atom with respect to the larger, red, reference atom. The original system is distinguishable from the periodic copies by the background color; they have a slightly darker background. Also, we only bother showing the periodic copies of the atom of consideration, the blue one. The systems depicted are of size L and $L \times L$ in the 1D and 2D cases respectively.

By studying figure 3.2 it becomes apparent that if the atom of consideration is closer to the reference atom than half the system length, the original image will be used to interpret its position. Otherwise, if it's farther apart than half the system length, a periodic image will be closer and therefor used as the atoms position. This principle is applicable at higher dimensions as well. We simply decompose the positions and check each dimension separately. For a system of arbitrary number of dimensions the minimum image convention is assured by using the progression of lisiting 3.1. Figure 3.3 illustrate how we decompose the positional separation of the atoms in the 2D case.

```
for (int k=0; k<numberOfDimension; k++){
  delta[k] = atom[j].pos[k] - atom[i].pos[k]
  if (delta[k] > L/2) {
    delta[k] = delta[k] - L
  }
  else if (delta[k] < -L/2){
    delta[k] = delta[k] + L
  }
}
```

Listing 3.1: Loop to compute the position of the closest periodic image of atom j with respect to the reference atom i.

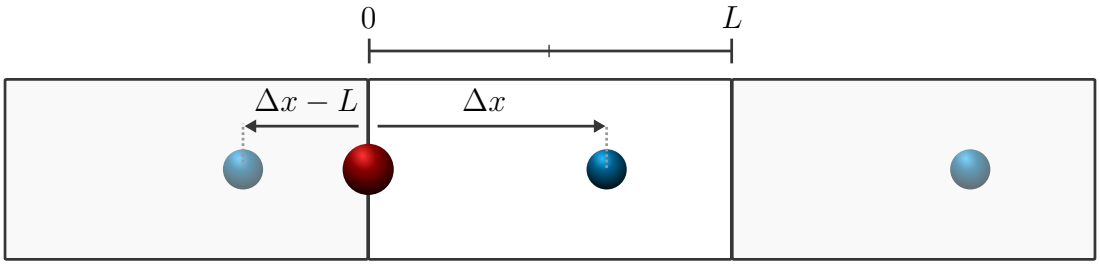


Figure 3.2: Minimum image convention in 1 dimension. If an atom is separated from the reference atom (red) by more than half the system length, $L/2$, the minimum image convention states that the distance between the two is the distance to the periodic copy, which is $|\Delta x - L|$. The white area indicates the system, while the gray areas are periodic replications of the system. Atoms in the gray areas are periodic images of the atom of consideration in the white area.

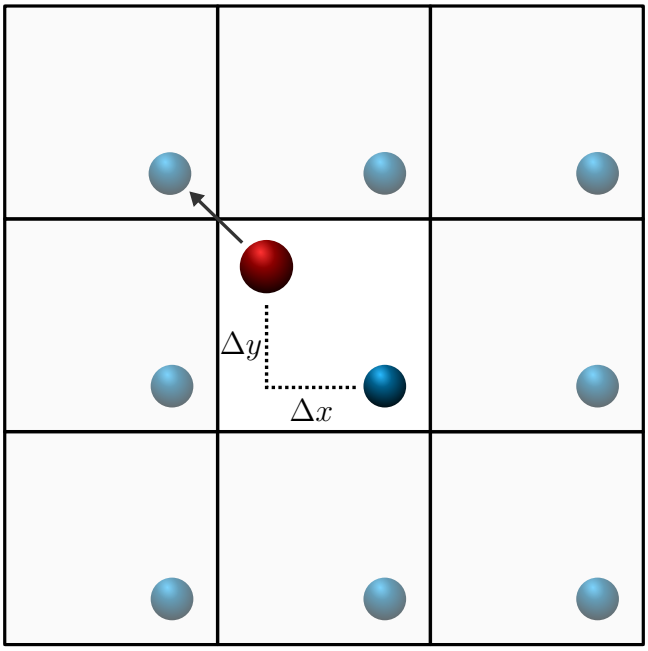


Figure 3.3: Minimum image convention in 2 dimensions. The position of an atom, with respect to the reference atom (red), is the position of the original or any of the replicated images that has the shortest distance to the reference atom. The minimum image is found by decomposing the atoms position, and carry out the The shortest distance in this case is marked with an arrow. Thus, when computing positional dependent quantities, it's the position of the northwest replica that will be considered as the position of the blue atom in this case.

3.4 Measuring physical quantities

3.4.1 Energy

In molecular dynamics we sample kinetic energy classically as

$$E_K = \frac{1}{2}\sum_{i=1}^{N} m_i {\mathbf{v}_i}^2. \tag{3.26}$$

The potential energy is off course dependent on the potential force field in use U, and is computed as

$$E_P = \sum_{i=1}^{N}\sum_{j>i}^{N} U(\mathbf{r}_i, \mathbf{r}_j). \tag{3.27}$$

3.4.2 Temperature

From thermodynamics we have that the kinetic energy, or total thermal energy is

$$E_k = \frac{f}{2} N k_b T, \tag{3.28}$$

for a system of N atoms with f kinetic degrees of freedom and temperature T. k_b is the Boltzmann constant. Since we assume point particles, we take only the translational degrees of freedom into account and ignore the rotational degrees of freedom. Since we assume Newtonian motion, the total kinetic energy in the system is also defined as in equation (3.26).

We can equate these to find the instantaneous temperature given as

$$T = \frac{\langle m_i {\mathbf{v_i}}^2 \rangle}{f k_b}. \tag{3.29}$$

Thus, we can approximate the instantaneous temperature using the atoms velocities. This value will fluctuate about the real value, so time-averaging is often used when measuring. In many cases it is desirable to control the temperature of the system, e.g. when using the canonical ensemble (NVT). Methods for doing this are called *thermostats*, and are described in section 3.5.

3.4.3 Pressure

There are many methods for computing the pressure of the system. Normally, for a N-body system of volume V and particle density $\rho = N/V$, it is taken from the virial equation for pressure.

$$P = \rho k_B T + \frac{1}{dV}\Big\langle \sum_{i<j} \mathbf{F}_{ij} \cdot \mathbf{r}_{ij} \Big\rangle, \tag{3.30}$$

where d is the number of dimensions, $\mathbf{F}_{ij}$ is the force contribution on atom i from atom j, positioned at $\mathbf{r}_{ij}$ relative to atom i. In this expression for the pressure we assume constant N,V and T, i.e. the canonical ensemble. The equation will not be the same for the micro-canonical ensemble. There are methods for converting averages from one ensemble to another [20], but that is outside the scope of this thesis.

3.5 Thermostats

Often, we want to control the temperature of our simulated system. This is done by modifying the velocity in equation (3.29). Methods for doing so are referred to as *thermostats*. There are many different types of thermostats, and they all have their strengths and weaknesses. We will look at two quite basic and different thermostats, as well as one that is common in professional use.

3.5.1 Berendsen

The Berendsen thermostat rescales the velocity of all atoms in the system so that the temperature approaches that of the surrounding heat bath. It rescales the atom's velocities by multiplying them by a scaling factor γ, given as

$$\gamma = \sqrt{1 + \frac{\Delta t}{\tau}\left(\frac{T_{bath}}{T} - 1\right)}, \tag{3.31}$$

where Δt is the time step length, and τ is a relaxation time, which tune the coupling to the heat bath. T_{bath} and T is the temperature of the heat bath and the current temperature of the system, respectively. Typically the relaxation time is set to $\tau \approx 20\Delta t$. Using $\tau = \Delta t$ would result in the velocities rescaling so the temperature would be exactly the target temperature, T_{bath}, every time step. This thermostat suppresses temperature fluctuations efficiently, rendering it well suitable for equilibration procedures. However, because it rescales the velocities, it produces dynamics that are inconsistent with the canonical ensemble.

3.5.2 Andersen

The Andersen thermostat simulates the coupling between the system and an imaginary heat bath as collisions between their atoms [21]. Atoms that collide are assigned a new normally distributed velocity with standard deviation $\sqrt{3k_B T_{bath}/m}$ about the target temperature, T_{bath}. The procedure of the thermostat goes as follows: For each atom in the system, we generate a random uniformly distributed number in the interval $[0, 1]$. If this number is less than $\Delta t/\tau$, the atom is considered to be colliding, and is assigned a new velocity. Here τ is

regarded as a collision time, and its value should be similar to τ in the Berendsen thermostat ($20\Delta t$). The Andersen thermostat is useful when equilibrating systems, but because randomly chosen particles are assigned randomly distributed velocities, it disturbs the dynamics of e.g. lattice vibrations. Also, this disturbance will decorrelate the system, rendering this thermostat ill-equipped when measuring e.g. diffusion coefficient. As a final note, if this thermostat is to be used, the newly assigned velocities may cause the systems center of mass to drift.

3.5.3 Nosé-Hoover

The most widely used thermostat is the Nosé-Hoover thermostat. It produces accurate dynamics and realistic constant temperature conditions. This thermostat differs from the Berendsen and Andersen thermostats in that it does not rescale the velocity of atoms. Instead it alters the hamiltonian of the system by adding a fictitious force of friction. The equation of motion is changed to

$$\mathbf{F}_i = -\nabla U_i(\mathbf{r}^N) - \xi m\mathbf{v_i}, \tag{3.32}$$

where ξ is a dynamic variable determining the strength of the thermostat. Due to this change, the integration scheme also has to be altered. As the details of this thermostat is somewhat extensive, I recommend reading *Understanding Molecular Simulation* by Frenkel & Smit [22] for a detailed derivation and explanation.

3.6 Efficiency improvements

The major part of the CPU time is spent in the force loop. At every time step we must recompute the force acting on each individual atom. When doing so, we should in theory include the contribution from all other atoms. Having a system consisting of N atoms would result in $N(N-1)/2 \propto N^2$ computations, if we apply Newton's third law. In this section we will look at the most fundamental efficiency improvements applied in molecular dynamics simulations.

3.6.1 Cut-off

Depending on the potential in use, the forces become negligible at certain distances. For instance if one uses the Lennard-Jones potential (introduced in section 3.2.1) the contributions are practically zero for atoms positioned at a distance $r \geq 2.5\sigma$. Therefore, during a simulation we choose to only account for the contributions from atoms closer than this *cut-off* distance, denoted r_c. Though the force at the cut-off range is practically zero, its not exactly zero. Without any alterations, particles intersecting the cut-off limit will experience an unphysical jerk, which may render the simulation unstable. To counteract this effect, we

shift (raise) the potential, ensuring it and its derivative to be zero at the cut-off. We use the following adjustment

$$V(r) = \begin{cases} V(r) - V(r_c) - \frac{\partial V(r)}{\partial r}\Big|_{r=r_c} (r - r_c) & , r \leq r_c \\ 0 & , r > r_c \end{cases} \tag{3.33}$$

which implies the force

$$F(r) = \begin{cases} F(r) = F(r) - F(r_c) & , r \leq r_c \\ 0 & , r > r_c \end{cases} \tag{3.34}$$

In practice we need to keep track of which atoms are within the cut-off range of each atom. This is achieved using cell lists and neighbor lists.

3.6.2 Cell lists and neighbor lists

The main purpose of the cell list is to make the building of neighbor lists more efficient. We need to check which atoms are neighboring atoms, but obviously we do not need to check the entire domain, since the cut-off length is relatively small. Therefore, we partition the system into several cubes with length equal to the cut-off length. We store the atoms contained by each cell in a *cell list*. Finally, when we build the neighbor lists we check only the atoms within the neighboring cells and those in the same cell, 27 cells in total. In this sense, by neighboring cells we mean any cells that share a face, edge or vertex with the reference cell. This is illustrated in figure 3.4. The pay-off of using cell- and neighbor lists is tremendous. Since we now, for each atom, only include contributions from atoms within a constant volume of size $4\pi {r_c}^3/3$, the number of contributions will only depend on the density, which is an intensive[2] property. Thus, the number of computations is reduced to $\mathcal{O}(N)$, which is an immense relief in computational expense!

3.6.3 Parallelization

It might be misguiding to refer to parallelization as an efficiency improvement, when on the contrary it most likely increases the CPU time usage. However, the real time consumed may be greatly decreased. It is intuitive that partitioning the work and processing these simultaneously will decrease the time, compared

[2]Physical property of a system that does not depend on the system size.

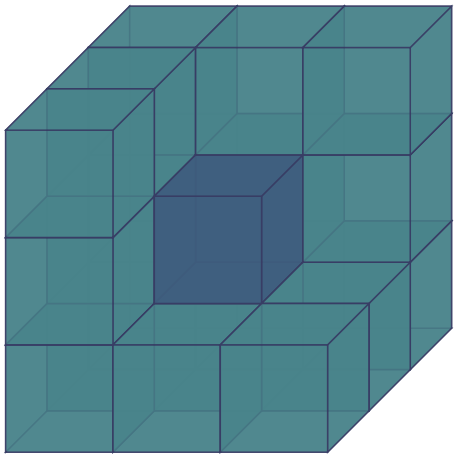


Figure 3.4: Illustrative figure of cells of concern when building the neighbor lists. The lighter cubes are considered neighboring cells; the darker cube is the cell containing the reference atom. The 7 cells in front of the dark cell are removed from the figure, but are also included.

to processing it serially. The speedup is defined as

$$S = \frac{T_s}{T_p}, \tag{3.35}$$

where T_s is the time used when executing the program on a single processor, and T_p the time used when running on p processors simultaneously. The time spent running a parallel implementation of a code using p processors is seldom trivially $T_p = T_s/p$. This is due to the fact that there is a certain amount of time used on *overhead*. This includes interprocess communications, idling and excess computations. In molecular dynamics simulations there is communication between processors when building the cell- and neighbor lists, and when computing thermodynamical properties such as energy, pressure, temperature, etc..

During this project we have mainly been using the local supercomputer at the department of physics at the University of Oslo. It provides users with the possibility to run up to 256 processes at once. Though, before doing so, it is considered good practice to check the speedup obtained by using several numbers of cores. In order to compute the speedup, we initialized a system containing $15 \times 15 \times 15$ unit cells of beta-cristobalite and saved it as a restart file. We then remotely ran the input script shown in Listings 3.2 from the supercomputer using 1, 2, 4, 8, 16, 32 and 64 processors in the same fashion as shown in Listing 3.3. The resulting speedup of using the respective number of processors is plotted in figure 3.5.

```
1 include "system.in.init"
2 read_restart ${filename}
3 include "system.in.settings"
```

```
4
5  variable N equal 20000
6  variable T equal 293
7
8  neighbor    0.3 bin
9  neigh_modify delay 10
10 timestep    0.002
11
12 fix nvt all nvt temp ${T} ${T} 1.0
13
14 run ${N}
```

Listing 3.2: LAMMPS input script executed using several numbers of processors, and timed separately.

```
1  mpirun -n 8 lmp_mpi -in speedup.in -var filename
       speedup.restart
```

Listing 3.3: Command used to execute the input script speedup.in on 8 parallel processors and set the filename variable to speedup.restart.

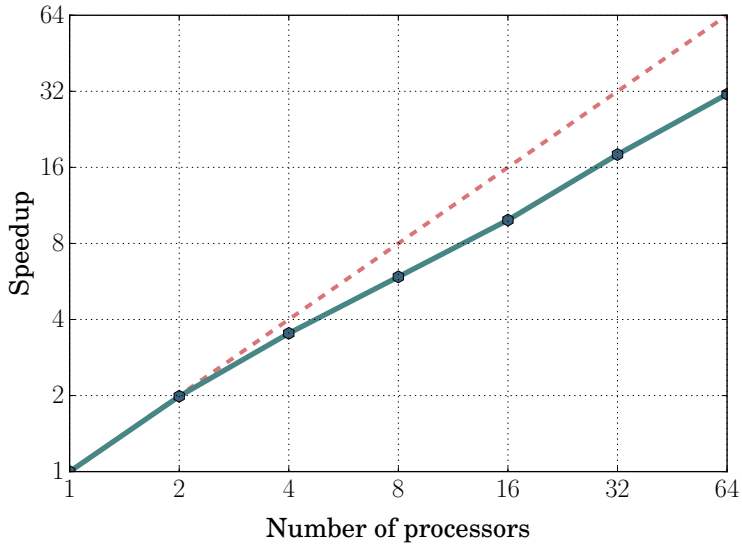


Figure 3.5: Strong scaling of parallel performance of LAMMPS using 81000 atoms, evenly distributed in a cubical simulation box, running on the local super computer. The marked points are measured speedup, while the dashed line represents ideal speedup ($S = T_s/T_p = p$).

The result indicate that the speedup is in fact not simply $S = p$. Nevertheless, we clearly see the advantage of using 64 processors in parallel as opposed to a single processor serially. We can finish a job that would have taken an hour in two

minutes! Also, we clearly see that the initial claim holds; this is not more efficient when regarding CPU time. In fact this result suggest that using 1 processor is twice as energy efficient as using 64.

Chapter 4

LAMMPS

LAMMPS is an acronym for *Large-scale Atomic/Molecular Massively Parallel Simulator*. It is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. Its development began in the mid 1990s at Sandia National Laboratories, with funding from the U.S. Department of Energy. It was a cooperative project between two DOE labs and three private companies. The development is still ongoing and contributions are revised thoroughly. Today LAMMPS is an open-source code with extensive and user friendly documentation. This is one of the main reasons why we have chosen to use LAMMPS as opposed to other molecular dynamics software.

In this chapter we will be presented with a guide to install LAMMPS, a brief explanation of commands and syntax, and an example of a valid input script.

4.1 Installation

Installing LAMMPS is a fairly simple procedure if only the basic settings are needed.

4.1.1 Linux

Users with a Unix based OS may download the LAMMPS distribution as a tarball from LAMMPS' download page[1] and then unpack it from the command line.

```
gunzip filename.tar.gz
tar xzvf filename.tar
```

The user should then change directory into `/path/to/lammps/src/`, and execute the following commands in order to list available packages.

```
make package-status
```

[1] http://lammps.sandia.gov/download.html

Installing specific packages is accomplished as shown below.

```
1 make yes-molecule yes-manybody yes-python yes-rigid
```

The above example installs the packages *molecule, manybody, python and rigid*. Next, the user can build LAMMPS using either of the lines below. Assuming the user has MPI installed, line 2 utilizes 8 processors to make the resulting executable compatible with parallelization in MPI. The argument `-j8` is optional, but saves time.

```
1 make serial
2 make -j8 mpi
```

At this point there should be an executable in the `/path/to/lammps/src/` directory named `lmp_serial` or `lmp_mpi`, depending on the previous choice. These are now ready to run. To use it one has to point to this file from the command line at every run. It may be practical to set up a symlink as shown below.

```
1 sudo ln -s /path/to/lammps/src/lmp_mpi
     /usr/local/bin/lmp_mpi
```

Then the executable will be available as `lmp_serial` or `lmp_mpi` from any directory.

4.1.2 Mac OS X with Homebrew

Mac users can follow the procedure described above, however they may also install even easier using *Homebrew*[2].

```
1 brew tap homebrew/science
2 brew install lammps              # serial version
3 brew install lammps --with-mpi   # mpi support
```

Where the user obviously should choose either line 2 or line 3, depending on whether the user wants MPI comparability. This will install an executable named "lammps", a python module named "lammps", and resources with standard packages. This is basically it. LAMMPS is now ready to run, though not all packages are installed. The location of the resources and available packages can be found using the following command.

```
1 brew info lammps
```

Specific packages are available as options, and may be installed using the following syntax.

```
1 brew install lammps --enable-manybody
```

[2] http://brew.sh/

In the example shown we installed the package manybody.

4.2 Input scripts

The executable made in the previous section can be used to read so-called input scripts. The input scripts contain LAMMPS commands to configure the simulation. This includes all settings and actions. This is naturally partitioned into two sections: *system configurations* and *run-time commands*. The scripts are executed on 8 CPU's in parallel as

```
mpirun -n 8 lmp_mpi -in in.system
```

Despite the widely common convention of letting the file type be determined by the letters following the punctuation mark, it's common in the LAMMPS manuals to name the input scripts with the name last. i.e. `in.name`. Luckily, both conventions are meretricious since the executable accepts any name/type for the input script.

4.2.1 System configurations

The first part of every input script contains information of the system properties. This includes: size, number of dimensions, boundary conditions, potentials, unit convention, information on the containing atoms, time step length, neighbor list update frequency and possibly a lot more. Listing 4.1 show the configurations that has been used for the entire duration of this project. Below there is a description of each command and its purpose.

```
units         metal
boundary      p p p
atom_style    atomic
read_data     "system.data"
pair_style    vashishta
neighbor      0.3 bin
neigh_modify  delay 10
timestep      0.002
```

Listing 4.1: Typical system configurations applied in this project.

`units` determines what unit convention should be used for the simulation. It might not seem like a difficult decision, but if you import data from a file for instance, it is crucial that LAMMPS interpret the data correctly. In this project we have chosen *metal* as the unit convention. The units are therefore as shown in table A.1.

`boundary` sets the style of boundaries for the simulation box in each dimension. There are four options: `p`, `f`, `s` and `m`. In the example above, we have chosen to use periodic boundaries through all the faces of the simulation box. It's possible to set different conditions in separate dimensions, e.g. `boundary p f p` sets fixed boundaries on the faces normal to the y-direction. One can even set different conditions on the two faces in the same dimension by using two letters, e.g. `boundary p fs p`, where the first letter indicates the boundary to be used on the *low face* and the last on the *high face*.

`atom_style` tells LAMMPS the structure of atom related data stored in a data file. This includes information about particle types, positions, charges, mass, bonds, angles, and potentially more, depending on the `atom_style`.

`read_data` reads a data file containing information as described above and additional information about the system, such as its size and shape. This is one of three ways to distribute initial atom positions. Another is the `read_restart` command, which is used extensively to load saved states from restart files. The last option is to use `create atoms`, which distributes atoms in a predefined way, i.e. in a lattice or a random collection. To use this last option, one has to first create a simulation box using the `create box` command.

`pair_style` determines which potential to use in the simulation.

`neighbor` sets the additional range of the neighbor list cutoff, and the method of constructing the neighbor lists. The consequence of having a larger range for the neighbor lists is that there are more particles to check for force contribution, however, we don't have to update the neighbor lists as often.

`neigh_modify` determines how often to update the neighbor lists.

`timestep` self-explanatory.

4.2.2 Run-time commands

Variables

LAMMPS offer the ability to store values in variables. These can be constants or dependent on other variables, like the time step for instance. Declaration of variables is done using the following syntax.

```
1 variable A equal 1000
2 variable B equal step/${A}
3 variable C equal ${B}
```

```
4 variable D equal v_B
```

Listing 4.2: Declaration of variables.

All of the above are valid variables. The `step` variable is a LAMMPS standard for current time step. We attend for `B` to be a linear function running from 0 to 1 over the course of `A` time steps. `C` evaluates `B` at the time step it is declared and returns that value whenever called upon. Thus, `C` will be a constant value. `D` will evaluate `B` whenever called upon, and return the current value of variable `B`. The variables `C` and `D` are of course superfluous, since we could only use `B` directly. The user can use math operations on variables, such as: addition, subtraction, multiplication, division, as well as square roots, logarithms, exponentials, and lots more.

Region & Group

In almost all simulations it is necessary to define regions and groups in order to give certain parts of the domain special properties. If the user wish to create a sphere of atoms, the typical process is to start from a block of atoms, define a spherical region in the block and remove exterior atoms. The regions specified can take the shapes: block, cone, cylinder, plane, prism or sphere, and regions can be combined. An example of removing atoms within a combined region is shown in listing 5.4. This example also show how to assign a group to atoms within a region. Each atom can be part of up to 32 different groups at a time. These are stored in a 32-bit integer. The user can do tons of actions on groups. Most computes and fixes act on specific groups. There is only one standard group in LAMMPS; the group `all`.

Computes

A compute defines a computation that will be performed on a group of atoms. This does not affect the dynamics in any way. The returned values are instantaneous. That is, they are computed from information about atoms on the current time step. The returned values can be global, local or per-atom quantities. These can be retrieved by an output commands using the following syntax:

```
1 c_computeID
2 c_computeID[1]
3 c_computeID[*][2]
```

The first alternative would return all values (scalars, vectors, arrays) computed. The second would return only the first element of a vector, or row of an array, and the last one would return the second element of all rows of an array. Note that LAMMPS counts from 1, which also deviate from other established standards. In section 4.2.3 we will see this in several example, where we refer to the `compute`

`com` command.

```
compute comID groupID com
```

This computes the position of the center of mass of all atoms within the specified group, and stores the result in a 3-element vector.

Fix

A fix is an operation that's applied to the system during time iteration. Examples include time evolution of atoms, i.e. updating their positions and velocities, using thermostats, applying external force on atoms, averaging values, etc. There are numerous fixes and computes in LAMMPS, and you can easily add new ones if you knows the structure of the framework. Some Fixes also compute and store values that can be retrieved by output commands. For instance the `fix addforce` command

```
fix addforceID groupID addforce fx fy fz
```

adds a given force to all atoms within the specified group. This `fix` stores the total force acting on the group before the force was added. This is stored as a 3-element vector. Its values can be obtained in the same manner as for computes:

```
f_fixID
f_fixID[1]
f_fixID[*][2]
```

4.2.3 Output

In order to benefit from our simulations, we need to be able to extract the data of interest somehow. Here we will present the four most basic types of output.

Thermodynamic output

prints computed values to the screen and logfile every N time steps. It is activated by the command

```
thermo N
```

where N should be replaced by the desired frequency of the output. This can be a variable. The default quantities given in the output are: time step, temperature, pairwise energy, molecular energy, total energy and pressure. The user can specify what values should be included by using the `thermo_style` command with the syntax of the following example.

```
thermo_style custom step c_comID v_Fz f_addforceID[3]
```

This line specifies that when the thermodynamic output is printed, it should contain the time step, all quantities returned by the compute `comID`, the current value of the variable `Fz` and the third element of the vector retrieved by the fix `addforceID`. The naming of the compute- and fix names can be whatever desired, even numbers. I just find it convenient to name them by what they represent.

Dump

Dump files store data about the state of atoms in a specified group with the given frequency. Below we illustrate two quite different dump commands.

```
1 dump dumpID1 all atom 100 name.dump
2 dump dumpID2 groupID custom 50 name*.dump id x y vx fx
```

Line 1 creates a file named `name.dump`. Every 100 time steps it writes data to this file in the *atom* format, i.e. `id type xs ys zs`, where `xs`, `ys` and `zs` are scaled coordinates relative to the size of the simulation box. Line 2 creates a file for every time step, where the asterisk in the name is replaced by the current time step. Also we define the format of the output to be unscaled x- and y-coordinates, as well as velocity and force in the x-direction.

Common visualization software can read most of the predefined formats in LAMMPS: atom, xyz, cfg, etc..

Fix

Certain fixes can also write specified quantities to files. For instance the commands

```
1 fix timeAvgID all ave/time 100 5 1000 c_comID file name.avg
2 fix spatialAvgID all ave/chunk 100 10 1000 chunkID c_comID
     file vel.profile
```

do respectfully time- and spatial averaging of the values returned by the compute `comID`. The numbers indicate how many values should be sampled, number of time steps between each sample, and how often to write this average to file. Note that the last command refers to a `compute chunk`, which governs how to grid the system. There is also a `fix print` command that writes single-line output to screen or file with a prescribed frequency.

```
1 fix printID all print 100 "z-component of COM: c_comID[3]"
```

This acts practically like the `thermo` command.

Restart files

In many cases it is practical to save certain checkpoints in a simulation, in order to be able to start a new simulation from said point. The `restart` command does just that.

```
1 restart N name.*.restart
```

This command saves the state of the system every `N` time step to individual restart files, distinguishable by the number representing the time step of the save. By "state of the system" we mean positions and velocities of all atoms, information of what groups each atom belongs to, the size and shape of the simulation box, boundary conditions, potential, etc.. When starting a new simulation, one of these "snapshots" can be loaded using the `read_restart` command as illustrated below.

```
1 read_restart name.timeStep.restart
```

4.3 Visualization

The LAMMPS package does not provide high-quality visualization software. However, the default formats of the dump command are well known to most of the ones out there. A couple of so-called high-quality visualization tools are: VMD, AtomEye, Ovito, ParaView and PyMol. All of the listed software are able to read `.xyz`-files, and also the standard `.dump`-files produced by LAMMPS. Typically, each atom is shown as a sphere, with a given radii and color, dependent on the atom type. One can then zoom, rotate and move the camera position. The author has only experience with VMD and Ovito among the listed software, and find Ovito to have the most user friendly interface. The user has four viewports, one along each dimension and one that is a perspective view. It's possible to color code atoms based on position, remove/hide all atoms outside a region, and do analysis on the fly. In addition it has an own scripting tool compatible with python.

4.3.1 Atomify

The user has to relate to several programs when working with LAMMPS. A text editor is used to create/edit scripts, the terminal is used to run the simulations, visualization software as the stated above are used to visualize, and lastly, computed values (which also demand scripts to obtain) are usually plotted using Python or Matlab. A software developed by Anders Hafreager and Svenn-Arne Dragly at the University of Oslo, was recently released in Mac App store. They aim to ease the tedious workflow of working with LAMMPS, by in-

troducing Atomify; a high performance live visualizer for LAMMPS simulations. It allows the user to edit input scripts, and by the push of a button start the simulation. The simulation is visualized live as it processes. Computes and fixes can be monitored, also live, and specific groups can be hidden or shown as one pleases. The possibility of having an all-in-one solution has been of great help to the author, especially when learning to use LAMMPS, and plotting the returned values from computes or fixes. This is a software that we highly recommend for everyone new to LAMMPS.

Chapter 5

Preparing a molecular dynamics simulation

We wish to construct a system consisting of two bodies made out of silica: a parallelepiped slab and a sphere cap. In order to do this we need to generate the spatial position coordinates (x,y,z) of every single atom. Considering that we are making a system consisting of about 10^5 atoms, this is obviously not done manually. We have chosen to use a tool named *Moltemplate*[1], which is included in the LAMMPS distribution.

The main idea is to manually enter the coordinates of only the atoms in a unit cell of the material one wish to generate, and then simply copy this unit cell wherever desired. In addition to atoms positions it will generate files containing data such as which atoms they share bonds with, if any, and angles between such bonds.

5.1 Silica

Silica is a chemical compound also known as Silicon dioxide, having the chemical formula SiO_2. It has several polymorph structures, the most common being quartz, which is one of the most abundant minerals in the Earth's crust. Other polymorphs include cristobalite, tridymite, coesite and more. If we wish to obtain amorphous silica it's insignificant which one we choose. Once the material is melted, it is indifferent which configuration we started from, as long as the density is correct. However, we may wish to keep the crystalline structure, in which case the choice does matter. In this project we will build the constituents of the system from a type of cristobalite named β-cristobalite. This is mainly because it has a simple structure and a cubical unit cell.

[1] http://www.moltemplate.org/index.html

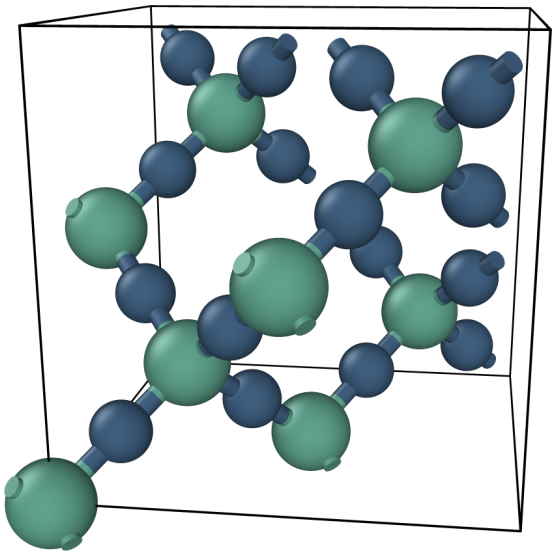


Figure 5.1: Unit cell of β-cristobalite. Light and dark spheres represent silicon and oxygen atoms respectively. The unit cell is cubical, with edges of length 7.12Å.

5.1.1 Unit cell of β-cristobalite

In order to construct the unit cell of a material, one should look up the coordinates of the atoms in a crystallography database. We have used the unit cell of β-cristobalite found at *Crystallography Open Database*[2]. At this site one can download a `.cif`-file (Crystallographic Information Framework) containing information about the spatial positions of each atom, the length of the unit cell edges and angles between faces of the cell. In the case of β-cristobalite the unit cell is cubical with edges of length 7.12Å. It contains 24 atoms: 8 silicon atoms and 16 oxygen atoms. The density of the unit cell can easily be computed and is 2.2114 g/cm^3. The format of the `.cif`-file is not easily readable. To extract the information we have used a tool named `cif2file`[3], developed by Torbjörn Björkman.

5.2 Constructing a crystal

The coordinates gotten from the `.cif`-file can now be implemented into *moltemplate* together with whatever bond and angle data required by the potential. In our simulations we will use the Vashishta potential, which does not require these, as it is non-bonded. Moltemplate has its own structure and syntax. The first step to build up a larger material is, as mentioned, to create the unit cell. Data con-

[2] http://www.crystallography.net/cod/1010944.html
[3] http://www.cif2cell.com-about.com/

cerning the unit cell are placed in a `.lt`-file, which is readable by Moltemplate. Such a file is shown in Listing 5.1. For a more profound understanding of the structure and syntax of these files, the reader is advised to read the moltemplate manual[4].

```
# File "beta-cristobalite.lt"

beta-cristobalite {
  write("Data Atoms") {
    $atom:Si1    @atom:Si    0.00    0.00    0.00
    $atom:Si2    @atom:Si    0.00    3.56    3.56
    $atom:Si3    @atom:Si    1.78    1.78    1.78
    $atom:Si4    @atom:Si    3.56    0.00    3.56
    $atom:Si5    @atom:Si    1.78    5.34    5.34
    $atom:Si6    @atom:Si    5.34    5.34    1.78
    $atom:Si7    @atom:Si    3.56    3.56    0.00
    $atom:Si8    @atom:Si    5.34    1.78    5.34
    $atom:O1     @atom:O     0.89    0.89    0.89
    $atom:O2     @atom:O     6.23    4.45    2.67
    $atom:O3     @atom:O     2.67    2.67    0.89
    $atom:O4     @atom:O     4.45    0.89    4.45
    $atom:O5     @atom:O     0.89    4.45    4.45
    $atom:O6     @atom:O     4.45    4.45    0.89
    $atom:O7     @atom:O     2.67    6.23    4.45
    $atom:O8     @atom:O     2.67    0.89    2.67
    $atom:O9     @atom:O     4.45    2.67    6.23
    $atom:O10    @atom:O     6.23    2.67    4.45
    $atom:O11    @atom:O     2.67    4.45    6.23
    $atom:O12    @atom:O     0.89    6.23    6.23
    $atom:O13    @atom:O     0.89    2.67    2.67
    $atom:O14    @atom:O     4.45    6.23    2.67
    $atom:O15    @atom:O     6.23    6.23    0.89
    $atom:O16    @atom:O     6.23    0.89    6.23
  }

  write_once("Data Masses") {
    @atom:Si 28.0855
    @atom:O  15.9994
  }
} # end definition of beta-cristobalite molecule type
```

Listing 5.1: Typical Moltemplate file containing unit cell data. The columns of the "Data Atoms" section hold, from left to right, information of atom ID, atom type, x-, y- and z-position. The "Data Masses" section stores the weight of silicon and oxygen atoms in atomic mass units.

We use the unit cells as building blocks, placing them concurrently until we have a crystal of the desired size. For example purposes, we generate a cube of

[4] http://www.moltemplate.org/doc/moltemplate_manual.pdf

$15 \times 15 \times 15$ unit cells, i.e. 81000 atoms. This is done in line 18-20 of listing 5.2 below.

```
1  # File "system.lt"
2
3  write_once("In Init") {
4    units          metal
5    boundary       p p p
6    atom_style     atomic
7    pair_style     vashishta
8  }
9
10 write_once("In Settings") {
11   pair_coeff   * *  SiO2.vashishta Si O
12   pair_modify table 16
13   pair_modify tabinner 0.1
14 }
15
16 import "beta-cristobalite.lt"
17
18 Slab = new beta-cristobalite [15].move(7.12, 0.00, 0.00)
19                              [15].move(0.00, 7.12, 0.00)
20                              [15].move(0.00, 0.00, 7.12)
21
22 write_once("Data Boundary") {
23   0.0   106.8   xlo   xhi
24   0.0   106.8   ylo   yhi
25   0.0   106.8   zlo   zhi
26 }
```

Listing 5.2: Moltemplate script to build a cube out of 15^3 unitcells of β-cristobalite (81000 atoms), and also specify system configurations (see section 4.2.1). The "write_once" sections create the files containing their contents. Line 17 imports the unit cell shown in listing 5.1, and line 19-21 copies it concurrently in a $15 \times 15 \times 15$ cube.

The *write_ once* sections with "In" as the first word of its argument, creates the files `system.in.init` and `system.in.settings`, which contain exactly what is shown in the snippet. They are not necessary, but help making the LAMMPS input script cleaner, since we can `import` these files instead of having all the configurations in the input script file. The last *write_ once* section with "Data" in its argument, incorporates the specified size of the simulation box in the `system.data` file. Line 19 creates a new unit cell at every point separated by 7.12Å in all three dimensions. The positions, and other properties defined by the *atomstyle*, of all the distributed atoms are stored in the file `system.data` as well.

The Moltemplate script can be run from the command line as

```
1  moltemplate -atomstyle "atomic" system.lt
```

Once complete, we can load the configurations and atomic data into our LAMMPS input script simply by including the file `system.in`.

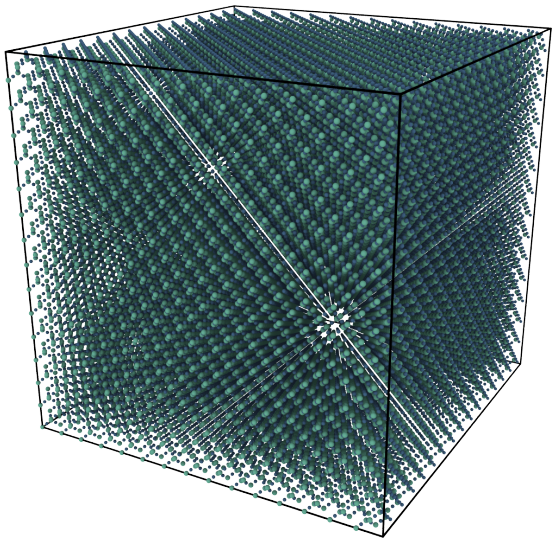


Figure 5.2: System built from $15 \times 15 \times 15$ unit cells of b-cristobalite.

5.3 Melting point

A simple verification we can do is to estimate the melting point of the material we are using. There are several factors that may affect the result, such as density, structure and of course parameters in the potential model. When increasing the temperature of the system that is in a solid state, we will eventually reach the melting point. At the phase transition from a solid state to a liquid, the atoms of the silica will have energy great enough to break the interatomic bonds. They will break loose from their regular arrangement and move about much more freely. An approach to computing the melting point is therefore to systematically increase the temperature stepwise and sample the mean square displacement of the atoms at each temperature level. The mean square displacement is the average of the square of the displacement every atom has from its initial position. It can be expressed as:

$$\langle r^2(t) \rangle = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{r}_i(t) - \mathbf{r}_i(0) \right)^2 . \tag{5.1}$$

where $\mathbf{r}_i(t)$ is the position of atom i at time t and N is the total number of atoms. The mean square displacement is expected to relate to the diffusion of atoms through

$$D = \frac{\langle r^2(t) \rangle}{2dt}, \tag{5.2}$$

where D is the diffusion coefficient, and d is the number of dimensions [23].

In practice the mean square displacement was computed using a standard *compute* in LAMMPS, namely the *msd compute*[5]. At every time step it stores a vector of 4 elements. The first 3 are the squared dx, dy and dz displacements averaged over the atoms of the specific group, while the 4th is the total mean square displacement for the specific group, i.e. $(dx^2 + dy^2 + dz^2)$. The $15 \times 15 \times 15$ system is sufficient for this test. The procedure is rather simple. For β-cristobalite we expect the melting point to be about 1986K[6], so we start out at 1500K. After equilibration, the following steps are repeated until we reach a target temperature.

- equilibrate for the current temperature
- compute msd for N time steps
- increase the temperature by a given step size

```
label meltingLoop
variable i loop 11
    fix nvtID all nvt temp ${T} ${T} 1.0
    run ${N}
    compute msdID all msd
    fix msdDumpID all ave/time 1 1 3 c_msdID[4] file
       msd_N${N}_T${T}.txt
    run ${N}
    uncompute msdID
    unfix nvtID
    unfix msdDumpID
    variable T equal ${T}+50
next i
jump SELF meltingLoop
```

Listing 5.3: Section of LAMMPS script illustration how to stepwise increase temperature and compute the mean square displacement of atoms.

[5] http://lammps.sandia.gov/doc/compute_msd.html
[6] https://en.wikipedia.org/wiki/Cristobalite

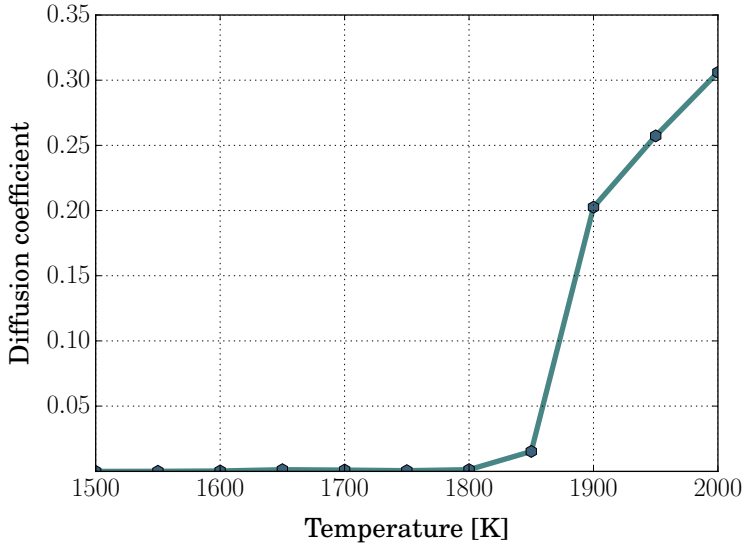


Figure 5.3: Measured diffusion coefficient as a function of increasing temperature. The face transition occurs where we see a rapid change in the behavior. Here we can estimate the measurement somewhere between 1850K and 1900K. These measurements were obtained from a cubical simulation box containing 15^3 unit cells of β-cristobalite (81000 atoms) evenly distributed, and with periodic boundary conditions.

This result infer a melting point somewhere in the range 1800K-1900K. Believe it or not, this is actually a pass on the test! In fact, anything deviating less than 20% from the experimental results is considered a pass, as stated by the developers of the potential. Unfortunately, this also show that there are some qualitative detail that is lost, and that when using this potential we should probably focus on the quantitative behavior.

5.4 Shaping the system

The cube of silica can be carved however we like by defining regions from which we delete the containing atoms. In LAMMPS this is done using the `region`, `union`, `intersect` and `delete_atoms` commands. Our implementation is stated in Listing 5.4, which is very simple due to the way we are going to treat the boundary conditions. We start out by defining a spherical region labeled `sphereRegion`, described by the xyz-coordinates of its center and a radii. The atoms within this region are assigned to a group labeled `sphereGroup`. Next, we define a cuboid (block) region named `slabRegion`, described by the position of its faces in x-, y- and z-direction. The atoms within this region are assigned to a group, which we label `slabGroup`. We combine these two regions using the `union` command and label the region outside of these regions `outRegion`. Finally, we delete the

atoms that are not in the sphere nor the slab; we delete the ones contained by `outRegion`.

```
region sphereRegion sphere 53.4 53.4 226.8 150
group  sphereGroup region sphereRegion

region slabRegion block 0 INF 0 INF 0 35.6
group  slabGroup region slabRegion

region outRegion union 2 sphereRegion slabRegion side out
delete_atoms region outRegion
```

Listing 5.4: Defining regions to keep or delete from a system of dimensions $106.8 \times 106.8 \times 106.8$ Å.

For the purpose of deleting atoms, the creation of groups was superfluous. However, at a later stage we will utilize them and this is an appropriate place for them to be assigned. The appliance of the script in Listing 5.4 on the system is shown in figure 5.4, where our perspective is along the y-axis.

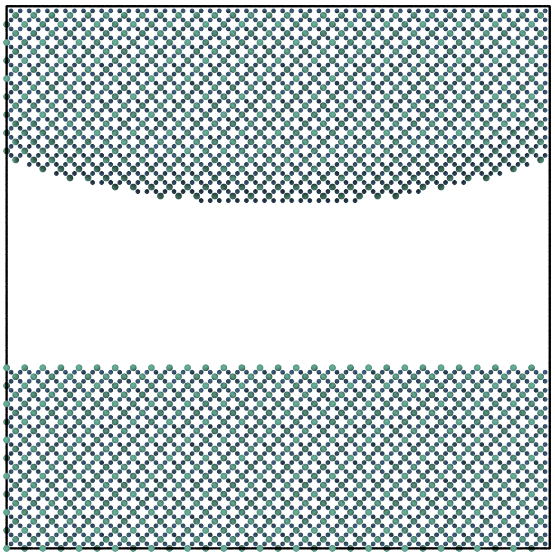


Figure 5.4: xz-perspective on a system built from $15 \times 15 \times 15$ unit cells of β-cristobalite, with certain regions carves out. This is a result from applying Listing 5.4 to the system shown in figure 5.2. The top shape is a sphere cap, while the bottom is a slab.

5.5 Moving the sphere towards the slab

We wish to push the sphere down onto the substrate in order to create a deformation on the substrate. There are many approaches to accomplish this. One could apply an external force, pushing the sphere with a controlled force. An-

other is to move the sphere at a controlled, constant velocity. We will use both of these methods on separate parts of the project. For the first part, we have settled on the following strategy: We apply periodic boundary conditions in all three dimensions. Secondly, we freeze the atoms at the bottom of the substrate so that their positions are fixed. They still interact with other atoms, but we restrain them from moving. This will, due to the periodic boundary conditions, allow us to consider the sphere cap and the slab as if they are not connected to each other, but to independent blocks of silica glass. For every N time steps we decrease the height of the system, while remapping the positions of the atoms. The remapping is a very important procedure. It ensures that we do not lose any atoms that else-wise would be lost when moving the z-boundary. A side-effect of the remapping is that the atoms in the system will be somewhat compressed in the z-direction. Though, if we do the compression slowly, this will not be of any concern.

One technique to fix atom positions is to explicitly set their velocity and force to zero at every single time step. A more efficient method is to simply omit the atoms of consideration from time-integration. This way we do not bother computing the force acting on them, updating their velocity or position at all! To do this, we simply create a group that contains all atoms except the once who should be fixed. Listing 5.5 show how to tell LAMMPS to only time-integrate the atoms in the group `excludingBT`, which excludes the bottom and top regions of our system. The omitted atoms are still considered to be present, we just don't bother updating their movement.

To deform/resize the system, we may use the command shown in listing 5.6. This command changes the position of the upper boundary of in the z-dimension by the length `compressionLength` during a `run`. Thus if we choose the run to last N time steps, the boundary will move with the velocity

$$\mathbf{v}_z = -\frac{\texttt{compressionLength}}{N \cdot 0.002} \quad [\text{Å/ps}] \tag{5.3}$$

since our time steps are 0.002 ps.

```
1 fix nvtID excludeBT nvt temp ${T} ${T} 1.0
```

Listing 5.5: Time-integrating only atoms in a specified group, `excludeBT`, effectively fixing all others.

```
1 fix ID all deform 1 z delta 0 -${compressionLength} remap x
```

Listing 5.6: LAMMPS command for deforming the simulation box, and remaping atom positions.

Part II

Numerical simulations and results

Chapter 6

Computing contact forces

Contact forces are forces acting between two bodies at the area of contact. They can be decomposed into normal an shear forces The normal force is defined as the force exerted on an object that is perpendicular to the contact surface, while shear force is the component lying tangential to the contact surface. In this chapter we will make an attempt to find the distribution of the normal and shear forces acting between a spherical cap indenting a substrate. This is not a trivial thing to compute in molecular dynamics. The reason is that we have to be able to define the contact surface, which is not always easy. For solid state physics there are several algorithms which attempt to do this by searching for areas with structural discrepancies or dislocations [24]. As time was a limiting factor, a simpler approach has been used. Secondly, we must distinguish contact forces from non-contact forces. As a matter of fact, to obtain the contact forces we have expanded the LAMMPS library by creating a custom compute class, since there currently is no such compute available. For the sake of completeness, the details of that procedure will be described.

Our strategy is simple, but not necessarily easy. First of, we divide the system into a 2D-grid in the xy-plane. Secondly, we compute the time-averaged force exerted on one body from another within the region of each cell in the grid. We approximate the slope of the contact surface within the cells using a least squares regression method on the atoms positioned at the surface. Finally, we project the average force of a cell onto the corresponding normal vector of the approximated surface.

The results will be posed graphically at the end of the chapter, as 3-dimensional plots in polar coordinates, with the corresponding radial distributions. We will point out observations, but the main discussion is found in section 8.1.

6.1 Creating a custom compute

A *compute* is a LAMMPS command that defines a computation that will be performed on a group of atoms. It does not affect the dynamics in any way, but simply uses already available information from the system. This can be atom positions, velocity, force, mass, or other properties like size of the simulation box. The *computes* produce instantaneous values, using information about the atoms on the current time step[1]. In LAMMPS there are more than 100 computes and chances are, they already have what you're looking for. If not, one might treat the data from other computes in some way to get the desired information. However, if there are no compute command that does the desired task, it is possible to create an own custom class and thereby expanding the LAMMPS library. In order to compute the normal forces acting on the sphere, we have written a custom compute class. The objective was for the class to save the force acting on each atom in one group from atoms of another group. In this section we will try to give a brief summary on how this was done.

6.1.1 Look for similar computes

Obviously, before writing any code we should know what we want the compute to calculate and how this should be done. Before starting off with a blank sheet in the editor, one should definitely search for similar computes in LAMMPS. This can potentially save hours of hard work! Our case serves as a good example. There is a compute named *group/group*[2] which computes the total energy and force interaction between two groups of atoms. This is almost what we want, but we need to know the force acting on each atom from atoms of other groups. Also, It should work with the Vashishta potential, which *group/group* currently does not. Thus, there are minor modifications needed and because of the similarities we chose to make our compute a subclass of this one.

6.1.2 Creating the class

All computes in LAMMPS are subclasses of the class named *compute*. From this superclass they inherit a bunch of variables, functions and flags, which the user may decide to set. Functions are of course declared in the header file, while variables and flags are set in the source file. The source code of the *group/group* compute is included in the LAMMPS distribution. Since we will be making a subclass of it, we change the *private* property to *protected* so that we have access to all the variables and functions, from our subclass.

We start out by creating a header file and decide upon a name for our class.

[1] http://lammps.sandia.gov/doc/compute.html
[2] http://lammps.sandia.gov/doc/compute_group_group.html

We have chosen the name *group/group/atom* since it is basically a per-atom version of the already existing compute *group/group*. A trailing *"/atom"* is the common naming convention of per-atom computes. A complete header-file is shown in Listing 6.1 and explained in detail below.

```
#ifdef COMPUTE_CLASS
ComputeStyle(group/group/atom,ComputeGroupGroupAtom)
#else

#ifndef LMP_COMPUTE_GROUP_GROUP_ATOM_H
#define LMP_COMPUTE_GROUP_GROUP_ATOM_H

#include "compute.h"
#include "compute_group_group.h"

namespace LAMMPS_NS {

class ComputeGroupGroupAtom : public ComputeGroupGroup {
 public:
  ComputeGroupGroupAtom(class LAMMPS *, int, char **);
  ~ComputeGroupGroupAtom();
  void compute_peratom() override;
  int nmax;
  double **carray;

 private:
  void pair_contribution() override;
};
}
#endif
#endif
```

Listing 6.1: Header file of our new compute: `compute_group_group_atom.h`.

As can be seen, there are not many additions to the variables and functions already existing in the superclass, as the main difference lies in the structure we store the computed values.

- `ComputeStyle` defines the command to be used in the input script to be `group/group/atom`, and the name to be `ComputeGroupGroupAtom`. The name is of no importance though.
- `nmax` is the number of atoms which are subject to a non zero force from atoms of another group at the current time step; it may vary.
- `carray` is a two dimensional array containing the force on atoms in one group induced by atoms of another group. Its dimension will necessarily be `nmax` $\times$ 3.

- `compute_peratom()` and `pair_contribution()` are functions which will be described below the corresponding source file.

```
#include <mpi.h>
#include <string.h>
#include "compute_group_group_atom.h"
#include "atom.h"
#include "update.h"
#include "force.h"
#include "pair.h"
#include "neighbor.h"
#include "neigh_request.h"
#include "neigh_list.h"
#include "group.h"
#include "kspace.h"
#include "error.h"
#include <math.h>
#include "comm.h"
#include "domain.h"
#include "math_const.h"
#include "memory.h"

#include <iostream>
using namespace LAMMPS_NS;
using namespace MathConst;

#define SMALL 0.00001

ComputeGroupGroupAtom::ComputeGroupGroupAtom(LAMMPS *lmp,
    int narg, char **arg) :
    ComputeGroupGroup(lmp, narg, arg),
    carray(NULL),
    nmax(0)
{
    if (narg < 4) error->all(FLERR,"Illegal compute
        group/group command");

    peratom_flag      = 1;  // Indicating a peratom compute
    size_peratom_cols = 4;  // # of Columns per atom.
    extarray          = 0;  // 0/1 if global array is all
        intensive/extensive
    scalar_flag       = 0;
    vector_flag       = 0;
}


ComputeGroupGroupAtom::~ComputeGroupGroupAtom()
{
    memory->destroy(carray);
}
```

```
45
46
47  void ComputeGroupGroupAtom::compute_peratom()
48  {
49      // grow array if necessary
50      if (atom->nmax > nmax) {
51
52          memory->destroy(carray);
53          nmax = atom->nmax;
54          memory->create(carray, nmax, size_peratom_cols,
                "group/group/atom:carray");
55          array_atom = carray;
56
57      }
58
59      if (pairflag) pair_contribution();
60      if (kspaceflag) kspace_contribution(); // This doesn't
            happen though. See compute_group_group.cpp
            constructor.
61  }
62
63
64  void ComputeGroupGroupAtom::pair_contribution()
65  {
66      int i,j,ii,jj,inum,jnum,itype,jtype;
67      double xtmp,ytmp,ztmp,delx,dely,delz;
68      double rsq,eng,fpair,factor_coul,factor_lj;
69      int *ilist,*jlist,*numneigh,**firstneigh;
70
71      double **x = atom->x;
72      int *type = atom->type;
73      int *mask = atom->mask;
74      int nlocal = atom->nlocal;
75      double *special_coul = force->special_coul;
76      double *special_lj = force->special_lj;
77      int newton_pair = force->newton_pair;
78      double *columns;
79
80      // invoke half neighbor list (will copy or build if
            necessary)
81
82      neighbor->build_one(list);
83
84      inum = list->inum;
85      ilist = list->ilist;
86      numneigh = list->numneigh;
87      firstneigh = list->firstneigh;
88
89      // loop over neighbors of my atoms
90      // skip if I,J are not in 2 groups
91
```

```
92
93      for (ii = 0; ii < inum; ii++) {
94          i = ilist[ii];
95
96          array_atom[i][0] = 0;
97          array_atom[i][1] = 0;
98          array_atom[i][2] = 0;
99          array_atom[i][3] = 0;
100     }
101
102     for (ii = 0; ii < inum; ii++) {
103         i = ilist[ii];
104
105         // skip if atom I is not in either group
106         if (!(mask[i] & groupbit || mask[i] & jgroupbit))
                continue;
107
108         xtmp = x[i][0];
109         ytmp = x[i][1];
110         ztmp = x[i][2];
111         itype = type[i];
112         jlist = firstneigh[i];
113         jnum = numneigh[i];
114
115         for (jj = 0; jj < jnum; jj++) {
116             j = jlist[jj];
117             factor_lj = special_lj[sbmask(j)];
118             factor_coul = special_coul[sbmask(j)];
119             j &= NEIGHMASK;
120
121             // skip if atom J is not in either group
122             if (!(mask[j] & groupbit || mask[j] &
                    jgroupbit)) continue;
123
124             int ij_flag = 0;
125             int ji_flag = 0;
126             if (mask[i] & groupbit && mask[j] & jgroupbit)
                    ij_flag = 1;
127             if (mask[j] & groupbit && mask[i] & jgroupbit)
                    ji_flag = 1;
128
129             // skip if atoms I,J are only in the same group
130             if (!ij_flag && !ji_flag) continue;
131
132             delx = xtmp - x[j][0];
133             dely = ytmp - x[j][1];
134             delz = ztmp - x[j][2];
135             rsq = delx*delx + dely*dely + delz*delz;
136             jtype = type[j];
137
138             if (rsq < cutsq[itype][jtype]) {
```

```
139                     eng = pair->single(i, j, itype, jtype,
                            rsq, factor_coul, factor_lj, fpair);
140
141                     // energy only computed once so tally full
                            amount
142                     // force tally is jgroup acting on igroup
143
144                     if (newton_pair || j < nlocal) {
145                         array_atom[i][0] += eng;
146                         if (ij_flag) {
147                             array_atom[i][1] += delx*fpair;
148                             array_atom[i][2] += dely*fpair;
149                             array_atom[i][3] += delz*fpair;
150                         }
151                         if (ji_flag) {
152                             array_atom[j][1] -= delx*fpair;
153                             array_atom[j][2] -= dely*fpair;
154                             array_atom[j][3] -= delz*fpair;
155                         }
156
157                         // energy computed twice so tally half
                                amount
158                         // only tally force if I own igroup
                                atom
159                     }
160                     else {
161                         array_atom[i][0] += 0.5*eng;
162                         if (ij_flag) {
163                             array_atom[i][1] += delx*fpair;
164                             array_atom[i][2] += dely*fpair;
165                             array_atom[i][3] += delz*fpair;
166                         }
167                     }
168                 }
169             }
170         }
171 }
```

Listing 6.2: Source file of compute: `compute_group_group_atom.cpp`.

Setting flags

In the constructor we set specific flags that LAMMPS uses to interpret what structure our data should have, and how to store them. We set the `peratom_flag` to be `True`, which indicates that we desire to store some data for each atom. `size_peratom_cols` defines the number of data values to store for each atom. The flag `extarray` is set to `False`, indicating that the per-atom value is an intensive value. Also, we set the `scalar_flag` and `vector_flag` to `False`, since we do not wish to return a vector or scalar value.

Destructor

Following the constructor is the destructor on line 40. Its only task is to free the memory occupied by the array once it is no longer needed.

compute_peratom()

If necessary, this function will resize the array to the number of atoms of concern, `nmax`. It does this using internal functions in the superclass *compute*, which we will not care to describe here. Finally it calls upon the function `pair_contribution()`, which fills the per-atom arrays with the energy and force components. A very important detail is that we only use the pair-contribution of the potential. We only sample the two-body part of the energy and force components. The many-body corrections are ignored. Hence, our measurements will lack that detail. Nevertheless, the two-body term is by far the predominant portion of the potential, and our results should be quantitatively correct.

pair_contribution()

In this function we sample the two-body force acting on each atom of a group from all atoms of another group. Before we dive into the code, some of the variables are described.

> `inum` is the number of atoms (assigned to the processor).
>
> `ilist` is a list of the atom indices of the atoms contained by the processor.
>
> `numneigh` is a list of the number of neighbors each atom has.
>
> `firstneigh` is the actual neighbor list. Thus, a 2-dimensional list containing the index of each neighbor for each atom.
>
> `groupbit` is a 32-bit integer associated with a specific group.
>
> `mask[i]` is a 32-bit integer containing information on which groups atom `i` is part of.

On line 93 we loop over all atoms (assigned to the processor), and assign the atom index to the variable `i`, on the next line. We then explicitly set their energy and force entries to zero. Line 102 - we loop through all atoms again, to fill these entries. First, we check if atom `i` is in the compute group or the second group, by doing a bitwise comparison of the atoms group-int with the group-bits. If it's not, we end the iteration and continue to the next atom. Line 109 - We store the position and type of atom `i`. Line 118 - Loop through all neighbors `j` of atom `i`. Line 125 - Check if `j` is in either of the groups Line 128 - Sort out which group

`i` and `j` are part of. If they are in the same group, we end the current iteration and continue to the next. Line 137 - Compute the square distance between the two atoms. Line 145 - If the interdistance is shorter than the cutoff length, compute the potential energy and interaction force. The force is normalized by the displacement, so that `fpair` $= F_{ij}/r_{ij}$. Line 151 - We sample the energy and force components in the per-atom array. If we do not use newtons third law, we sample the energy contribution twice inside `pair->single`. Hence, we must half the contributions in that case.

6.1.3 Usage

```
1   compute ID group1-ID group/group/atom group2-ID keyword
       value
```

Listing 6.3: Syntax for using custom compute

The compute class shown in listing 6.2 calculates the energy and force interaction between each atom in group1 and all atoms in group2. The groups can be the same group. It is invoked by using the syntax in listing 6.3, where

- ID is the reference to the invoked compute.
- group1-ID and group2-ID are the specified groups we wish to compute the energy and interaction force between. The energy and interaction force is only stored for atoms in group1, and the force is the force acting on group1 from group2.
- keyword is optional and can be either *pair*, *kspace*, *boundary* or *molecule*, with value *yes* or *no*. For a description of these, please look up the documentation for compute group/group[3].

The compute successfully stores the contact force components of all atoms in an array when invoked. We are interested in the spatial distribution of the forces, which describes the force at a given position. It is important to recognize that we operate at the discrete atomic level, and that "force at a given position" is a very vague description. As a solution, we partition the system into a grid, and for each chunk in the grid we store the sum of contact forces. Letting the sum be the collective force within the cell. We divide the system into a 2d-grid in the xy-plane using `compute chunk/atom` as shown in listing 6.4. On the next line we call upon our compute, stating that we wish the force acting on the sphere from the slab. Thus, the force that should have a positive z-component in this case. We could be content with using the result from a single time frame, but since the instantaneous values fluctuate, it's of interest to apply time averaging.

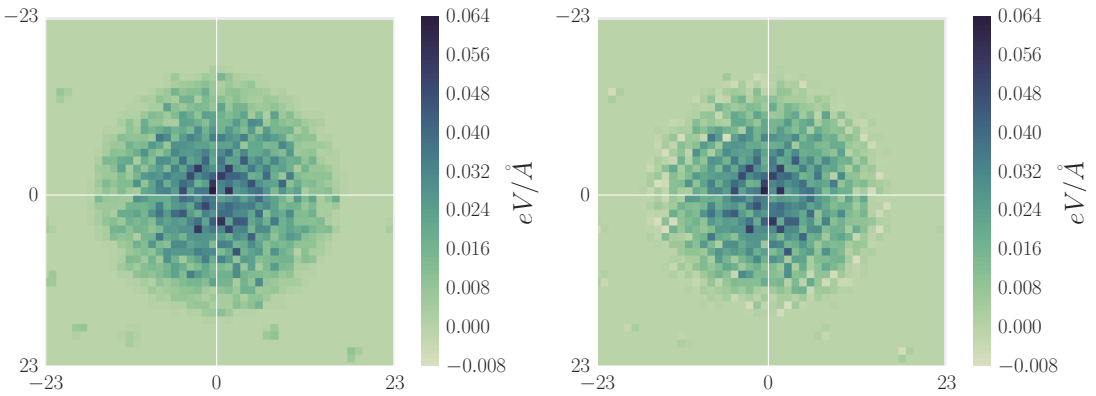
[3] http://lammps.sandia.gov/doc/compute_group_group.html

The `compute ave/chunk` is used for summation and time averaging. It sums the per-atom vectors of atoms in each chunk from specified time steps, and averages this over time. We obtain a matrix of the time-averaged value of the sum of force components on atoms within each grid element of the system. This is stored to a text-file with specified name.

At this point, we can study for instance the distribution of the magnitude of contact force, or contact force in a specified direction as in figure 6.1 . The speckles outside the contact area are atoms/molecules that have broken loose from their original bonds, and drifted until they attached to the other body. In retrospect, it is obvious that this could be avoided by redefining the groups after the equilibration procedure. Though, as it will not alter the quantitative results, this was ignored.

```
1 compute chunkID all chunk/atom bin/2d x 0 7.12 y 0 7.12
2 compute ggaID sphereG group/group/atom slabG pair yes
3 fix fixComputeID sphereG ave/chunk ${Nevery} ${Nrepeat}
    ${Nfreq} chunkID c_ggaID[2] c_ggaID[3] c_ggaID[4] file
    forces.txt
```

Listing 6.4: Example usage of compute group/group/atom with spatial binning and time averaging.



(a) Magnitude of force. **(b)** Force in z-direction.

Figure 6.1: Distribution of force. The color at a square represents the time averaged magnitude (a) and z-component (b) of the contact force of atoms within the square, hence not dependent on direction. This is the result from a simulation where we push a sphere with radius of curvature $R = 200\mathring{A}$, $40\mathring{A}$ into a half-space of area 46×46 unit cells

6.2 Least squares plane regression

The method of least squares aims to find parameters which minimize the sum of the squared residuals, where residuals are the difference between observed values and the approximated value. We will use this method to approximate the slope of the surface of the substrate. This will be done by partitioning the system in a grid and do a plane approximation on each cell of the grid. In other words, we seek the coefficients in the plane equation

$$z = ax + by + c \tag{6.1}$$

that minimizes the sum of the squared residuals

$$S = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} \left(z_i - f(x_i, y_i, \boldsymbol{\beta})\right)^2, \tag{6.2}$$

where $f(x_i, y_i, \boldsymbol{\beta})$ is the right hand side of the plane equation and $\boldsymbol{\beta}$ is the set of coefficients. The minima has the property that the differential with respect to any coefficient is zero.

$$\frac{\partial S}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial r_i^2}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial r_i^2}{\partial r_i} \frac{\partial r_i}{\partial \beta_j} = -2 \sum_{i=1}^{n} r_i \frac{\partial f(x_i, y_i, \boldsymbol{\beta})}{\partial \beta_j} = 0, \ \forall \ \beta_j \in \boldsymbol{\beta} \tag{6.3}$$

When approximating a plane we have three coefficients to account for: a, b and c. This leaves us with the following set of equations:

$$-2 \sum_{i=1}^{n} \left(z_i - ax_i - by_i - c\right) \frac{\partial}{\partial a} \left(ax_i + by_i + c\right) = 0 \tag{6.4}$$

$$-2 \sum_{i=1}^{n} \left(z_i - ax_i - by_i - c\right) \frac{\partial}{\partial b} \left(ax_i + by_i + c\right) = 0 \tag{6.5}$$

$$-2 \sum_{i=1}^{n} \left(z_i - ax_i - by_i - c\right) \frac{\partial}{\partial c} \left(ax_i + by_i + c\right) = 0, \tag{6.6}$$

which corresponds to

$$\sum_{i=1}^{n} z_i x_i = a \sum_{i=1}^{n} x_i^2 + b \sum_{i=1}^{n} x_i y_i + c \sum_{i=1}^{n} x_i \tag{6.7}$$

$$\sum_{i=1}^{n} z_i y_i = a \sum_{i=1}^{n} x_i y_i + b \sum_{i=1}^{n} y_i^2 + c \sum_{i=1}^{n} y_i \tag{6.8}$$

$$\sum_{i=1}^{n} z_i = a \sum_{i=1}^{n} x_i + b \sum_{i=1}^{n} y_i + nc. \tag{6.9}$$

This can be expressed as a matrix equation.

$$\begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ \sum z_i \end{bmatrix} \tag{6.10}$$

where we have omitted the indices from the sums to better readability. Solving this linear system retrieves the optimal coefficients in the sense of the least squares method. The normal vector of the plane will be $\mathbf{n} = [a, b, 1]$. This vector will be used to compute the normal force. To use this method, we stored the position of all the atoms at the substrate surface. These positions were the data points at which we did the regression. Since we know the force acting on a chunk/cell, and now also the normal vector from the approximated plane of the surface atoms, we can compute the normal force as

$$\mathbf{F_N} = |\mathbf{F}| \cos\theta \frac{\mathbf{n}}{|\mathbf{n}|}. \tag{6.11}$$

The cosine of the angle between the two vectors is given as

$$\cos\theta = \frac{\mathbf{F} \cdot \mathbf{n}}{|\mathbf{F}| \cdot |\mathbf{n}|}, \tag{6.12}$$

meaning that the normal force may be expressed as

$$\mathbf{F_N} = \frac{\mathbf{F} \cdot \mathbf{n}}{|\mathbf{n}|} \frac{\mathbf{n}}{|\mathbf{n}|}. \tag{6.13}$$

In general there are two valid normal vectors $\mathbf{n}$ that satisfy the plane equation. However, our procedure assures that we always get the one with a positive z-component. If we compute the contact forces acting on the sphere from the substrate, a positive normal force, will imply a repulsive force, while a negative value an attractive force.

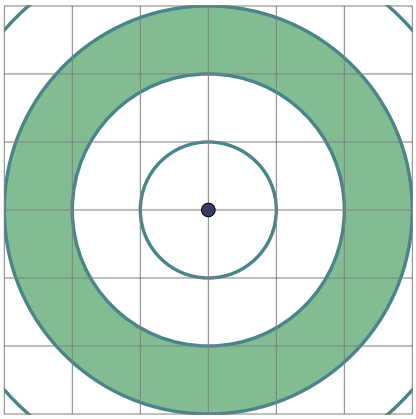


Figure 6.2: Radial binning. The third radial bin is colored, and its value will be the average value of the contribution within the bin.

6.3 Radial distribution

Our system is partitioned into a grid. Each cell in the grid holds the sum of all contact forces on atoms within the cell. We would like to know the radial distribution of the normal force, which should express the averaged value of the force at a given radial distance from the center of the spherical indentation. We achieve this by defining discrete radial bins with fixed bin width, and average contributions within each bin. A coarse method of doing this is to average the values of the cells who's center is within the bin. This is illustrated in figure 6.3. In many applications where the bin width can be large compared to the length of the cells, this method might suffice. However, the current radii of the contact area between the sphere and the slab is only about 15-20 unit cells, and therefor having a large bin width will result in very few data points.

A different, slightly more sophisticated approach is to compute the fraction of the area of each cell that's actually within the bin, and multiply this by the value associated with the cell. This product will be the cells contribution. We can sum all these contributions and average them by dividing by the ratio of the bin area A to the area of a cell a. This means that even cells that do not have their center within the bin might contribute to the resulting bin average. How much, however, will depend on the fraction of the cell's area that is within the bin. This may be regarded as a smoothing of our coarse radial binning method, and will give a more correct result than the coarse binning method already described. An illustration of this binning method is shown in figures 6.4. The figures clearly show that some cells have a larger area within the bin than others, and thus contribute more.

By exploiting the symmetry we are only required to compute the weights for cells in a section of the domain, and not all cells. We have chosen to regard only the upper right corner of the system, meaning $x \geq x_c$ and $y \geq y_c$, where (x_c, y_c)

is the coordinates of the center of the indentation. As the actual computations are easy to follow from the source code, the reader is encouraged to take a look at it if this is of interest. In our computations we have partitioned the space into square cells of length equal the unit cell length (7.12Å). The bin width was set to 1 unit cell length, though this is allowed to vary.

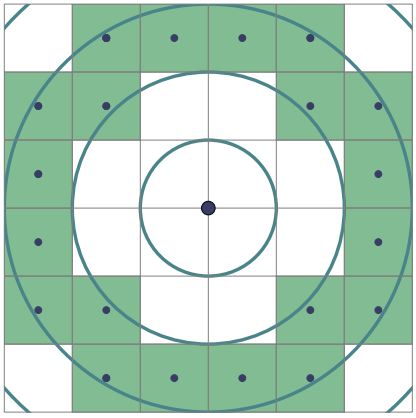


Figure 6.3: Coarse radial binning. The value appointed to the radial bin is the average of the value of the cells who's center is within the bin. The cells with center within the third radial bin are colored, and their centers drawn.

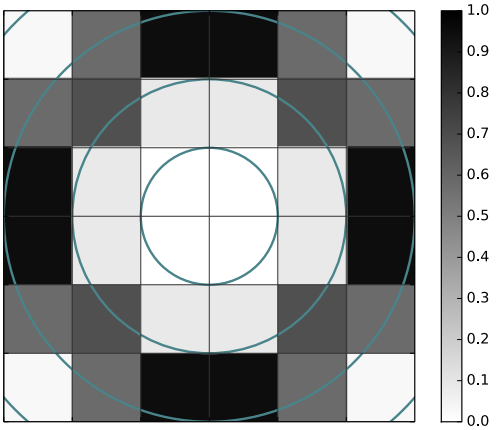


Figure 6.4: Radial binning based on weighted contributions of intersecting cells. The fraction of cell's area that are within the third radial bin is color coded; black being 1 and white being 0.

On the same data set as used in figure 6.1, the computed radial distribution is as shown in figure 6.5. The figure show the spatial distribution in polar coordinates (top row) and the radial average (bottom row) of the magnitude of force, normal force and shear force. The normal force was found by treating the data as

described in section 6.2. What is meant by shear force is in this situation dependent on how we define it. Usually, the shear force is the component of contact force that is parallel to the contact surface (perpendicular to the normal force). However, by strictly using that definition, the computed shear force would have only positive contributions, and no notion of direction. Instead, we define the shear force as the radial component of the shear force. If its general direction is away from the center of indentation, its value will be positive, while if its towards the center it is negative. This way the shear force has a defined direction.

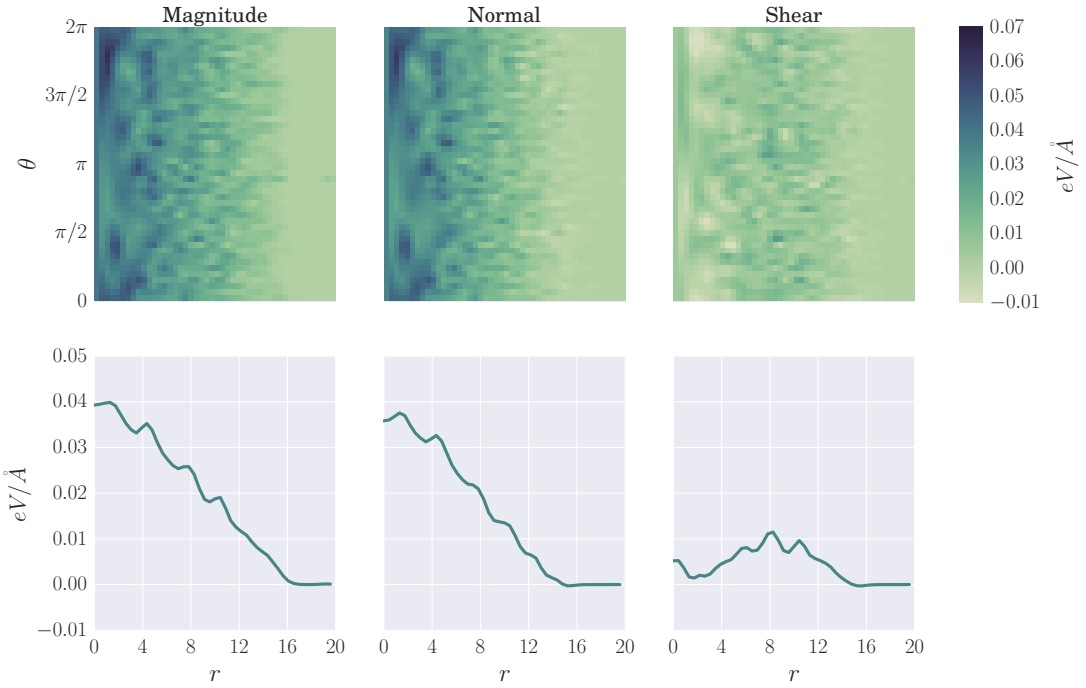


Figure 6.5: Spatial and radial distribution of the magnitude of force, as well as normal and shear force. The spatial distributions shown on the top row are equivalent to the ones in figure 6.1, only in polar coordinates. The magnitude of the force components (y-axis) are the force acting on a square of area $7.12\mathring{A} \times 7.12\mathring{A}$. The value of the radial bins at a given radius r in the figures at the bottom row, are the average value of the column at that r in the top row. The units of r is unit cell length ($7.12\mathring{A}$). This figure show the result from a simulation where we push a sphere with radius of curvature $R = 200\mathring{A}$, $40\mathring{A}$ into a half-space of size $327.52\mathring{A} \times 327.52\mathring{A} \times 157.64\mathring{A}$. This does not imply that the indentation is $40\mathring{A}$ deep, but since contact was established, the top of the sphere has moved $40\mathring{A}$ towards the substrate. The real depth of indentation is less, due to deformations.

6.4 Results - Radial force distribution

6.4.1 Specific description of experiment

Using the methods described in this chapter we were able to monitor the contact force distribution, between the sphere and the substrate, during indentation. The sphere was initially positioned $20\mathring{A}$ above the substrate, leaving no interaction forces between the two bodies. After equilibration the sphere was lowered at a constant rate of $0.5\mathring{A}$/ps, by using the technique of deforming the simulation box, described in section 5.5. At most, the sphere was lowered $60\mathring{A}$, giving a depth of indentation of $h = 40\mathring{A}$. The depth of indentation is here defined as if the sphere was perfectly rigid, unable to be deformed. Thus, a depth of indentation $h = 20\mathring{A}$ is the depth the sphere would indent had it been perfectly rigid. This is not the true depth, however. Due to elasticity the sphere deforms, leaving the indentation to be less than the listed value. A negative depth of indentation describe a separation between the sphere and substrate.

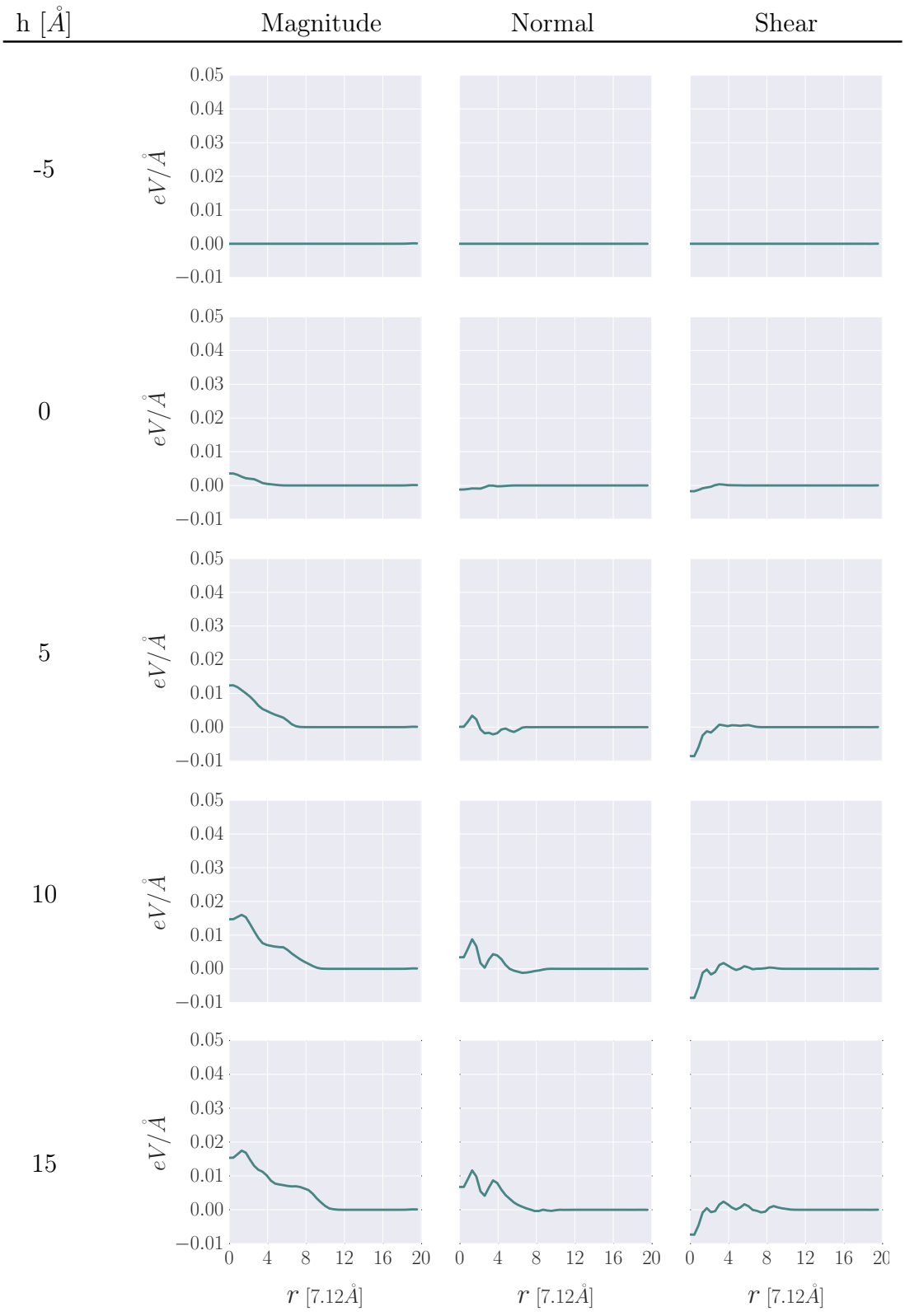
Instead of using momentary measured values for the force distributions, we present the average distribution over a time-frame of 5000 time steps (10 ps). Thus, from we begin sampling until we stop, the (top of the) sphere has moved $5\mathring{A}$ further down towards the substrate. During the 5000 time steps, we sampled every 5th step. Thus, a total of 1000 samples were used for every resulting mean distribution[4].

6.4.2 Description of results

The results from this simulation are shown in figure 6.6. The figure display the radial distribution of magnitude of contact force and the normal- and shear components, for a sequence of increasing depth of indentation h. The stated values of h are the final values of the time-averages[5]. The first two intervals are not shown, since no interacting forces can be observed until $h = 0\mathring{A}$, and so they look identical to the interval at $h = -5\mathring{A}$. The x-axis represents radial distance r from the center of indentation, in units of the unit cell length of β-cristobalite ($7.12\mathring{A}$). The sign of the force components indicate their direction. A positive normal force indicates a repulsive normal component, and a negative normal force an attractive normal component. The shear force is defined as the component of the traditionally defined shear force that is in the radial direction. A positive sign indicates that the shear component is directed away from the center of indentation. Likewise, a negative sign imply that the direction is towards the center. Numeric value of the forces are to be considered the average force acting on atoms of one of the bodies by atoms of the other body, at given radial distance.

[4]This was achieved using Nevery=5, Nrepeat=1000 and Nfreq=5000 in line 3 of listing 6.4.

[5]For example $h = 10\mathring{A}$ means the interval from $h = 5\mathring{A}$ to $h = 10\mathring{A}$.


h [Å]
Magnitude
Normal
Shear
-5
0
5
10
15
eV/Å
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
0
4
8
12
16
20
r [7.12Å]

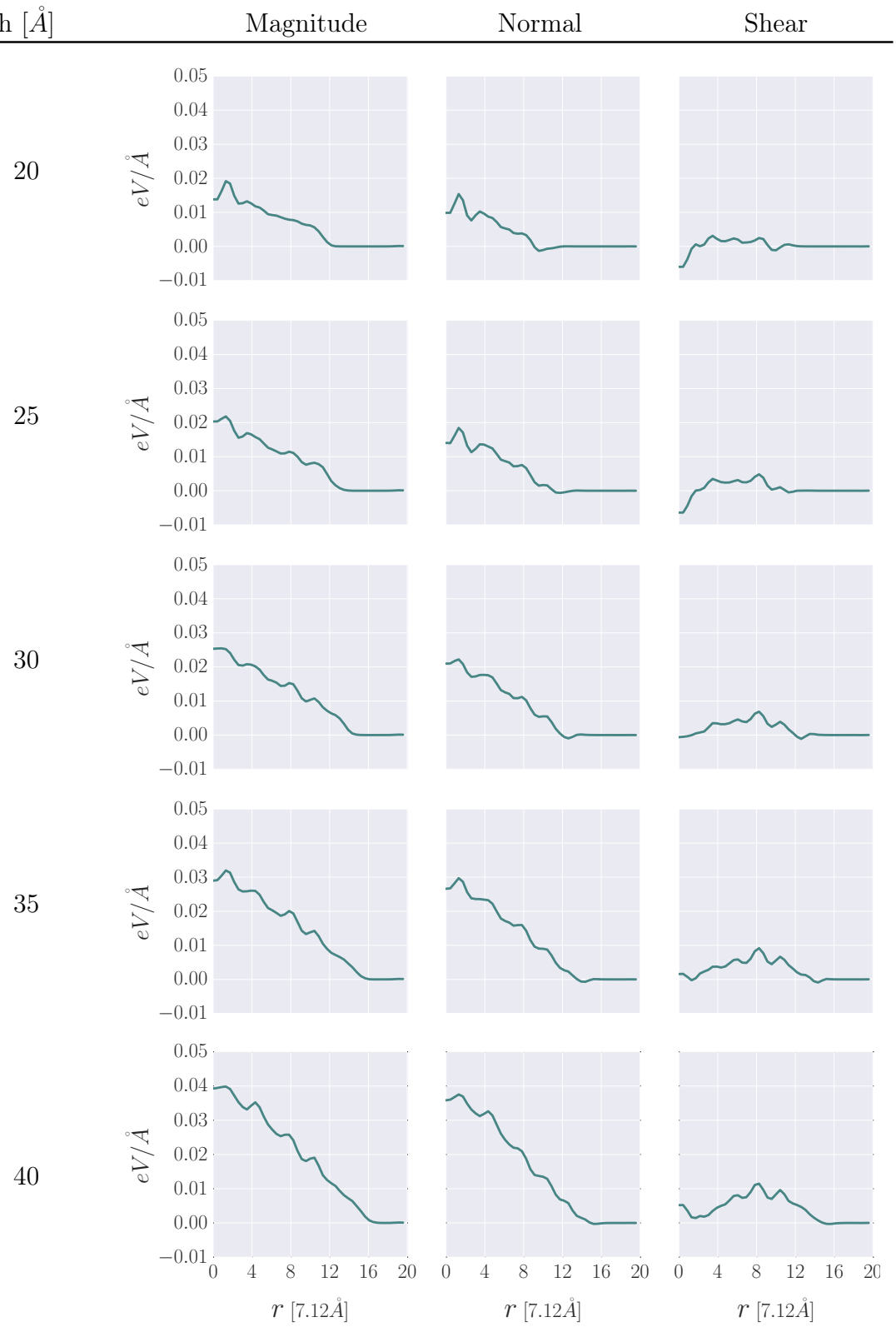

h [Å]
Magnitude
Normal
Shear
20
25
30
35
40
eV/Å
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
0
4
8
12
16
20
r [7.12Å]

Figure 6.6: Radial distribution of contact forces acting on a sphere with radius of curvature $R = 200\mathring{A}$ by a substrate of size $327.52\mathring{A} \times 327.52\mathring{A} \times 156.64\mathring{A}$ at different depths of indentation h. Both bodies are made of β-cristobalite. The radial values are given in units of the unit cell length of β-cristobalite i.e. $7.12\mathring{A}$. The depth of indentation is defined as if the sphere was perfectly rigid. $h = 0\mathring{A}$ being the initial point of contact, $h > 0\mathring{A}$ being an indentation, and $h < 0\mathring{A}$ distance between the surface of the substrate and the lowest point of the sphere. Since the sphere isn't perfectly rigid it deforms and the depths of indentation aren't really as large as the values suggest. A positive normal force imply a repulsive force between the bodies, while a negative normal force imply an attractive force. A positive shear force imply the force to be directed away from the center of indentation, and a negative shear force imply it to be directed toward the center. This way of defining the shear force scales its true value to the component in the radial direction. Hence, with the shear force $\mathbf{F}_S$ we have defined here, we have $\mathbf{F} \neq \mathbf{F}_N + \mathbf{F}_S$.

6.4.3 Observations

We do not detect any interacting forces between the two bodies until we reach the interval $h = -5$ to $h = 0$, shown in the second row of the figure. At this interval we see a small negative value in the normal force, indicating a weak attractive force. Also, the shear force is negative, meaning the shear force is directed towards the center. As the sphere is pushed farther towards/into the substrate the contact forces rise in magnitude, and both the normal- and shear force gain positive contributions.

We notice the normal force to have its maximum value close to the center for all intervals after repulsive forces are established. Theoretically the maximum normal force should be right at the center. However, this is not what we observe. There seems to be a systematic dip in the normal force right at the center. Had the dip not been systematic we could probably blame the discrepancy on bad statistics, considering the number of contribution to the average is very small near the center, and increases as $\propto r$. Since it isn't we cannot use that argument. The exact cause of the behavior is difficult to pinpoint. It may be caused by a wrong definition of the center of indentation in the post-processing; basically that we have placed the center incorrectly. It could be a consequence of atomic structure. Both the sphere and the substrate are made of the same material, with the same crystalline structure. The sphere was simply carved out, without any consideration to its structure or broken bonds. It seems reasonable to expect the sphere to move slightly in order to fit best possibly with the structure of the substrate, hence shifting the center of indentation from the expected position. From figure 6.1b and figure 6.5 it is not clear if the forces are larger on one side of the center, than another. The fact that the shear force near the center seems

to be unreasonably low for most of the intervals may also point to an erroneous placing of the center during post-processing. However, this shifts sign at the later stages. Again, at this radial position there's bad foundation in the statistics, so I won't read too much into this, as it is not completely systematic.

There is another systematic dip in the radial normal distribution. The rightmost non-zero value imply a small attractive force between the bodies. This behavior was expected. The parts of the bodies that we macroscopically would consider not in contact are attracted by each other. We can see from the figure how its radial position moves outwards as the indentation depth increases. From this we could estimate the area of contact to be within the outermost radii where the normal force is positive. This edge of the contact area is also possible to be seen in the radial distribution of shear force. We see a negative value at the dip, because we study the force acting on the sphere from the substrate, leaving the attractive shear force outside the edge to be directed towards the center.

The shear force conforms much better with our expectations. At least at the last intervals it's clear that the maximum value is at a radial distance half of the radii of contact, and is zero both at the center and outside the edge. The discrepancy due to the dip at the center has already been addressed.

6.5 Results - Hysteresis

As a continuation of the simulation giving the resulting distributions shown in figure 6.6, we may rise the sphere as well and study the difference in force interaction between the two cases. This is known as hysteresis. As the sphere has been lowered $60\mathring{A}$ we held it still for 2ps, before rising it at the same speed as it was lowered. The reason for the equilibration is to prevent artifacts due to pressure waves. The flipbook in the lower right corner is a visualisation of the complete simulation. We time-average the momentary normal force distribution in the same way as when it was lowered (described in section 6.4.1). We want to study the total normal force acting between the two bodies, but this term is somewhat vague. Does *total normal force* refer to the sum of the forces' z-component at the contact area, or the sum of local normal force contributions in the contact area? Macroscopically we would choose the sum of z-components, however here we choose the sum of local normal force contributions. As we move the sphere at a predefined rate, we compare the value of the total normal force between the descending and ascending cases at the same depth of indentation. The result is shown in figure 6.7, where the measured normal force while descending and ascending the sphere is shown by the lighter and darker curve, respectively.

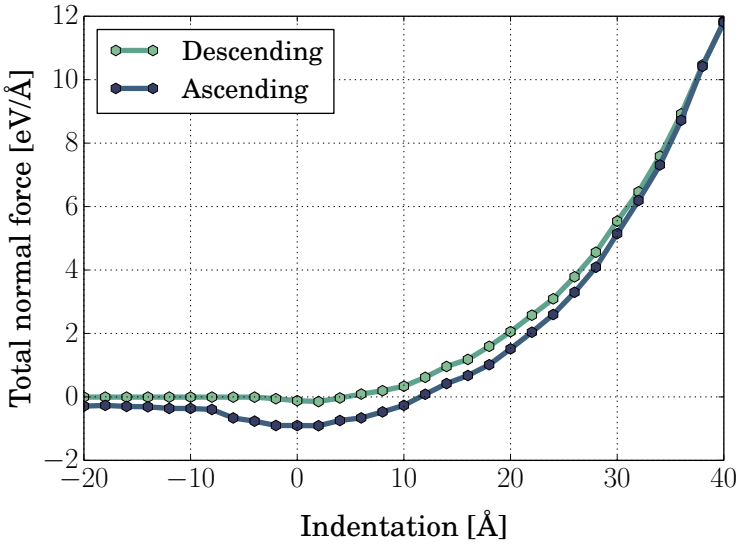


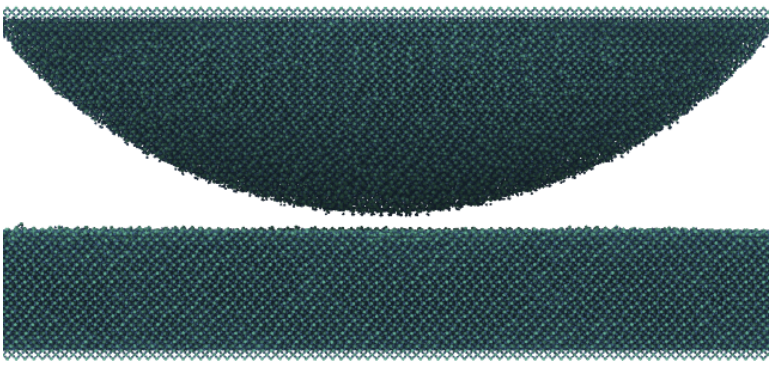
Figure 6.7: Hysteresis plot obtained by simulating a sphere ... pushed down upon/into a flat substrate with constant velocity. The total normal force is computed as the sum of all local normal components of contact force. The indentation is defined as zero where the tip of the sphere is at the same height as the surface of the substrate. The measured normal force while descending and ascending the sphere is shown by the lighter and darker curve, respectively. After the sphere was lowered $60\mathring{A}$, we equilibrated a short time before beginning the ascend. Forces measured are averages of 1000 consecutive time points. Only every second measurement is shown, in order to not clatter the figure.

For identical height of the sphere, the measured normal force was greater during the descend than the ascend. This was an expected result and is due to friction. Atoms from the sphere and the substrate form bonds that counteracting the repulsive forces. Thus, resisting the sphere from rising. When ascending the sphere, these bonds must be broken. The area between the two curves constitutes an energy loss, due to braking of these bonds. The excess energy is dissipated as heat.

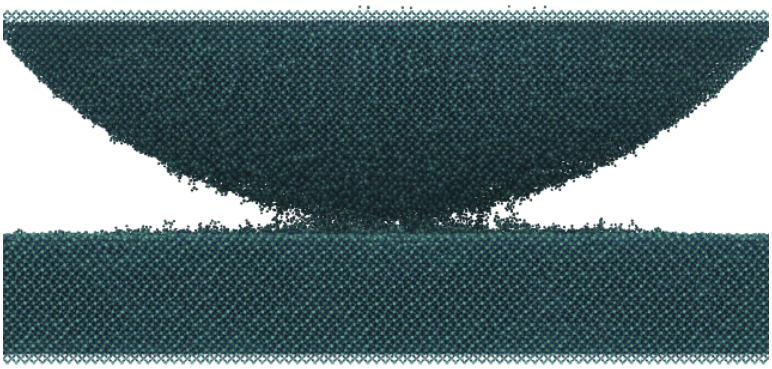
The two curves does not coincide at the height of initial contact (indentation height = 0). In fact, when ascending the two bodies seem to be in contact until the sphere has been elevated $10\mathring{A}$ above the initial height of substrate surface. This is due to adhesion between the two bodies, meaning that atoms form bond with atoms of the other body, and may even loosen from its own. The effects of adhesion are of course only present during the ascend. Figure 6.8a and 6.8b shown the sphere at the same height above the substrate. The latter figure illustrates this effect.

For elevations above $10\mathring{A}$, there is very little contact between the two bodies, and I suspect the measured normal force between them to not be due to these

small contributions, but an effect of the method used. As the sphere ascends from the substrate, some atoms from the substrate may have attached to surface of the sphere. As we are rising the sphere at a constant velocity it makes sense that average force between atoms of the sphere and the attached substrate atoms are attractive. Though the atoms that initially was part of the substrate has become part of the sphere, the groups registered in LAMMPS were defined before that happened. As of now, the computes used do not accept dynamic groups, so this detail is not registered. However, as we can see this issue only result in a minor discrepancy.



(a) Descending



(b) Ascending

Figure 6.8: Snapshot of the system when the tip of the sphere is located $15\mathring{A}$ above the surface of the substrate, while lowering (a) and rising (b) the asperity. These figures illustrate the presence of adhesion. Our view is in the y-direction.

Chapter 7

Computing the coefficient of static friction

As described in section 2.2.1, the coefficient of friction has a changing value depending on the relative motion of the bodies in contact. If they move (slide), the coefficient is usually significantly lower than if they are stationary. In our simulations, we have done the same experiment as described in said chapter, only in a molecular dynamics simulation. We will experience that the velocity[1] dependence of the static friction force F_s impose a challenge in interpreting the results. However, due to a clear linear dependence we are able to extrapolate F_s for velocities approaching 0, which as we shall see, is a minimum limit for the static friction force.

Once the method for finding the coefficient of static friction has been presented, we will study if/how it is dependent on the thickness of the substrate. The results will be posted at the end, but the main discussion is found in section 8.1.

7.1 Maintaining a normal force

From the same simulation done when we pushed the sphere down onto the substrate, we systematically saved a restart file at every 10000 time step. These can then be reloaded separately in order to compute the static force at different values of normal force. Throughout the simulation, we must be certain to remain the normal force acting on the rigid top at the same value as when the restart file was written. In our simulations we have used `fix addforce` with a constant value in the z-direction. When lowering the sphere, the load was linearly increased. Because we change the circumstance from a linear increase to a sudden constant value, we equilibrate a short while before measuring, in order to avert effects of pressure waves.

[1] The velocity at which we pull the spring fixed to the spherical cap.

7.2 Adding a shear force

In order to add a shear force, we fix a spring with stiffness k to the rigid top layer of the sphere cap using the `Fix smd` command

```
fix ID sphereUpperG smd cvel ${k} ${v} tether 500 NULL
    NULL 0.0
```

Listing 7.1: LAMMPS command `Fix smd`, used to add a spring force to a group of atoms. The spring is considered fixed to the center of mass of the group and to a teatherd point, which we may move either with contant velocity of constant force.

which start out with the spring in its resting length, and begin pulling it with a constant velocity v toward the position $x = 500$. We disregard other dimensions, meaning we omit them from the distance computation and force application of the spring. At this point there should be two external forces acting on the rigid top of the sphere cap: the normal force in the z-direction, and the spring force in the x-direction. Each atom i will be assigned an external force

$$\mathbf{F}_i = -k(\mathbf{r}_i - \mathbf{r}_0)\frac{m_i}{M}, \tag{7.1}$$

where k is the spring constant, $\mathbf{r}_i$ and $\mathbf{r}_0$ are the position of atom i and the tethered point, respectively, m_i is the mass of atom i, and M is the total mass of the group of atoms. As we only regard the x-dimension, the position vectors are evaluated as $\mathbf{r}_i = x_i$ and equivalently $\mathbf{r}_0 = x_0$.

7.3 Procedure

In the completed simulations we have used a time frame long enough for the spring force to start decreasing (slip). Typically they were of the order of 50000 time steps (0.1ns). The procedure was equivalent for all time steps, only the velocity at which we pull the spring was increased between simulations. We increase the velocity between simulations because we expect the static force F_s to increase when the normal force increases. In order for the spring to exert a force equal to F_s, we have to extend the spring farther. By increasing the velocity at which we pull the spring we reduce the number of time steps needed, saving time and computational expense. This can be seen in figure 7.3.

`fix smd` computes 7 quantities, that are stored as a vector. As described in the documentation[2], these quantities are: the x-, y-, and z-component of the pulling force, the total force in direction of the pull, the equilibrium distance of the spring, the distance between the two reference points, and finally the accumulated PMF (the sum of pulling forces times displacement), this order. We

[2] http://lammps.sandia.gov/doc/fix_smd.html

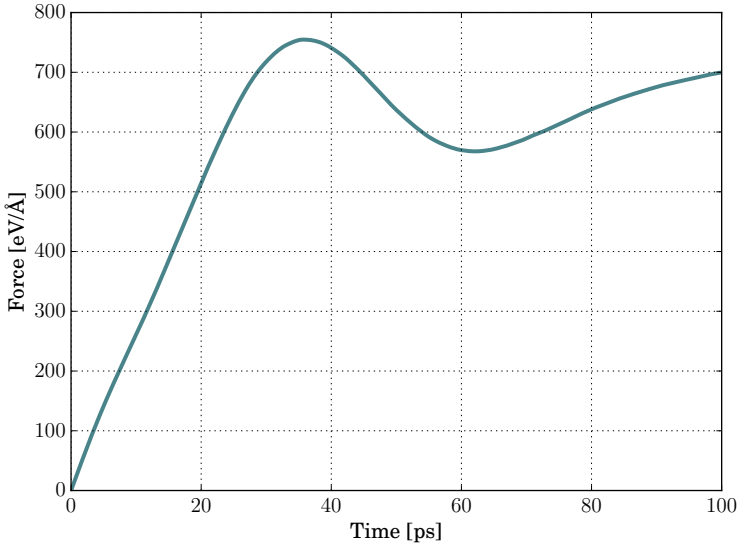


Figure 7.1: Spring force as a function of time, where we pull the spring by a constant velocity $v = 3$ [Å/ps].

are interested in the first value, i.e. the total pulling force component in the x-direction. By setting up a custom format for the *thermo_style*, we consistently store the value of the total force component in the x-direction every 100 time step. This way, we can monitor the force applied by the spring as a function of time.

7.4 Treating the results

The result from a single simulation is shown in figure 7.1. As we see, the spring force increases linearly as expected from Hooke's law. We are really only interested in finding the maximum value of spring force, namely the static friction F_s. Since we know the value of the normal force, we may be enticed to compute the coefficient of static friction as $\mu = F_s/N$ momentarily. However, this would be erroneous. It turns out that the static friction is dependent on the velocity at which we pull the spring. F_s increases with the velocity. Thus, the spring force will engage sliding at a lower value at slow pulling speeds, than at high. This can be deduced from our results as well. Figure 7.2 show the evolution of the spring force from 5 different simulations, all using a different velocity, and demonstrating that the maximum spring force is different in each case. We may rescale the x-axis by multiplying with the velocity to obtain the spring force as a function of the extension of the spring. So, how are we supposed to conclude a coefficient of friction from our simulations? The solution is to find the relation between the coefficient and the velocity. By using the dataset shown in figure

7.2 and 7.3 we may plot $\mu(v)$ by computing $\mu = dF_s/dN$ for every velocity. The result is shown in figure 7.4. The relation is obviously linear, at least for the considered range of velocities. By linear regression the approximated function for the velocity dependent coefficient of friction is

$$\mu(v) = 0.0702v + 0.3992. \tag{7.2}$$

By rescaling the axis of figure 7.3 by this function, we get data collapse as shown in figure 7.5. Note that we normalize the y-axis by dividing with the normal force N. Collapse of the curves indicate that at all these velocities the system is subject to the same physical process. Had the curves been very distinguishable, we could conclude that there is some other effect taking place. These curves overlap quite nicely, so I do not see any cause of concern[3]. The collapse is not of use to us in the pursue of computing the coefficient of static friction. The result that is of real interest is the relation posed in equation (7.2). In macroscopic experiments the spring is loaded very slowly compared to the velocities we simulate. We really seek the value of dF_s/dN when the velocity approaches zero. This is the minimum value of the coefficient. In the case discussed it amounted to $F_s/N \approx 0.4$. This illustrates a common method used in molecular dynamics. For situations we are not able to simulate, like a very low velocity, we try to find a behavior and extrapolate to estimate the value of interest. In this case the behavior was very clearly a linear function, justifying this kind of extrapolation.

So now, for every value of the normal force we study, we need to take a series of simulations using different velocities. It is not intuitive how $\mu(v)$ will change as we perturb N. However, we're only interested in the value of $\mu(v = 0)$. Hence, as long as $\mu(v)$ remains linear, its dependence on N doesn't matter to us. Figure 7.6 show the result of one such series of simulations. For every point $(N, \mu(N))$ there was done 5 simulations using different velocities. The one shown in figure 7.4 is one at which the substrate had a depth XX and the normal force was XX. The constant term in the linear approximations is the stored value of $\mu(N)$. Obviously since $\mu = F_s/N$ we can multiply the y-axis of this figure to obtain the shear force versus normal force, as shown in figure 7.7. Again we see a linear dependence, and may apply linear regression to obtain the relation.

$$F_s(N) = 0.255N + 174.906 \tag{7.3}$$

which is very assuring, since this is in agreement with Amontons' law of friction. The slope of this line indicates the rate at which the shear force increases with increasing normal force. Thus, the slope of this line is the coefficient of static friction we have been looking for. In this example case it was $\mu = 0.255$.

[3]Though this is a subjective opinion.

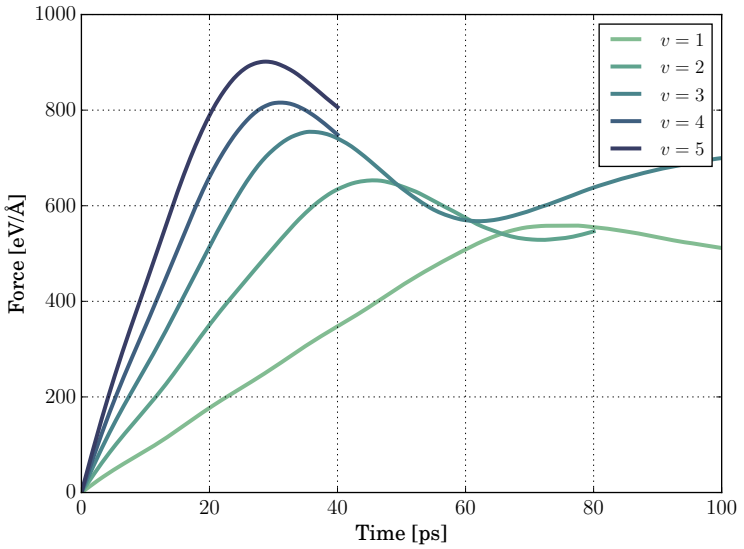


Figure 7.2: Spring force in the lateral direction as a function of time for several values of velocity [$\mathring{A}$/ps], at which we pull the spring.

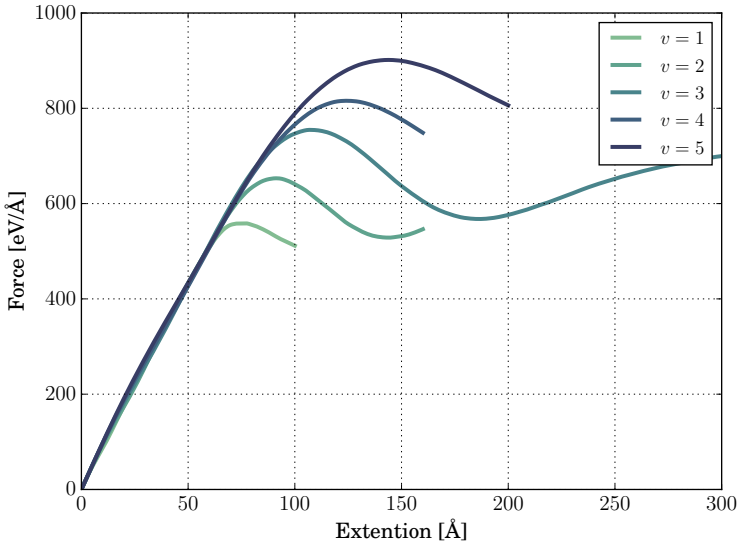


Figure 7.3: Spring force in the lateral direction as a function of the extension of the spring for several values of velocity [$\mathring{A}$/ps], at which we pull the spring.

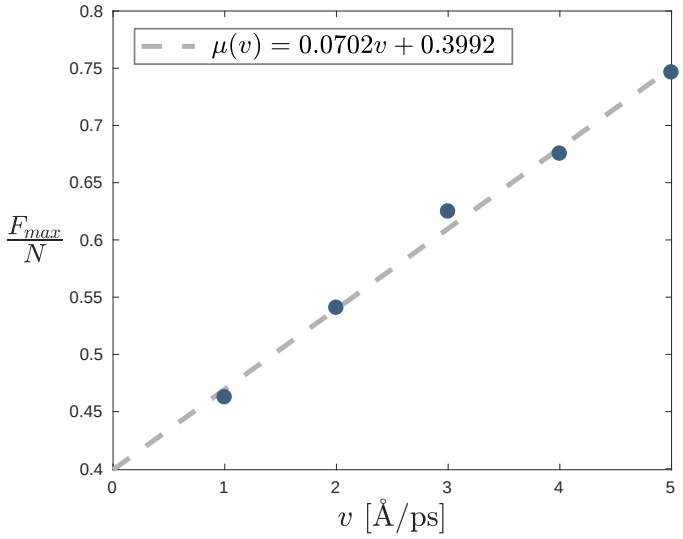


Figure 7.4: Coefficient of friction μ as a function of velocity v. Linear regression is suited to approximate the relation $\mu(v)$.

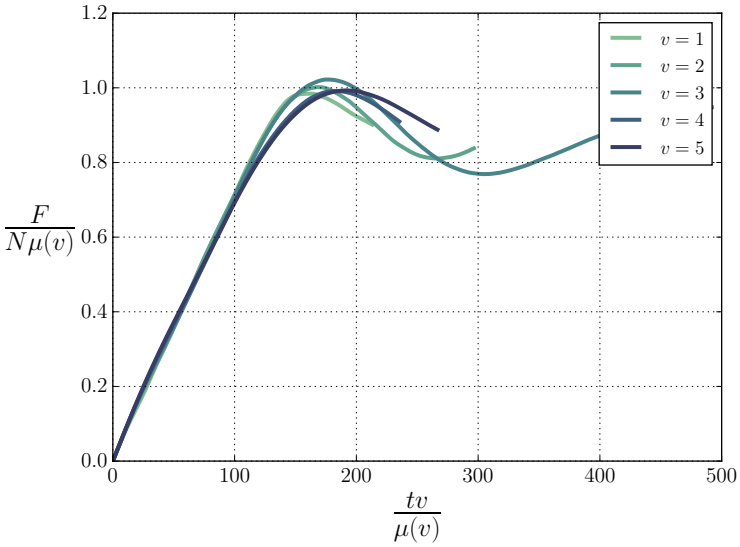


Figure 7.5: Data collapse of series of simulations utilizing different velocities.

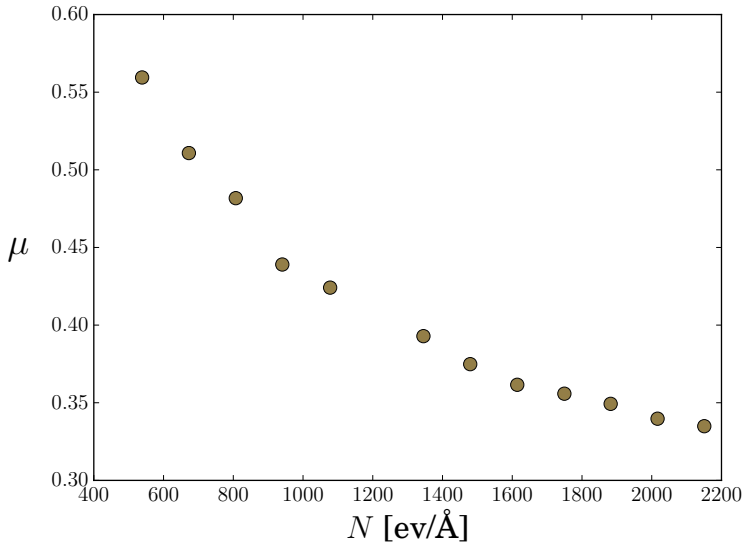


Figure 7.6: Coefficient of friction μ as a function of normal force/load N.

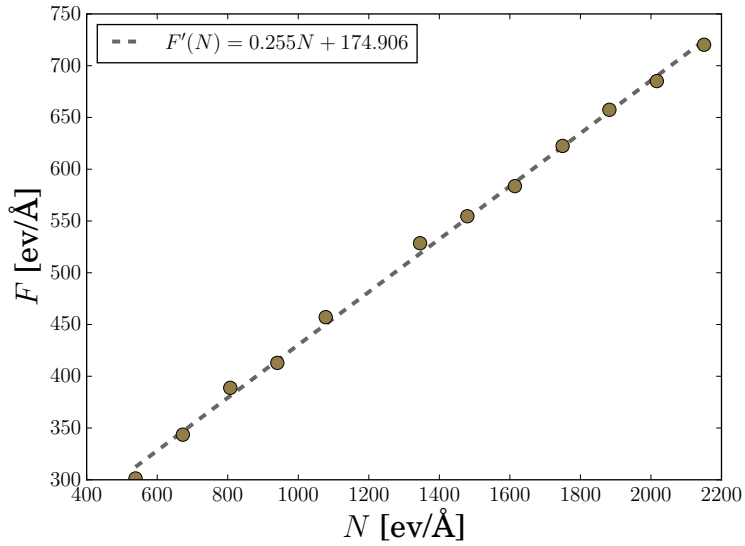


Figure 7.7: Shear force F as a function of normal force/load N.

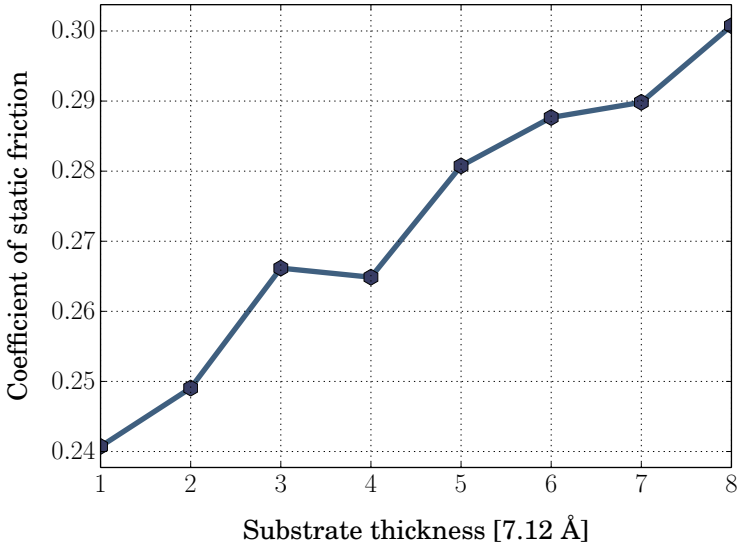


Figure 7.8: Measured coefficient of static friction for various thicknesses of the substrate. Each measurement is taken as the slope of a linear approximation as done in illustrated in figure 7.7, as defined by Amontons' law.

7.5 Results

Our prime goal was to identify a dependence in the coefficient of static friction on the thickness of the substrate. We were therefore obliged to do the same simulation with the simple perturbation of changing the thickness of the substrate. The substrate is formed by carving out all atoms positioned above a specified z-coordinate, and beneath the sphere. It is therefore critical to be aware of where we cut the substrate. The way the crystalline structure of the substrate was oriented in our simulation, a difference in $1\mathring{A}$ in the upper bound of the substrate (where we make the cut), would change the upper most layer of atoms of the substrate to be all oxygen or all silicon. If for one measurement they are all oxygen and another all silicon[4], we will get very different results. To remove the risk of error, we vary the thickness of the substrate by steps with length of the unit cell ($7.12\mathring{A}$). For each thickness of the substrate, we estimated the coefficient of static friction using the method described in the previous section of this chapter. The result is shown in figure 7.8.

The coefficient of static friction increases with the substrate thickness. From the looks of the result, it is tempting to claim a linear relation, at least for this range of substrate thickness.

[4]Of course the surface would contain both types of atoms soon after starting time integration, but the concentration of each type would be different.

Chapter 8

Discussion, Summary and future work

8.1 Discussion

8.1.1 Force distribution

Figure 6.6 shows the radial force distribution in the area of contact, between an elastic sphere and an elastic, flat substrate. The distribution shows the average force acting on the sphere from the substrate within an area a_c, at a radial distance r from the center of indentation. It is therefore equivalent to the pressure distribution, we need only to scale the y-axis with a factor $1/a_c$.

Comparison with Hertz theory

The continuum prediction of the distribution with absence of friction[1] was developed by Hertz, as described in section 2.5.2. In order to compare our distribution to that of Hertz theory, we need to know the radii of contact a (2.23), and the maximum force p_0 (2.25). A challenge is that we do not know E^*, and E^* is not isotropic, so we cannot use equation (2.22) as it stands. Theories can be applied with an effective modulus E^* that depends on orientation and is determined numerically [1], and there exists methods for defining the contact area [15], and thereby a. However, due to limited time and the fact that it is obvious to the author that there are behavioral differences, we do not attempt to compute the theoretical prediction. Instead, we may naively assume the radii of contact a to be the shortest radial distance where the force is zero, $F(r) = 0$. Similarly, we may assume p_0 to be the larges measured force. It's naive because we expect our results to deviate from the theory, and so we are probably wrong in assuming to have the same maximum force p_0 and contact radii a as Hertz. Also, the sim-

[1] Among other assumptions (section 2.5.2).

ulations performed by Robbins resulted in a radii of contact that was far from the prediction. Nevertheless, this enables us to compare the form of the theory's distribution to the one obtained by our simulation, but nothing else. The solid line in figure 8.1 is the same result as the bottom row of figure 6.6. The dashed line represents the distribution predicted by Hertz.

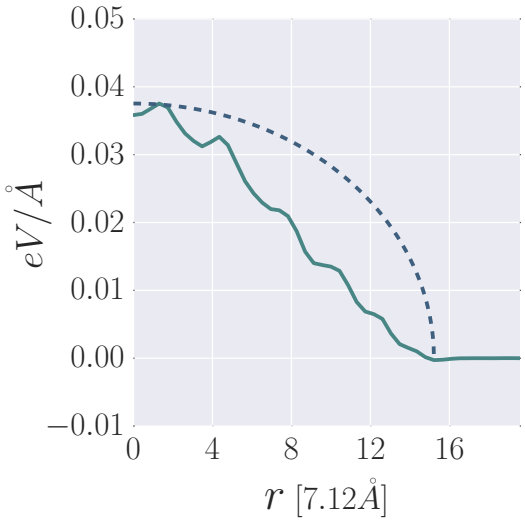


Figure 8.1: Comparison of normal force distribution resulting from simulation (lighter, solid line) and Hertz theory (darker, dashed line).

Not surprisingly, we see big differences in the form of the distributions. The one resulting from the simulation has a much more linear form than what was predicted by Hertz, seemingly in the whole area of contact. The slope of our normal force distribution as a function of radial distance does not fall as $dF/dr = -\infty$, like Hertz theory predicts. Similarly we do not have $dF/dr = 0$ at the center. Reasons for the differences may be numerous. In the simulations, we break almost every assumption made by Hertz. We have friction present, we do not know if we are within the elastic limit, the bodies are finite in size, and the surfaces are discrete, not continuous.

Comparison with Robbins and Luan

Robbins et al. conducted a simulation [2] studying the effects of surface structure on the pressure distribution. They kept the sphere rigid and made both the sphere and substrate out of noble atoms. They used the Lennard-Jones potential truncated at its minimum $r = 2^{1/6}\sigma$ in order to only have repulsive forces between the bodies. This way they did not have friction present. They used several differently constructed surfaces, the most relevant to us is the amorphous, and the stepped tips. As we do not have a rigid sphere, the surface atoms will organize

differently than the bulk atoms. Thus, our surface is the most similar to their amorphous sphere, whereas our bulk is similar to their stepped tip. These two cases showed extremely different behaviors in the resulting pressure distributions. The stepped tip exhibited a qualitatively different behavior than all the others, with a distribution resembling that of a "flat punch". Their results are shown in figure 2.9.

As it is the surface atoms that compose the contact forces, we will compare our results to the distribution obtained by an amorphous tip. This distribution seems to be approximately linear, for most of the range of $r < a$. We have drawn a dashed line in subfigure (d) to highlight this observation. It conformed much better with the theory near the center, though as mentioned - and visible in the figure - there is a weaker statistical foundation for these measurements.

The form of our distribution conform surprisingly well with Robbins' at radial distances not too close to the center. Both were seemingly linear in this region. They deviate only near the center, where our result continues its linear behavior, while Robbins' bends off closer to the theoretical prediction. We find it surprising because of the big differences in our methods: presence of friction, elasticity of the sphere, potential model and material. Also, since there are so many differences in our methods, it is difficult to point out exactly what causes the dissimilarity. To accomplish that, we would have to use much more advanced theoretical models [1], which is impossible to do in a short project like this. However, the fact that both our simulation resulted in a somewhat linear form, may also be evidence that the radial force distribution has a qualitatively different behavior at the nano-scale than the continuum theory predicts.

8.1.2 Coefficient of static friction

The coefficient of friction was found to be dependent on the substrate thickness, having a lower value for thinner substrates. This dependence proved to be in agreement with the experimental results of Julien Scheibert [3]. His experiment included a tip made out of polymer indenting a substrate of the same polymer, laying on top of glass. He concluded that for thin substrates, the coefficient of friction between the two bodies increases linearly. Even though we do not use an isotropic polymer, but a crystalline silica, we see the same tendency; the coefficient increased linearly. This coefficient was only measured for thin substrates ($\sim$1nm-5nm), and we do not expect this linear behavior to remain for thick substrates. Ultimately, it should converge towards a finite value as the substrate thickness increases. These results infer that material properties may behave differently at nano-scale than they do macroscopically.

As satisfying as it is to see simulations reproduce the results of a physical experiment, we need to be critical regarding the methods used in the measurements. For instance, J. Scheibert accounts for the changing area of contact in his

measurements[2], whereas we do not. We pull the spring at ridiculously high velocities, $1\mathring{A}/\text{ps} = 100\text{m/s}$, while in a laboratory, the shear force can be increased for hours before initiating slip. Since the resulting measurements for each velocity fit very well on a line, and it is still only $\sim 10\%$ of the sound propagation speed in the material [25], this should not be of concern.

Numerical values of the coefficient should not be interpreted as the real value. The test of finding the silica's melting point (section 5.3) proved that we missed the real value with $\sim 7.5\%$. However, quantitative behaviors should still be accurate and reliable. This is why we do not bother measuring the slope of the linear dependence, but merely state that there is a linear dependence. Also, we have only completed 8 measurements. If we conducted an additional measurement with a thickness of 9 unit cells, and got a result deviating from this linear form, it could potentially change our conclusion completely. Since we cannot simulate indefinitely, we have to cease when we are confident in the conclusion. In this case 8 simulations was found to be sufficient. Also, each single measurement requires a lot of computational power. To measure the coefficient for a specific thickness, we first had to measure the shear force for 5 different velocities at each value of the load. This was used to find $\mu(v)$ so that we could extrapolate back to $\mu(v = 0)$, for the given load. For each substrate thickness we had 13 different values of N. Having 8 different substrate thicknesses the number (of successful) simulations we have performed to compute the coefficient of static friction was 520. The simulations used on average 80 hours of total CPU time, amounting to 41600 hours of CPU time to produce these results.

8.1.3 Error evaluation

It's always of interest to know how reliable our results are. This is however a difficult question to answer when working with Molecular Dynamics. Our results are highly dependent on our model and methods. Changing the potential model or thermostat may affect the results. Nevertheless, ones we have decided upon these configurations, we can analyze the data errors produces. If we do the same simulation several times, using different initial conditions, we may compute an average and also the standard deviation in the results. We did this for the simulation discussed in chapter 6. We repeatedly did the same simulation, using a different *seed* for the initial velocities. Thus, the velocities were distributed randomly 10 different ways. An averaged radial force distribution was computed, as well as the standard deviation in the results. This is shown in figure averagedDistributionsN10, where the line drawn is the average and the surrounding shaded area represents the standard deviation. The distance between the shaded area's edge to the average is the standard deviation. We see that at large radial distance, the results conform. Close to the center, there are much larger devi-

[2]The area of contact increases with substrate thickness.

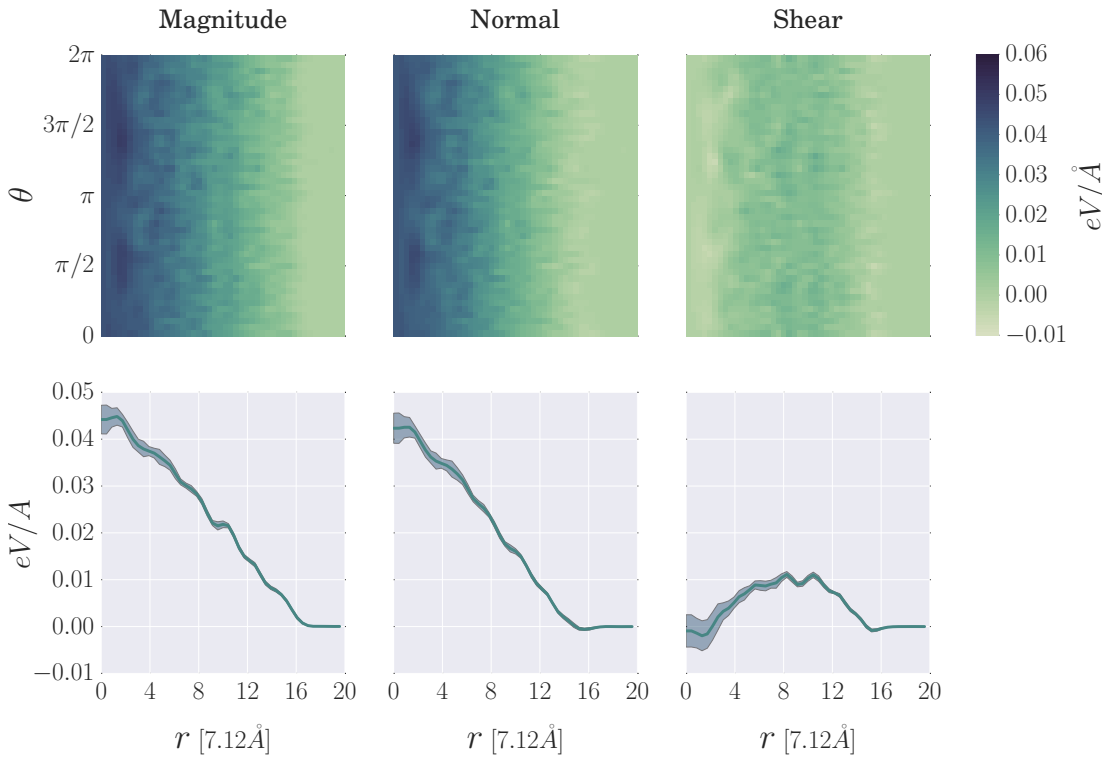


Figure 8.2: Averaged force distribution from 10 simulations following the procedure described in 6. The solid line represents the average radial distribution, while the surrounding shaded area represents the standard deviation in the resulting data.

ations. This is reassuring. The low deviation indicates that all the simulations gave approximately the same result at large radial distance, where the statistical foundation is better. Closer to the center, the radial average is computed from much fewer contributions, making the variance larger. These errors are not significant to our conclusion in this case.

For the simulation concerning the coefficient of static friction, we did not perform such an analysis. The reason is simply the huge amount of computational power needed, and therefore economical expense. This should off course be kept in mind, should the results of this thesis be used for future work.

8.1.4 Molecular Dynamics as a method

Since the problem at hand was partly to compare continuum theory to nano-scale measurements, Molecular Dynamics was a suitable method, because this is the typical length scale we operate on in Molecular Dynamics. We observed that due to the time scales we're able to compute, we may be obliged to simulate processes at high velocities, for instance pulling a spring or lowering the asperity. In some

cases this may not be suitable, but it sufficed for our task. The qualitative results from Molecular Dynamics are difficult to evaluate, since it is highly dependent on many factors: potential model, thermostat, boundary conditions, etc.. One must therefore be careful when deciding upon these aspects of the model. However, if the conditions are reasonable, the quantitative results may provide valuable insight in processes that are difficult to measure otherwise. It is also a good supplement to experiments, as it has been in this work.

8.2 Summary

In this thesis we have developed a Molecular Dynamics model consisting of a spherical cap - resembling an asperity - and a flat substrate. Both bodies were made of β-cristobalite, which is a type of silica. We allowed both bodies to remain elastic in contrast in contrast to the simulation performed by Robbins, and did not restrict frictional forces in contrast to both Hertz theory and Robbins' simulation.

We computed the contact force distribution, and were able to decompose it into normal force and shear force as well. In order to achieve this, we expanded the LAMMPS library with a custom compute, that computet per-atom forces acting between atoms of the two bodies. We also created a method that applies least squares plane regression to approximate the normal vector of the discretized surface. The resulting radial distribution of the normal force was much more linear than expected from the continuum theory of Hertz, and partly similar to the results of Robbins, despite the different restrictions between the models already described.

It was proven that the coefficient of static friction was linearly dependent on the substrate thickness, in the range of thicknesses accounted for ($\sim$1nm-5nm). This was in agreement with the experimental results of Julien Scheibert, and we support his claim; the coefficient of static friction is linearly dependent on substrate thickness for thin substrates.

Our initial goals were met to a satisfactory level, though the comparison to Hertz theory and the results of Robbins necessarily had to be purely quantitative, due to the anisotropic material we used in our model. In the part concerning the coefficient of static friction we were only interested in the qualitative behavior, and this was obtained.

8.3 Prospects for future work

There are many potential methodological errors that may arisen in our model, and if we should continue working with these methods, it might be of interest to refine them a bit. When computing the force distribution for instance, the

center of the indentation is defined to be in the geometric center of the system. It does not account for the possibility that the asperity might move, however slightly, due to atomic structure. A possible solution could be to let the center of indentation be defined by the center of mass of the asperity.

We found behavioral similarities and differences between the distributions obtained by Robbins and our-self. Considering the differences in our models, it would be interesting to investigate the cause of those discrepancies. Are they due to the presence of friction, or the fact that all bodies are elastic? Will we get the same behavior for other materials? Does the distribution depend on ambient conditions? At this point we have developed a model that can be altered as desired, and be used for further studies of friction. Though we have only considered single asperity contact, these results may be of relevance when creating frictional models for larger length scales.

Appendix A

A.1 LAMMPS units

Table A.1: Unit convention of the *metal* unit style in LAMMPS.

Mass	grams/mole
Distance	Angstroms
Time	picoseconds
Energy	eV
Velocity	Angstroms/picosecond
Force	eV/Angstrom
Torque	eV
Temperature	Kelvin
Pressure	bars
Dynamic viscosity	Poise
Charge	multiple of electron charge
Dipole	charge · Angstroms
Electric field	volts / Angstrom
Density	gram / cm^dim

A.2 Building LAMMPS with custom compute

In order to use the custom compute `group/group/atom`, some files in the lammps distribution must be perturbed. They are located in `/path/to/lammps/src/`. All files that have been altered during this thesis are available at my githuh-account in the directory named `modifiedLammpsFiles`. The compute uses only the two-body part of the potential. As a consequence, we must set the `single_enable` flag to be true in `pair_vashishta.cpp`. Secondly, the `private` attributes in `compute_group_group.h` must be changed to `protected`, allowing our custom compute to have access to them. Before we move these files into the source code folder, we must run the make command:

```
make yes-molecule yes-manybody yes-python yes-rigid
```

Now we can move the files into the source code folder, and build LAMMPS for instance as

```
make -j8 mpi
```

The custom compute can then be used as described in section 6.1.

A.3 Example code for computing coefficient of static friction

The following code was applied to compute the coefficient of static friction. It is divided into three parts. The first initializes the system, calling upon the 'system.prepare' and 'system.in', which calls upon 'system.in.init', 'system.data' and 'system.in.setting'. It then equilibrates the system at the desired temperature. The second adds a force to the top of the sphere, which increases in magnitude. While the force increases, and the sphere is pushed into the substrate, we save restart files every 10000 step. Finaly, we sequentially load every restart file (corresponding to distinct normal loads), and attach a spring. For each value of normal load, we did a simulation for five different velocities at which we pulled the spring. Since there we conducted 520 simulations, it was crucial to automate this process. A simple an very specific python script applying regular expression is also shown. This script could then be called upon inside a loop iterating through all the combinations. We then had to change the value of the substrate thickness in 'system.prepare' before repeating.

```
units          metal
boundary       p p p
atom_style     atomic
pair_style     vashishta
```

Listing A.1: system.in.init

```
  pair_coeff  * *  SiO2.vashishta Si O
  pair_modify table 16
  pair_modify tabinner 0.1
```

Listing A.2: system.in.settings

```
variable bottomThickness equal 3
variable slabThickness   equal 7.12*5
variable sphereRadius    equal 200
variable slabUpperLimit  equal
   ${slabThickness}+${bottomThickness}
variable sphereHeight    equal
   ${slabUpperLimit}+${sphereRadius}+5

#-------- Define the region of the sphere --------
region sphereR sphere 163.76 163.76 ${sphereHeight}
   ${sphereRadius}
region sphereUpperR block INF INF INF INF 155 170
region chopTopR     block INF INF INF INF 170 INF
group  sphereG1     region sphereR
group  sphereUpperG region sphereUpperR
group  chopTopG     region chopTopR
group  sphereG      subtract sphereG1 sphereUpperG

# -------- Define the region of the slab ---------
region slabR block INF INF INF INF ${bottomThickness}
   ${slabUpperLimit}
group  slabG region slabR
region slabBottomR block INF INF INF INF 0.0
   ${bottomThickness}
group  slabBottomG region slabBottomR

# Delete all atoms exterior to the sphere and slab
region cutOutR  union 4 sphereR slabR slabBottomR
   sphereUpperR side out
group  cutOutG1 region  cutOutR
group  cutOutG2 union   cutOutG1 chopTopG
delete_atoms    group   cutOutG2

# ----- Create group excluding some groups -------
group excludeB subtract all slabBottomG
group excludeBT subtract all slabBottomG sphereUpperG
```

Listing A.3: system.prepare

```
include "system.in"
velocity all create 1 277387 mom yes
include "system.prepare"

variable N equal  20000
variable T equal  293
variable Fz equal 0.0

neighbor  0.3 bin
neigh_modify  delay 10

thermo    100
timestep  0.002

fix nvtID initialFreezeG nvt temp ${T} ${T} 1.0

restart ${N} restartFiles/SiO2_initial*.restart
dump myDump all atom ${N} dumpFiles/SiO2_initial*.dump

balance 1.1 shift xyz 50 1.0
run ${N}

unfix nvtID
fix   nvtID excludeBT nvt temp ${T} ${T} 1.0

run ${N}
```

Listing A.4: Initialization

```
include "system.in.init"
read_restart restartFiles/SiO2_initial40000.restart
include "system.in.settings"

variable N equal 160001
variable T equal 293
variable Fz equal 0.0231*(40000-step)/160000

neighbor    0.3 bin
neigh_modify delay 10
timestep    0.002

restart 10000 restartFiles/lower/SiO2_lower_*.restart
dump dumpID all atom 20000
   dumpFiles/lower/SiO2_lower_*.dump

fix nvtID excludeB nvt temp ${T} ${T} 1.0

fix rigidTop sphereUpperG rigid single force 1 off off on
   torque 1 on on on
fix addforce sphereUpperG addforce 0 0 v_Fz

compute comTop sphereUpperG com      #Center of mass
compute tempID excludeBT    temp     #Temperature

thermo 100
thermo_style custom step c_tempID v_Fz c_comTop[3]

fix fixBalanceID all balance 10000 1.1 shift xyz 50 1.0
run ${N}
```

Listing A.5: Adding vertical force to sphere

```
variable loadStep equal 130000

include "system.in.init"
read_restart
   restartFiles/lower/SiO2_lower_${loadStep}.restart
include "system.in.settings"

variable loadStep equal 130000

variable   N      equal   40000
variable   T      equal   293
variable   Fz     equal   0.0231*(40000-${loadStep})/160000
variable   K      equal   10
variable   vel    equal   -2

print "k=${K}   vel=${vel}   Fz=${Fz} loadStep=${loadStep}"

neighbor     0.3 bin
neigh_modify delay 10

thermo      100
timestep    0.002

fix nvtID excludeB nvt temp ${T} ${T} 1.0

region slab block INF INF INF INF 3.0 56.0
group sphereIndentTmp region slab
group sphereIndentG subtract sphereIndentTmp slabG

fix rigidTop sphereUpperG rigid single force 1 on off on
   torque 1 on on on
fix addforce sphereUpperG addforce 0 0 ${Fz}
fix smdID sphereUpperG smd cvel ${K} ${vel} tether 500
   NULL NULL 0.0

compute comTop    sphereUpperG  com      #Center of mass
compute comSphere sphereIndentG com
compute tempID    excludeBT     temp     #Temperature

thermo_style custom step c_comSphere[1] c_comTop[1]
   f_smdID[1] f_smdID[6] f_addforce[3]

balance 1.1 shift xyz 50 1.0
run ${N}

print "k=${K}   vel=${vel}   Fz=${Fz} loadStep=${loadStep}"
```

Listing A.6: Loading spring while monitoring spring force

```
import os, sys, re

inJob            = open('jobs/job.sh', 'r')
inInputScript    = open('inputScripts/system.run', 'r')

try:
    step = int(sys.argv[1])
except:
    print 'First argument represents step, and should be
       an int.'
try:
    velocity = float(sys.argv[2])
except:
    print 'Second argument represents velocity, and should
       be a float.'
try:
    N = int(sys.argv[3])
except:
    print 'Third argument represents # time steps, and
       should be an int.'

job              = 'jobs/job%dv%d.sh'%(step,velocity*100)
inputScript      =
   'inputScripts/system.run.%dv%d'%(step,velocity*100)
outJob           = open(job, 'w')
outInputScript   = open(inputScript, 'w')

inputContent     = inInputScript.read()
jobContent       = inJob.read()
inJob.close()
inInputScript.close()

jobContent = re.sub(r'(-in\s)(.*)', r'\1%s'%inputScript,
   jobContent)
jobContent = re.sub(r'job-name=filip',
   r'job-name=filip%d-%d'%(step/1000,velocity), jobContent)

inputContent =
   re.sub(r'(variable\s*vel\s*equal\s*)(-\d*)', r'\1 -
   %.02f'%velocity,  inputContent)
inputContent =
   re.sub(r'(variable\s*loadStep\s*equal\s*[^\d])(\d*)',
   r'\1 %d'%step, inputContent)
inputContent =
   re.sub(r'(variable\s*N\s*equal\s*[^\d])(\d*)', r'\1
   %d'%N, inputContent)

outJob.write(jobContent)
outInputScript.write(inputContent)
outJob.close()
outInputScript.close()
```

```
40
41 sbatch = 'sbatch %s'%job
42 os.system(sbatch)
```

Listing A.7: Script to create input scripts and corresponding slurm job scripts, taking velocity, time step and number of iterations as input arguments.

Bibliography

[1] K. L. Johnson, "Contact Mechanics," 1985.

[2] B. Luan and M. O. Robbins, "Contact of single asperities with varying adhesion: Comparing continuum mechanics to atomistic simulations," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 74, no. 2, pp. 1–17, 2006.

[3] J. Scheibert, "Personal conversation," 2017.

[4] J. R. Bursten, "Nano on reflection," *Nature Nanotechnology*, vol. 11, no. 10, pp. 828–834, 2016.

[5] B. Luan and M. O. Robbins, "The breakdown of continuum models for mechanical contacts," *Nature*, vol. 435, no. 7044, pp. 929–932, 2005.

[6] ""Friction." Merriam-Webster.com. Merriam-Webster, n.d. Web. 25 Feb. 2017.."

[7] I. M. Hutchings, "Leonardo da Vinci's studies of friction," *Wear*, vol. 360-361, pp. 51–66, 2016.

[8] F. P. Bowden and D. Tabor, *The Friction and Lubrication of Solids*. No. v. 1 in Oxford Classic Texts in the Ph, Clarendon Press, 2001.

[9] B. N. J. Persson, *Sliding Friction*. NanoScience and Technology, Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[10] F. P. Landes, *Viscoelastic interfaces driven in disordered media: Applications to friction*. 2015.

[11] S. Timoshenko and J. Goodier, *Theory of elasticity*. McGraw-Hill, third ed., 2001.

[12] C. Cudworth, *Introduction to linear elasticity*, vol. 12. 1986.

[13] W. Voight, "Lehrbuch der Kristallphysik," *Teubner, Leipzig*, vol. 962, p. 1928, 1928.

[14] M. H. Müser, L. Wenning, and M. O. Robbins, "Simple microscopic theory of Amonton's laws for static friction," *Physical Review Letters*, vol. 86, no. 7, pp. 1295–1298, 2001.

[15] S. Cheng and M. O. Robbins, "Defining contact at the atomic scale," *Tribology Letters*, vol. 39, no. 3, pp. 329–348, 2010.

[16] K. Mørken, "Numerical Algorithms and Digital Representation," 2016.

[17] R. T. Cygan, J.-J. Liang, and A. G. Kalinichev, "Molecular Models of Hydroxide, Oxyhydroxide, and Clay Phases and the Development of a General Force Field," *The Journal of Physical Chemistry B*, vol. 108, no. 4, pp. 1255–1266, 2004.

[18] A. C. T. van Duin, S. Dasgupta, F. Lorant, and G. W. A., "ReaxFF: A Reactive Force Field for Hydrocarbonds," *Journal of Physical Chemistry A*, vol. 105, no. 41, pp. 9396–9409, 2001.

[19] P. Vashishta, R. K. Kalia, J. P. Rino, and I. Ebbsjö, "Interaction potential for SiO2: A molecular-dynamics study of structural correlations," vol. 41, no. 17, 1990.

[20] J. L. Lebowitz, J. K. Percus, and L. Verlet, "Ensemble dependence of fluctuations with application to machine computations," *Physical Review*, vol. 153, no. 1, pp. 250–254, 1967.

[21] P. H. Hünenberger, "Thermostat algorithms for molecular dynamics simulations," *Advances in Polymer Science*, vol. 173, pp. 105–147, 2005.

[22] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, 1996.

[23] A. Einstein, "On the Motion of Small Particles Suspended in a Stationary Liquid, as Required by the Molecular Kinetic Theory of Heat," *Annalen der Physik*, vol. 322, pp. 549–560, 1905.

[24] A. Stukowski, "Computational analysis methods in atomistic modeling of crystals," *Jom*, vol. 66, no. 3, pp. 399–407, 2014.

[25] H. J. Melosh, "A hydrocode equation of state for SiO2," *Meteoritics & Planetary Science*, vol. 42, no. 12, pp. 2079–2098, 2007.