

The Department of Physics, Chemistry and Biology

Master's Thesis

**Design and Implementation of a General Molecular  
Dynamics Package**

Lars Erik Rosengren, Peter Steneteg

LITH-IFM-EX--06/1584--SE



**Linköpings universitet**  
**INSTITUTE OF TECHNOLOGY**

The Department of Physics, Chemistry and Biology  
Linköpings universitet  
SE-581 83 Linköping, Sweden



Master's Thesis  
LITH-IFM-EX--06/1584--SE

# Design and Implementation of a General Molecular Dynamics Package


Lars Erik Rosengren, Peter Steneteg

Supervisor: **Valeriu Chirita**  
IFM, Linköpings universitet

Examiner: **Valeriu Chirita**  
IFM, Linköpings universitet

Linköping, 24 March, 2006



	<b>Avdelning, Institution</b> Division, Department  Thin Film Physics The Department of Physics, Chemistry and Biology Linköpings universitet SE-581 83 Linköping, Sweden	<b>Datum</b> Date  2006-03-24
<b>Språk</b> Language  <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English  <input type="checkbox"/> _____	<b>Rapporttyp</b> Report category  <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	<b>ISBN</b> _____ <b>ISRN</b> LITH-IFM-EX--06/1584--SE <b>Serietitel och serienummer ISSN</b> Title of series, numbering _____
<b>URL för elektronisk version</b> <a href="http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-6053">http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-6053</a>		
<b>Titel</b> Title                      Design and Implementation of a General Molecular Dynamics Package           <b>Författare</b> Lars Erik Rosengren, Peter Steneteg Author		
<b>Sammanfattning</b> Abstract  <p>There are many different codes available for making molecular dynamic simulation. Most of these are focused on high performance mainly. We have moved that focus towards modularity, flexibility and user friendliness. Our goal has been to design a software that is easy to use, can handle many different kind of simulations and is easily extendable to meet new requirements.</p> <p>In the report we present you with the theory that is needed to understand the principles of a molecular dynamics simulation. The four different potentials we have used in the software are presented. Further we give a detailed description of the design and the different design choices we have made while constructing the software.</p> <p>We show some examples of how the software can be used and discuss some aspects of the performance of the implementation. Finally we give our thoughts on the future of the software.</p>		
<b>Nyckelord</b> Keywords    MDSinecura, Molecular Dynamics, Computational Physics		



# Abstract

There are many different codes available for making molecular dynamic simulation. Most of these are focused on high performance mainly. We have moved that focus towards modularity, flexibility and user friendliness. Our goal has been to design a software that is easy to use, can handle many different kind of simulations and is easily extendable to meet new requirements.

In the report we present you with the theory that is needed to understand the principles of a molecular dynamics simulation. The four different potentials we have used in the software are presented. Further we give a detailed description of the design and the different design choices we have made while constructing the software.

We show some examples of how the software can be used and discuss some aspects of the performance of the implementation. Finally we give our thoughts on the future of the software.





# Acknowledgements

Thanks to

Valeriu Chirita - Examiner and great support when we needed.

Henrik Wiklund, Oscar Grånäs, Sebastian Andersson - Partners in the CDIO project.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	2
1.2	Goal . . . . .	2
1.3	Thesis Outline . . . . .	2
1.4	Intended Audience . . . . .	3
1.5	Summary . . . . .	3
<b>2</b>	<b>Molecular Dynamics</b>	<b>5</b>
2.1	Theory . . . . .	6
2.2	A simple MD program . . . . .	6
2.3	Potential . . . . .	7
2.3.1	Lennard-Jones . . . . .	7
2.3.2	Embedded-Atom Method (EAM) . . . . .	8
2.3.3	Stillinger-Weber . . . . .	9
2.3.4	Tersoff . . . . .	10
2.4	Integration . . . . .	12
2.4.1	Verlet . . . . .	13
2.4.2	Velocity Verlet . . . . .	14
2.4.3	Higher order algorithms . . . . .	14
2.5	Possibilities . . . . .	14
2.6	Summary . . . . .	15
<b>3</b>	<b>MDSinecura</b>	<b>17</b>
3.1	Design . . . . .	17
3.2	Neighbours . . . . .	18
3.2.1	Internal design . . . . .	19
3.2.2	External design . . . . .	20
3.3	Atom types . . . . .	21
3.4	Integration . . . . .	22
3.4.1	Internal design . . . . .	22
3.4.2	External design . . . . .	23
3.5	Thermostat . . . . .	23
3.5.1	Velocity Scaling . . . . .	23
3.5.2	Andersen Thermostat . . . . .	23

3.6	Boundary conditions . . . . .	23
3.7	Potential . . . . .	24
3.7.1	Framework . . . . .	24
3.8	Tabulation . . . . .	25
3.9	Sample . . . . .	26
3.9.1	Framework . . . . .	26
3.10	Initialization of the geometry . . . . .	27
3.10.1	Ad hoc method for making advanced surfaces . . . . .	27
3.11	Summary . . . . .	30
<b>4</b>	<b>Simulation Example</b>	<b>31</b>
4.1	Initiation . . . . .	31
4.1.1	Simulation parameters . . . . .	31
4.1.2	Potential parameters . . . . .	31
4.1.3	Geometry . . . . .	32
4.2	Simulation . . . . .	32
4.3	Results . . . . .	33
4.4	Summary . . . . .	33
<b>5</b>	<b>Performance</b>	<b>35</b>
5.1	Benchmark . . . . .	35
5.1.1	Time per atom and time step . . . . .	35
5.2	Cutoff length . . . . .	35
5.2.1	Three-body Potentials . . . . .	36
5.3	Tabulation . . . . .	37
5.4	Numerical vs. Analytical Derivatives . . . . .	37
5.5	Data generation . . . . .	37
5.6	Summary . . . . .	37
<b>6</b>	<b>Discussion</b>	<b>39</b>
6.1	Future . . . . .	39
6.1.1	Plugin based potentials . . . . .	39
6.1.2	Plugin based sample calculations . . . . .	40
6.1.3	Modified embedded-atom method . . . . .	40
6.1.4	Samples . . . . .	40
6.2	Conclusion . . . . .	40
<b>A</b>	<b>User Guide</b>	<b>43</b>
A.1	System Requirements . . . . .	43
A.2	Starting the program . . . . .	43
A.2.1	Select a Workspace . . . . .	44
A.2.2	Create a new project . . . . .	45
A.2.3	Create a new simulation . . . . .	45
A.3	Configure the simulation . . . . .	46
A.3.1	Simulation . . . . .	46
A.3.2	Calculations . . . . .	47
A.3.3	Visualization . . . . .	47

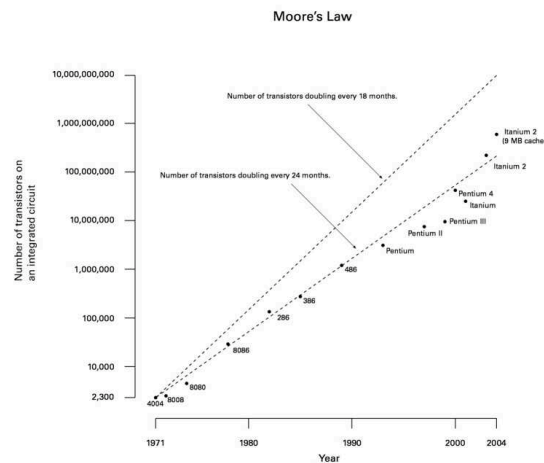
<b>Contents</b>	<b>xi</b>
A.3.4 Potential . . . . .	48
A.4 Setting up a geometry . . . . .	48
A.4.1 Control . . . . .	49
A.4.2 Atom Info . . . . .	49
A.4.3 Groups . . . . .	49
A.4.4 Bulk . . . . .	51
A.4.5 Surface Bulk . . . . .	51
A.4.6 Atom Bombard . . . . .	51
A.4.7 Bombard . . . . .	52
A.4.8 Surface . . . . .	53
A.5 Running the simulation . . . . .	54
A.5.1 Remote Simulation . . . . .	54
A.6 Viewing the Results . . . . .	54
A.6.1 Results . . . . .	55
A.6.2 Plots . . . . .	55
A.6.3 Visualization . . . . .	55
<b>Bibliography</b>	<b>57</b>
<b>List of Figures</b>	<b>58</b>
<b>List of Tables</b>	<b>60</b>
<b>List of Examples</b>	<b>60</b>
<b>List of Listings</b>	<b>60</b>



# Chapter 1

## Introduction

The great increase in computational power, as predicted by Moore seen in figure 1.1, that we have seen in the last century has opened up amazing possibilities for simulations of the universe on all scales. Together with new theories of physics this has totally revolutionized the possibilities to simulate the world around us down to atomic scales. One of the methods to do this is called molecular dynamics and it is the subject for this thesis. The first molecular dynamics simulations tried to simulate the dynamics of liquids but has since then been used in a wide range of different areas.



**Figure 1.1.** Moore's law and the growth of the number of transistors for Intel processors. Figure from [10].

## 1.1 Background

During the CDIO<sup>1</sup> course TFYY78<sup>2</sup> we came across a few molecular dynamics programs. What we saw were programs where the user interaction was all about reading and writing text files. When we were supposed to make our own MD program in the project part of the course we wanted to make it much more user friendly and intuitive than the ones we used. The result was a program with a full fetched graphical user interface with 3d visualization. The simulation part was simple but it worked and we could simulate a few materials such as rare gases and simple metals. When the course was over and we had to find a subject for our master thesis work a continuation of the project felt very natural.

## 1.2 Goal

The goal with this work has been to create a modular and flexible molecular dynamics package with an intuitive user interface and good visualization possibilities. The focus has been to allow for the study of a wide range of systems and to be able to add new kinds of simulations with as little effort as possible.

## 1.3 Thesis Outline

Here is a short description of the outline of the thesis.

**Chapter 2** This chapter covers the theory of molecular dynamics. It starts with the basic mathematics and goes on with describing the implemented potentials and integrators.

**Chapter 3** The next chapter describes our implementation of the theory from chapter 2. It also describes more implementation specific tricks for doing a real simulation software.

**Chapter 4** Here is an example simulation that shows the use of the implementation, how to set it up and how to analyze the results.

**Chapter 5** This chapter contains a discussion about the performance of our program. What we have done and what could be done for faster simulations. Also a discussion on how different settings influences both the simulation speed and the required storage space.

**Chapter 6** The last chapter is a discussion on how we have accomplished our goals and some ideas about future development of this package.

**Appendix A** A user guide to the program with descriptions of most of the implemented features and how to use them.

---

<sup>1</sup><http://cdio.org>

<sup>2</sup>Advanced Physical Applications I, Computational Physics



## 1.4 Intended Audience

The content of this thesis is directed toward an audience with basic knowledge of material physics and some basic orientation in programming. For example a fourth year physics student with some elementary courses in material science and computer programming. A few parts of the thesis might require a higher level of theoretical understanding but the overall concepts should still be clear to the above mentioned audience.

## 1.5 Summary

In this chapter the purpose of our work and the goal with the thesis has been described. In the next chapter the basic theory of molecular dynamics simulations is presented and discussed. Four different potentials that we have used are described and a few different integration algorithms are shown.



## Chapter 2

# Molecular Dynamics

The dynamics of many-body systems consisting of three or more bodies has been shown not to be soluble in general. A solution that has been proven useful in many cases, are numerical simulations instead of theoretical calculations. This branch of science has of course exploded alongside the capabilities of modern computers. Numerical simulations of many-body systems have application in many fields of physics all the way from simulating the galaxies we live in down to the molecules and atom we are made of.

One way of simulating atoms and molecules is called Molecular Dynamics (MD). It has its basis in the deterministic properties of our world, so well put forward by Laplace[7].

Given for one instant an intelligence which could comprehend all the forces by which nature is animated and the respective positions of the beings which compose it, if moreover this intelligence were vast enough to submit these data to analysis, it would embrace in the same formula both the movements of the largest bodies in the universe and those of the lightest atom; to it nothing would be uncertain, and the future as the past would be present to its eyes.

Molecular dynamics is a way to simulate materials on an atomic scale. The input that is necessary to do this is all the atoms' positions and their velocities. MD is a deterministic way to simulate the movement of the atoms. The simulation is divided into a number of time steps, usually in the order of femtoseconds<sup>1</sup>. For each time step all the forces between all the atoms are calculated and then integrated to obtain new positions and velocities. This is iterated to the end of the simulation. During the time steps material properties etc. can be calculated from the positions and velocities.

---

<sup>1</sup>10<sup>-15</sup> seconds

## 2.1 Theory

The molecular dynamics methods is based on ordinary Classical Mechanics[5]. One way to express it is with the Lagrangian equation of motion.

$$\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}_k} \right) - \frac{\partial \mathcal{L}}{\partial q_k} = 0 \quad (2.1)$$

where  $\mathcal{L}$  is the Langrangian

$$\mathcal{L} = \mathcal{K} - \mathcal{V} \quad (2.2)$$

where  $\mathcal{K}$  is the kinetic energy and  $\mathcal{V}$  is the potential energy. For an MD system equation ( 2.1) is given by:

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i \quad (2.3)$$

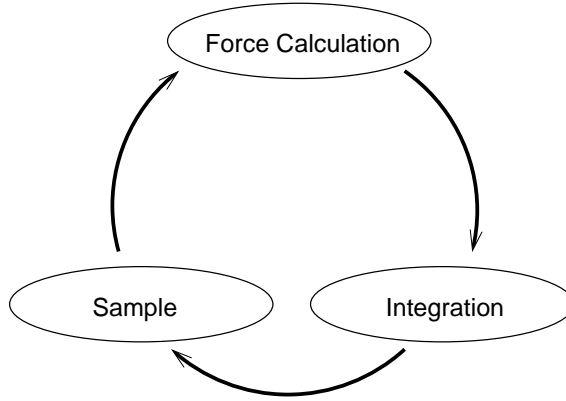
where

$$\mathbf{f}_i = \nabla_{\mathbf{r}_i} \mathcal{L} = -\nabla_{\mathbf{r}_i} \mathcal{V} \quad (2.4)$$

With this, all forces on all particles in the system can be calculated with the knowledge of the potential.

## 2.2 A simple MD program

The most important parts of the MD loop can be seen in figure 2.1. A simulation starts with an initialization where the atoms are placed and preparations are done for the simulation. After that comes the MD loop that contains at least three important parts



**Figure 2.1.** The MD loop with only the most important parts.

- Force calculation
- Integration
- Sample

As the name suggests, the Force calculation calculates the forces which are used to obtain new positions in the Integration. Finally properties are calculated in Sample. This procedure goes on for the number of time steps that is wanted and then the simulation is finished.

## 2.3 Potential

The potential is the heart of the MD simulation because it is the way to determine all forces. It is the correctness of the potential that determines the quality of the results. Using quantum mechanics it is possible to get very accurate descriptions of the interaction but the price is a huge computational expense. The more approximations are done, compared to the real quantum mechanical description, the faster it is to make a simulation, but at the same time some of the accuracy of the result has to be sacrificed. In classical MD the classical potential has the objective to describe the much more advanced quantum mechanics in a quite simple mathematical formula. This is done to hugely increase the size of the systems that can be simulated and also the time that can be simulated. Below the potentials implemented in our program are described. In these descriptions most of the notation from the referenced paper is used.

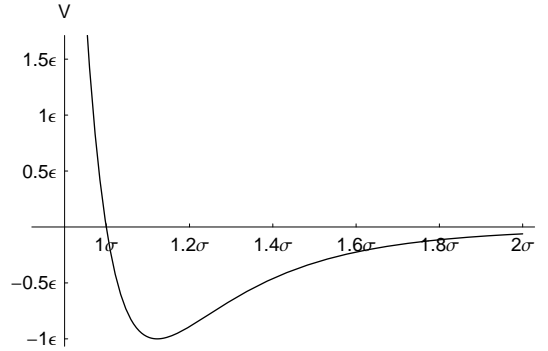
### 2.3.1 Lennard-Jones

Perhaps the most commonly used potential in molecular dynamics is the Lennard-Jones potential (also referred to as the L-J potential or the 12-6 potential).

$$V = \begin{cases} 4\epsilon \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right] & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases} \quad (2.5)$$

In this equation  $\sigma$  and  $\epsilon$  are material specific parameters. A neutral atom is subjected to two distinct forces the repulsive force originating in Pauli exclusion principle and the attractive van der Waals force. This is the interaction the Lennard-Jones potential tries to model with its two parts. The  $1/r^{12}$  dominates at short distances and models the Pauli repulsion. At long distances the  $1/r^6$  part dominates and models the attractive van der Waals force. In figure 2.2 on the next page the characteristic shape of the potential is shown, with its steep repulsive slope for short distances. At about  $1.122\sigma$  the minimum is reached at a depth of  $\epsilon$  and then the attractive, slowly ascending, slope for longer distances converging towards zero as the distance grows.

Van der Waals forces dominate in systems of closed-shell atoms, like rare gases, thus the L-J potential simulate their behaviour quite well. Most of the atoms in



**Figure 2.2.** The Lennard-Jones potential, reaches its minimum around  $1.122\sigma$ .

the periodic system are open-shell atoms and there the interactions are dominated by ionic and covalent bonds. Thus the L-J potential will mimic the behaviour of such systems poorly.

Despite of the weaknesses, the L-J potential is often used as a model system when one wants to study fundamental concepts rather than specific material properties. Another reason for its wide use is that it is very inexpensive to calculate.

Equation 2.5 on the preceding page has been truncated at  $r_c$  to make the computations simpler, instead of evaluating the interaction between all atoms only the atoms within the cutoff  $r_c$ , have to be considered. This truncation leads to a discontinuation in the energy at  $r_c$ , which will give rise to problems with the energy conservation, since every time an atom goes over the cutoff, the energy will jump a bit. To solve this problem there are several different solutions. The most common one is perhaps to simply shift the potential upwards. This will remove the discontinuity in the energy but will still leave one in the potential gradient. Another approach that also solves this is to insert a small new spline function in the interval between  $r_s < r_c$  and  $r_c$ .

### 2.3.2 Embedded-Atom Method (EAM)

In an attempt to better describe the potential in metals, the embedded-atom method was developed. The idea is that the atoms are embedded in an electron gas generated from the neighbouring atoms. It's the so-called embedding function that gives the attractive part of the potential. Following the notation from [6] the potential can be written as

$$\mathcal{V} = \sum_i F(\rho_i) + \frac{1}{2} \sum_{\substack{i,j \\ (i \neq j)}} \phi(r_{ij}) \quad (2.6)$$

The first part of the potential is the attractive embedding function and the second part is the repulsive two-body interaction.

$$F(\rho) = -E_c \left( 1 - \ln \left( (\rho/\rho_e)^{\alpha/\beta} \right) \right) (\rho/\rho_e)^{\alpha/\beta} - 6\phi_e (\rho/\rho_e)^{\gamma/\beta} \quad (2.7)$$

$$\rho_i = \sum_{j \neq i} f(r_{ij}) \quad f(r) = f_e e^{-\beta(r/r_e-1)} \quad \phi(r) = \phi_e e^{-\gamma(r/r_e-1)} \quad (2.8)$$

Here  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\rho_e$ ,  $f_e$ ,  $r_e$  and  $E_c$  are either material or structure specific parameters. If the two atoms are of different elements, the binary interaction  $\phi^{ab}(r)$  is calculated with

$$\phi^{ab}(r) = \frac{1}{2} \left( \frac{f^b(r)}{f^a(r)} \phi^{aa}(r) + \frac{f^a(r)}{f^b(r)} \phi^{bb}(r) \right) \quad (2.9)$$

### 2.3.3 Stillinger-Weber

The Stillinger-Weber potential[8] was the first MD potential developed with the purpose to simulate the covalent materials silicon, germanium and carbon. The primary idea is that a quite general potential can be written as a sum of one-body, two-body, three-body, and so on, contributions:

$$\begin{aligned} \mathcal{V} = & \sum_i v_1(i) + \sum_{\substack{i,j \\ (i < j)}} v_2(i, j) + \sum_{\substack{i,j,k \\ (i < j < k)}} v_3(i, j, k) \\ & + \dots + v_N(1, \dots, N) \end{aligned} \quad (2.10)$$

In SW only the two-body and three-body contributions are considered.

$$\mathcal{V} = \sum_{i,j} v_2(i, j) + \sum_{i,j,k} v_3(i, j, k) \quad (2.11)$$

The two-body interaction is described with

$$v_2(r) = \begin{cases} A(Br^{-p} - 1)e^{(r-a)^{-1}}, & r < a \\ 0, & r \geq a. \end{cases} \quad (2.12)$$

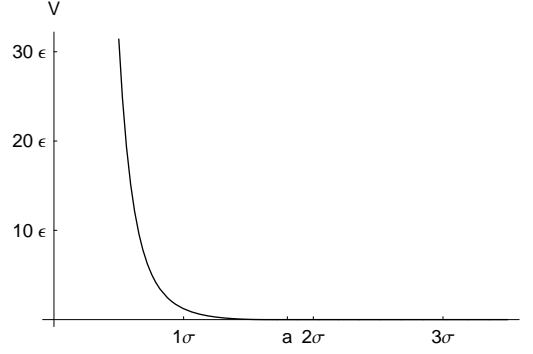
Stillinger-Weber uses some material specific fitting parameters  $A$ ,  $B$ ,  $p$ ,  $q$ ,  $\lambda$  and  $\gamma$  and also the reduced units  $\sigma$  and  $\epsilon$  that are also material specific. The two-body interaction is shown in figure 2.3 on the next page. The three-body part of the potential is the source of all attractive forces. It is built up of three similar functions as can be seen in:

$$v_3(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) = h(r_{ij}, r_{ik}, \theta_{jik}) + h(r_{ji}, r_{jk}, \theta_{ijk}) + h(r_{ki}, r_{kj}, \theta_{ikj}), \quad (2.13)$$

where

$$h(r_{ij}, r_{ik}, \theta_{jik}) = \lambda e^{\gamma(r_{ij}-a)^{-1} + \gamma(r_{ik}-a)^{-1}} \left( \cos \theta_{jik} + \frac{1}{3} \right)^2. \quad (2.14)$$

The purpose of  $\cos \theta_{jik} + \frac{1}{3}$  is to allow for  $120^\circ$  angles, which are the angles found in tetrahedral materials with the so-called diamond structure.



**Figure 2.3.** The two-body part of the Stillinger-Weber potential. The energy on the y-axis is in the reduced unit  $\epsilon$  and the length on the x-axis is in the reduced unit  $\sigma$ .

### 2.3.4 Tersoff

The Tersoff potential[9] is, like Stillinger-Weber, a three-body potential made to simulate covalent materials, such as silicon, germanium and carbon and also alloys of covalent materials for example silicon carbide. The Tersoff potential does not use the same additive approach as SW. Instead the point of the three-body interaction is to scale the two-body potential with the idea that each extra bond an atom has decreases the strength of all bonds.

The Tersoff potential is described in equation (2.15) to (2.22). The potential energy is taken as a sum of all pair interactions.

$$\mathcal{V} = \frac{1}{2} \sum_{i \neq j} V_{ij}, \quad (2.15)$$

where  $V_{ij}$  is a simple two part interaction, portrayed in figure 2.4 on the facing page.

$$V_{ij} = f_c(r_{ij}) (f_r(r_{ij}) + b_{ij} f_a(r_{ij})). \quad (2.16)$$

The function  $f_c(r_{ij})$  seen in figure 2.5 on the next page, is a smooth cutoff function.

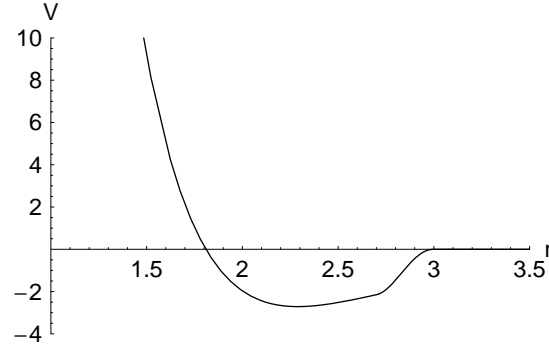
$$f_c(r_{ij}) = \begin{cases} 1, & r_{ij} < R_{ij} \\ \frac{1}{2} + \frac{1}{2} \cos\left(\frac{r_{ij} - R_{ij}}{S_{ij} - R_{ij}}\right), & R_{ij} < r_{ij} < S_{ij} \\ 0, & r_{ij} > S_{ij} \end{cases} \quad (2.17)$$

The other two functions  $f_r$  and  $f_a$  are the repulsive and the attractive part of the potential.

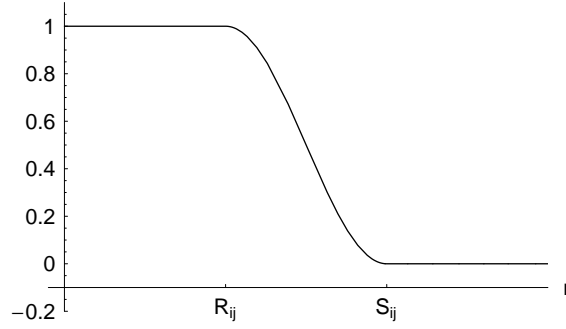
$$f_r(r_{ij}) = A_{ij} e^{-\lambda_{ij} r_{ij}} \quad f_a(r_{ij}) = -B_{ij} e^{-\mu_{ij} r_{ij}} \quad (2.18)$$

The parameter  $b_{ij}$  make the potential a three part potential; it depends on the positions of all the surrounding atoms. It affects the pair potential in such a way





**Figure 2.4.** The basic shape of the Tersoff two part function, with the all parameters from silicon and the parameter  $b_{ij}$  set to unity. The sharp turns around 2.7-3 is due to the cutoff function.



**Figure 2.5.** The Tersoff cutoff function, creates a smooth cut between  $R_{ij}$  and  $S_{ij}$ .

that a large value of  $b_{ij}$  makes the bond stronger and a small value makes it weaker, as shown in figure 2.6 on the following page. In a broad sense, a small value of  $b_{ij}$  corresponds to a crowded environment and vice versa.

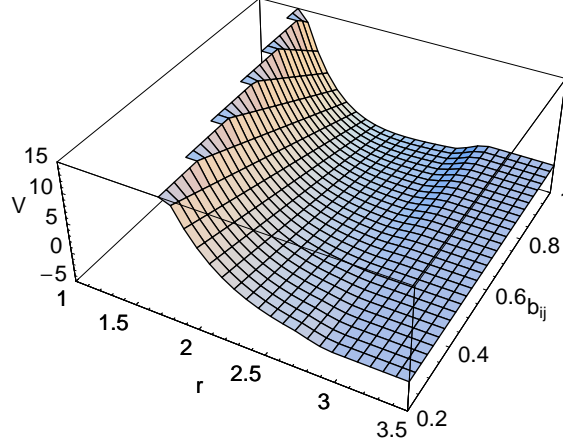
$$b_{ij} = \chi_{ij} (1 + \beta_i^{n_i} \zeta_{ji}^{n_i})^{-1/2n_i} \quad (2.19)$$

The function  $\zeta_{ij}$  works as a kind of bond counter inside the cutoff.

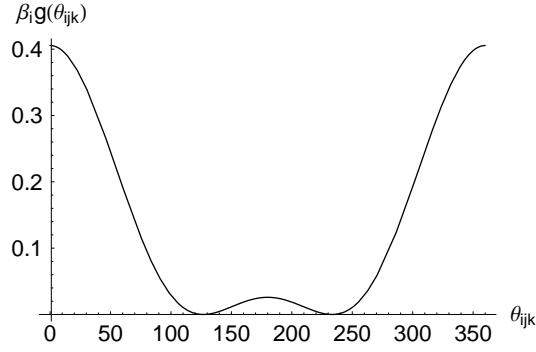
$$\zeta_{ij} = \sum_{k \neq i, j} f_c(r_{ik}) \omega_{ik} g(\theta_{ijk}) \quad (2.20)$$

The counter is angle dependent as can be seen in figure 2.7 on the next page, with the effect that atoms at positions with bond angles of about 125 degrees form stronger bonds.

$$g(\theta_{ijk}) = 1 + \frac{c_i^2}{d_i^2} - \frac{c_i^2}{d_i^2 + (h_i - \cos \theta_{ijk})^2} \quad (2.21)$$



**Figure 2.6.** The Tersoff potential function as a function of both distance  $r$  and the local environment  $b_{ij}$ .



**Figure 2.7.** The angular dependence of the Tersoff potential, bond angles around 125 and 235 degrees are favoured

$$\begin{aligned}
 \lambda_{ij} &= \frac{\lambda_i + \lambda_j}{2} & \mu_{ij} &= \frac{\mu_i + \mu_j}{2} & A_{ij} &= \sqrt{A_i A_j} \\
 B_{ij} &= \sqrt{B_i B_j} & R_{ij} &= \sqrt{R_i R_j} & S_{ij} &= \sqrt{S_i S_j}
 \end{aligned} \tag{2.22}$$

All of  $A$ ,  $B$ ,  $\lambda$ ,  $\mu$ ,  $\beta$ ,  $n$ ,  $c$ ,  $d$ ,  $h$ ,  $R$ ,  $S$  and  $\chi_{ij}$  are material parameters.

## 2.4 Integration

After the forces are calculated, the equations of motions need to be integrated to new positions and velocities. There are several methods to do this with different properties. What properties affect the efficiency of the integrator?

**speed** The execution speed of the integrator.

**memory use** The computer memory needed by the integrator.

**energy conservation** How good the integrator is in conserving the simulation energy.

**trajectory prediction** How close the predicted phase space trajectories are to the "true" trajectories.

The first thought is that the speed of the integrator is very important to get a fast simulation. This however is not really the case because the integrator only takes a very small amount of the simulation time compared to the force calculation. Therefore it is more important for the speed if the integrator allows for the use of larger time steps because then the forces need to be calculated fewer times.

The amount of memory the integrator uses is negligible for small simulation but for simulations of large systems it can be very important.

Energy conservation is very important for a good integrator. It is often closely connected to the time reversibility of the integrator. Energy conservation can be divided into short time and long time conservation and these two properties are usually in conflict, meaning that an integrator with good short time energy conservation often shows energy drifts for longer time spans and the opposite for integrators with good long time conservation.

When first looking at MD the accuracy of the trajectory prediction seems to be one of the most important properties of the integrator. Being able to follow a single atom moving very close to its true trajectory. However this is not the case at all. The chaoticity of most MD systems will make every infinitesimal deviation in the initial conditions lead to an exponentially divergence of the trajectory compared to the true one. This would happen even with an exact integrator because of the finite numerical accuracy of a computer.

Why then is this not a big problem for MD? This is because MD is not used to get exact progress for a single atom but instead give statistical predictions and properties. Of course the trajectories given by the integrator, although not exactly correct, need to be in some sense close to the true trajectories. More discussion of this can be found in [4].

There are many different integrators that fulfil these properties differently. The first thing that comes in mind is to use a truncated Taylor expansion.

$$r(t + \delta t) = r(t) + v(t)\delta t + \frac{f(t)}{2m}\delta t^2 + \dots \quad (2.23)$$

This is not a very good integrator because its not reversible and do not conserve the energy at all.

We need to look at some more advanced types of integrators.

### 2.4.1 Verlet

The Verlet algorithm is one of the simplest algorithms and at the same time one of the best for most cases. It gives good long time accuracy at the cost of a quite

poor short time accuracy which leads to shorter allowed time steps. The memory usage of this integrator is as small as is possible and it is also very fast.

$$r(t + \delta t) \approx 2r(t) - r(t - \delta t) + \frac{f(t)}{m} \delta t^2 \quad (2.24)$$

The velocities are obtained by

$$v(t) = \frac{r(t + \delta t) - r(t - \delta t)}{2\delta t} \quad (2.25)$$

### 2.4.2 Velocity Verlet

Velocity Verlet is a Verlet-like integrator that gives the same trajectories but also gives the velocities in a more straightforward way. It has the advantages of the Verlet algorithm and is probably the most used integrator.

$$r(t + \delta t) = r(t) + v(t)\delta t + \frac{f(t)}{2m} \delta t^2 \quad (2.26)$$

$$v(t + \delta t) = v(t) + \frac{f(t + \delta t) + f(t)}{2m} \delta t \quad (2.27)$$

With this method the velocities need to be calculated after the new forces are calculated from the new positions.

### 2.4.3 Higher order algorithms

There are also more advanced higher order algorithms that can be used in MD. They usually provide higher short time accuracy, which allows for larger time steps, but they also tend to have a larger energy drift than the easier Verlet algorithms. These are often a little more complicated to implement, use more memory and are slower, but for some special simulations the gain of a larger time step is big enough to make it worth the effort. For more on this subject see [4] and [1].

## 2.5 Possibilities

Molecular dynamics can be used for many different purposes. For example it can be used to calculate statistical properties such as

- Temperature
- Pressure
- Potential energy
- Kinetic energy
- Mean square displacements

and more. It can also be used to calculate material properties such as

- Cohesive Energy
- Specific Heat
- Melting temperature
- Bulk modulus

etc.

Some popular systems to investigate with MD are

**Liquids** Liquids was the first target for MD simulations and continue to be interesting. For example viscosity and heat flow can be simulated.

**Defects** Point defects, dislocations and even planar defects like stacking faults and grain boundaries are comonly studied.

**Surfaces** Surface growth, diffusion and much more are simulated.

**Molecules** Study of advanced molecules like proteins, DNA, etc are commonly used in drug design.

## 2.6 Summary

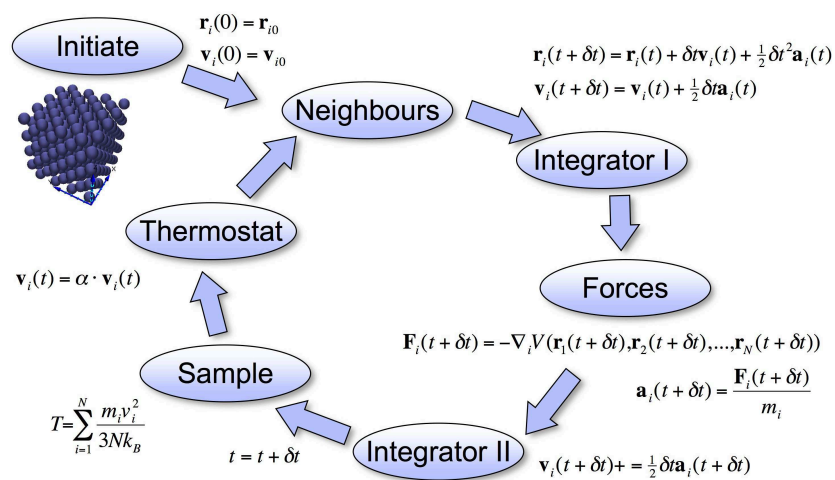
In this chapter the main theoretical parts of a molecular dynamics program has been discussed and explained. It should have given you the needed knowledge to be able to understand the background to the different design choices that we present in the next chapter.



## Chapter 3

# MDSinecura

MDSinecura is the name of our molecular dynamics package. Section 2.2 on page 6 described a very simple MD program and here that simple overview will be filled with all other necessary parts for a working MD code. In figure 3.1 the real MDSinecura loop is shown and in the following sections of this chapter the different parts are explained and analysed.



**Figure 3.1.** A detailed description main loop of the MDSinecura.

### 3.1 Design

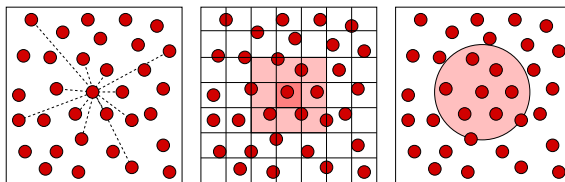
Our design philosophy is to separate the different functionalities of the simulation code. The different parts only need know the outer design of the other parts and

not how they work inside. This is quite easily accomplished with the use of an object-oriented language and we have tried to make the most of it.

## 3.2 Neighbours

The most straightforward way to make a MD simulation is to consider all interactions between all atoms. With a two-body potential this will give a double-loop over all atoms. This means that the simulation time increases with the square of the number of atoms. For 100 atoms there will be 10000 pairs. For a three-body potential the simulation time will increase with the cube of the number of atoms.

To allow for long/large simulations something needs to be done about this problem. To solve this problem we use the cutoff distance of the potentials to only get out the pairs that actually interact with each other. Two common ways to do this are either to use neighbour lists or to use neighbour cells. See figure 3.2 for a view of the different approaches.



**Figure 3.2.** Three different approaches to handling neighbours, from left to right, calculate all pairs, neighbour cells and neighbour lists.

**Neighbour list** The idea of neighbour lists is to go through all pairs and store the ones that have a shorter distance than the cutoff in a list. In the force calculation only the force for pairs in the list needs to be calculated. The list is not updated each time step so the  $O(n^2)$  work of going through all possible pairs only needs to be done every so often. Neighbour lists are very efficient for slow moving system where the pairs are close to constant. In such systems you can update the list with a long interval and because of this it is possible to save time. In other systems, like for example surface bombarding, the list needs to be updated very often because of the large movement of the atoms. In these fast moving system the benefits of the neighbour lists will be small. The problem with neighbour lists in our approach to MD is that it is difficult in a general way to determine how often the lists need to be updated. This does not fit well with our idea to do as general code as possible.

**Neighbour cells** The idea with neighbour cells is to divide the simulation box into cells. Each time step the atoms are sorted into different cells and when calculating the forces only the neighbouring cells need to be looked into for other atoms. All atoms in non-neighbouring cells will be too far away and are not interesting. The sorting only requires a single loop ( $O(n)$ ) over all atoms and for a large system of atoms only a very small part will be looped



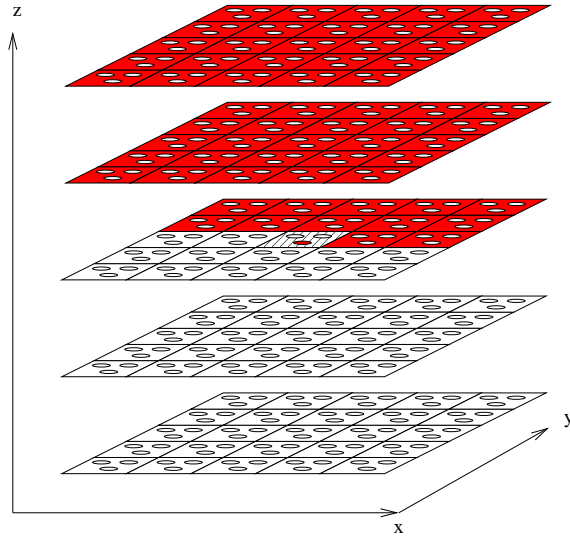
over in the force calculation. The only problem with this is that there will be some pairs with larger distance than cutoff that survives the cells and these needs to be sorted out in force calculation. The neighbour cells approach makes the whole simulation scale as  $O(n)$  to the number of atoms, which can be seen in figure 5.1 on page 36

We have chosen to use the neighbour cells and the reasons are that it easily can be used in almost any kind of simulation and the scaling with the number of atoms is very good.

### 3.2.1 Internal design

The neighbours are handled with a cell division where the cell length is half of the cutoff instead of as usual with cell length equal to cutoff. To get the possible atoms within cutoff you will need to go two cells away. For moderate range interaction these smaller cells will result in an significant increase in speed as it removes some not wanted pairs with a distance larger than the cutoff. Because most potentials only involve nearest neighbour interaction, the extra cells do not make a significant change and we plan to benchmark with the simpler case to see how much different it makes.

Figure 3.3 shows how the cells are used when using a two-body interaction potential. Because the potential only depends on the distance between the atoms and the forces on the two atoms are exactly the same but with the opposite direction we only need to get a pair once and because of that we only need to, for each neighbour cell, go through half of the interacting cells.



**Figure 3.3.** The implementation of neighbour cells using half cutoff sized cells. The colouring shows which cells are checked when using a two-body interaction potential.

There are actually three different internal designs of the neighbour cells depending on how the potential is formulated.

**two-body** In the two-body case the force is calculated per pair and for that each pair only needs to be returned once. Because of this only half the neighbouring cells are checked for each atom as described above.

**ordinary three-body** Almost the same as for the two-body but we add one more atom chosen from the remaining atoms so that each triplet is returned only once. The third atom is found so that one of the distances from the two other atoms are smaller than cutoff.

**special three-body** In the last case all pairs are returned first and then the third atom is returned to create all possible triplets for that pair. This will result in that each triplet is returned three times, one for each pair in the triplet. This version needs to be used to work with the formulation of the Tersoff potential.

### 3.2.2 External design

The neighbours handling is designed so that the implementation of the force calculation does not need to know how to decide the next atom. The only thing needed is to decide which of the different Neighbour cells implementations that is to be used by the force calculation. There are six methods that needs to be called to loop over the necessary atoms.

**hasPrimaryAtom()** Asks if there are more primary atoms.

**hasSecondaryAtom()** Asks if there are more secondary atoms.

**hasTertiaryAtom()** Asks if there are more tertiary atoms.

**getPrimaryAtom()** Gets the next primary atom from the neighbours handler.

**getSecondaryAtom()** Gets the next secondary atom from the neighbours handler.

**getTertiaryAtom()** Gets the next tertiary atom from the neighbours handler.

The methods `hasTertiaryAtom()` and `getTertiaryAtom()` are only to be used when using a three-body potential.

To finish off the neighbours section is an example that shows how to use the neighbours when implementing the force calculation.

---

#### Listing 3.1: Using neighbours for two body potential

---

An example of how to use neighbour cells when using an easy two-body potential, for example Lennard-Jones.

---

```
1 while (neighbours.hasPrimaryAtom()) {  
2     i = neighbours.getPrimaryAtom();  
3     while (neighbours.hasSecondaryAtom()) {  
4         j = neighbours.getSecondaryAtom();  
5         r = getDistance(i, j);  
6         if (r < cutoffDistance) {  
7             f = getForce(r);  
8             forces[i] += f;  
9             forces[j] -= f;  
10        }  
11    }  
12 }
```

---

### 3.3 Atom types

In some simulations it might be needed that the simulation handles different atoms differently. To show this we will look at the example of a surface simulation. A picture of the setup can be seen in figure 3.4 on the next page. In the figure there are 5 different groups of atoms that are coloured differently. The groups from the bottom and up:

**Bottom layer** To simulate a large bulk underneath the surface and also to give stability to the surface it is very common to lock the positions of the atoms in the bottom layer. The atoms interact as usual but they are not moved.

**Thermostat layers** When doing a simulation with bombarding atoms (e.g. surface growth) the incoming atoms will add a lot of energy to the surface. To get rid of the energy we can use a few layers to take away extra energy. These layers simulate a larger bulk beneath where the energy can disappear.

**Surface layers** These are ordinary layers without any special functions. However it is possible to put a thermostat on the whole system (see section 3.5 on page 23) and then these atoms would also be affected.

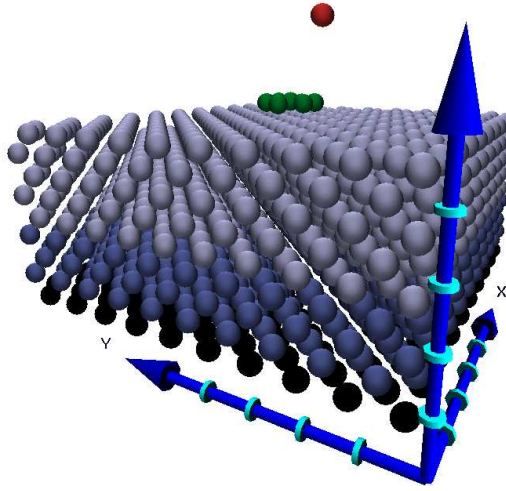
**Surface structure** The surface structure should not be affected in any way by any thermostat.

**Bombarding atom** The bombarding atom must not be affected by thermostats or it will not follow the path specified in the configuration.

Some atoms need to be moved in a special way and others need to be disconnected from the thermostat. To solve all this we have created a few "Atom types" and all of these can have different integrators and thermostats. The implemented atom types can be seen in table 3.1 on the next page.

Type	Integrator	Thermostat
Common	Normal	Normal
Locked	Disabled	Disabled
Scaled Temperature	Normal	Scaling thermostat
Bombarding	Normal	Disabled
Surface	Normal	Disabled

**Table 3.1.** The different atom types defined in the software. Normal means that the type uses the thermostat and/or integrator chosen in the configuration.



**Figure 3.4.** This is how a surface simulation could look like. Colour coding shows the different types of atoms.

## 3.4 Integration

As described in the atom types section above the integrator may have special functions for different atom types. In this section the normal integrator is described. For theory of integration see section 2.4 on page 12.

### 3.4.1 Internal design

The ordinary integration for a normal atom is the commonly used Velocity Verlet integrator described in the following equations.

$$r(t + \delta t) = r(t) + \delta t v(t) + \frac{1}{2} \delta t^2 a(t) \quad (3.1)$$

$$v(t + \delta t) = v(t) + \frac{1}{2} \delta t (a(t) + a(t + \delta t)) \quad (3.2)$$

### 3.4.2 External design

The integrator only has two methods that needs to be called for each atom. What happens in the integrator is not interesting when calling these methods. Which integrator is called is decided when setting up the simulation.

**integratePosition(int atom)** Integrates the new position for atom. Called before the force calculation.

**integrateVelocity(int atom)** Integrates the new velocity for atom. Called after the force calculation.

## 3.5 Thermostat

In many simulations there is a need for some kind of temperature management. In a bulk simulation one might want to do an NVT ensemble simulation. While simulating bombarding of a surface one often has to arrange for energy removal from the bulk emulating heat transfer deeper into the bulk. This is often done by some kind of temperature scaling in the bottom layers of the simulation box. For this a framework has been developed consisting of a set of different types of scaling and heat bath functions.

### 3.5.1 Velocity Scaling

Velocity Scaling is the easiest way to give the simulation a constant temperature. This is done by simply taking the average of the velocities and then scaling them to the wanted temperature. This method does not correspond to any known ensemble but can be very useful in some situations.

### 3.5.2 Andersen Thermostat

The Andersen thermostat is a stochastic constant temperature thermostat invented by Andersen[2]. The idea is that the system is coupled with a heat bath with the desired temperature and then each atom has a probability to interact with the heat bath and get a new random velocity taken from a Maxwell-Boltzmann distribution corresponding to the temperature of the heat bath. By doing this it can be shown that it will generate a canonical distribution.

## 3.6 Boundary conditions

A MD simulation often requires some kind of boundary conditions . There can be several different types. In the case of solid-state simulations the most common one is periodic boundary conditions (pbc), mainly used for simulating a homogeneous bulk. The periodic boundary condition can be applied in several different ways. For a bulk simulation the pbc will be applied in all three directions but for a surface simulation the pbc might only be applied in the two directions in the plane.

The boundary conditions are mainly used when trying to find the distance between atoms and when moving atoms and they are needed from almost everywhere in the simulation. The actual code required to calculate the periodic boundaries is very simple and the important part is to make an implementation that is general and easy to modify.

Our implementation uses a pbc class hierarchy that can easily be configured to fit the demands of the simulation (surfaces/bulk). All distance calculation requires the use of the boundary conditions so it is very important to use it to calculate all the distances through out the simulation code.

## 3.7 Potential

The most important part of every molecular dynamics simulation software is its potential code. From the potential all the forces are calculated and it is the accuracy of the forces that determine the accuracy of the whole simulation in the end. For classical molecular dynamics there are not any general potentials that apply to all systems, like there are in the ab initio case. Instead, more specific potentials have to be used for each system to be simulated. In many cases a potential must be developed solely for the system of interest, often based on ab initio calculations. This means that many different potential codes have to be supported in a useful simulation package. For this a general and easy way to add and modify potentials in the software is needed. It is also important to keep all non-potential specific code out of the potential implementation to not be required to implement it again and again.

Another aspect of the potential code is that it is here that most of the computations are carried out. This means that if good performance is wanted the potential code is the place to do optimizations.

In a normal simulation the evaluation of the forces is the most expensive computational task. In a system with  $n$  atoms one has to in principle calculate  $O(n^2)$  forces. In many cases this can be simplified by using different approximations, limiting the range of the forces for example. Using this approximation it may be sufficient to calculate only an order of  $O(n)$  forces.

Within this lies a conflict; it is very hard to build a framework for the potential code that is both optimized for speed and general enough to be useful with different kinds of potentials. Somewhere along the line between speed and flexibility a choice has to be made. In our implementation of the line is drawn closer to the flexibility than in many other codes.

### 3.7.1 Framework

In our framework the potential code has two responsibilities. It has to calculate the potential energy and assign accelerations to all the atoms in the simulation. To do this the potential code has access to a few tools. An overview of the framework can be seen in figure 3.5 on the next page and the different parts are:

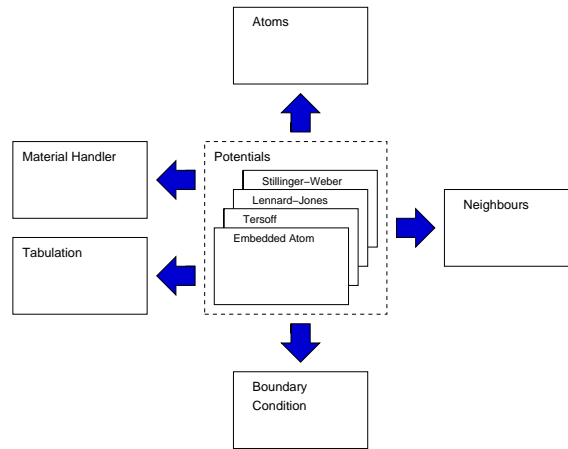
**Atoms** The main structure that hold all information about the atoms.

**Neighbours** The neighbour cells handler described in section 3.2.2 on page 20.

**Boundary handler** A class that manages all information about boundary conditions in the simulation. Described in section 3.6 on page 23.

**Material handler** A class that contains all material properties needed for the simulation.

**Tabulation** A framework that handles tabulation of various mathematical function described in section 3.8.



**Figure 3.5.** Overview of the potential framework.

## 3.8 Tabulation

Since a lot of the force calculations demand evaluation of several complicated mathematical expressions there is a motivation for tabulating some of these. In our design we built a framework for handling the tabulation of mathematical functions. The framework incorporates convenient methods for generation of the table and methods for extraction of the spline interpolated function values and derivatives.

**createTabulation(tabulationStep, cutoff, tabulationFunction)** Generates a new table for the supplied function in the interval  $[0, \text{cutoff}]$  with the given step.

**evaluate(rij)** evaluates the function at the point  $\text{rij}$  using cubic spline interpolation.

**evaluateDerivative(rij)** evaluates the derivative of the function at point  $\text{rij}$  using cubic spline interpolation.

**Listing 3.2: Tabulation**

An example of how the tabulation framework can be used for the Lennard-Jones potential

```

1  TabulationTable potentialTable =
2  TabulationFactory.createTabulation(tabulationStep ,
      cutoff , new TabulationFunction() {
3      public double evaluate(double r) {
4          return 4*epsilon*
5          ( Math.pow(sigma/r,12) - Math.pow(sigma/r,6) );
6      }
7  });
8
9  double rij = distance(atom_i, atom_j);
10
11 double potential = potentialTable.evaluate(rij);
12 double force = -potentialTable.evaluateDerivative(rij);

```

## 3.9 Sample

Many different physical quantities can be calculated from the atomic position and velocities over time. To make these calculations as simple and flexible as possible we have developed a small framework to do these calculations.

### 3.9.1 Framework

The framework gives the sample code access to a few different objects:

**Atoms** The structure that holds all the information concerning the atomic positions, velocities and accelerations. This is the primary source of information from which most quantities are calculated.

**Plotter** A convenience class used for managing different kinds of plots that might be generated during the calculations. Using this class removes any need for the sample code to handle direct disk input/output.

**Result Writer** A convenience class that manages all the calculated quantities. It handles all the storing of the data and gives an easy way of presenting them.

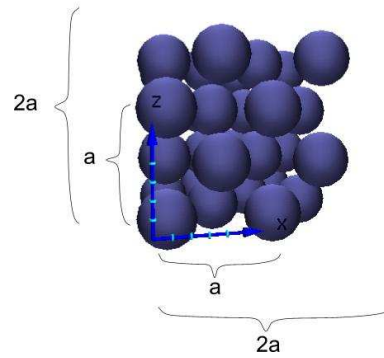
**Boundary handler** A class that manages all information about boundary conditions. Described in section 3.6 on page 23.



**Material handler** A class that contains all material properties needed for the different calculations.

### 3.10 Initialization of the geometry

For most solid-state simulations the atoms need to be placed in a crystal lattice. This is very easy for a cubic unit cell and can be done by just putting out unit cells of the material into a block. For these systems it is also straightforward to comply with the periodic boundary conditions because of the symmetry of the unit cells. In figure 3.6 this is shown. This works very well for doing bulk simulations but the only surface that can be generated is the (1,0,0) surface.



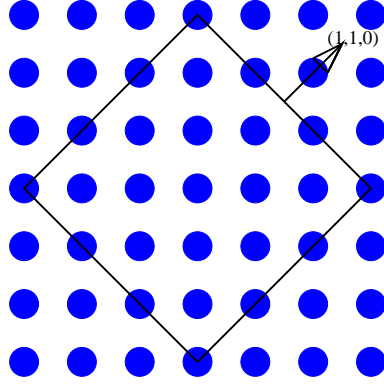
**Figure 3.6.** A small fcc bulk with the side length  $2a$ . Due to the periodic boundary conditions there are not atoms placed on the furthest right, top and back sides.

#### 3.10.1 Ad hoc method for making advanced surfaces

When doing surface simulations the physics can be very different depending on which surface are studied. To make different simulations, e.g. (1,1,1) surfaces, possible we need another way to place the atoms.

##### First try

The first approach was to rotate the bulk and "cut" out a smaller part with the wanted surface upwards, as can be seen in figure 3.7 on the next page. This did not work due to the periodic boundary conditions together with numerical instabilities. The cutting would often be placed in the exact position of an atom that, due to periodic boundary conditions, would be placed on both boundaries. With numerical instability both, or none, of these would sometimes be placed inside the box. Even if we got the correct amount of atoms inside the cut, the surface would still be non-flat due to some atoms would be placed in the bottom layer and some on the surface. We did not find a way to solve this so we instead tried to come up with another solution.



**Figure 3.7.** An example in 2D of how to cut out a rotated surface from a bigger bulk.

The idea behind the solution is to find out how to build up each layer of the material with the surface normal as the wanted miller indices. And then how each layer is translated in relation to the others.

### Spanning a layer

The way we do this is to build an experimental bulk, in the wanted crystal structure, pick out one atom, and then find all other atoms in the same plane with the wanted miller indices. If the first atom is in position  $\bar{r}_0$  we then go through all other atoms at positions  $\bar{r}_i$  and the ones in the same layer as the first atom, with the surface normal  $\hat{n}$ , are those atoms that fulfil

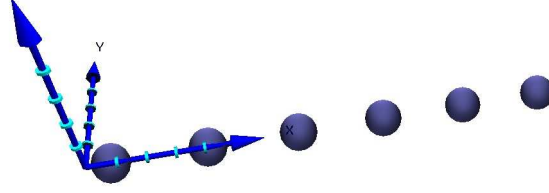
$$(\bar{r}_i - \bar{r}_0) \cdot \hat{n} = 0. \quad (3.3)$$

The thing now is to find the two shortest non-parallel  $\bar{r}_i - \bar{r}_0$  of the atoms in the layer. This is done with an ordinary sorting algorithm. With these two vectors  $\bar{r}_i - \bar{r}_0$  and  $\bar{r}_j - \bar{r}_0$  we can span the whole layer. We do this by putting the next shortest vector along the  $x$ -axis, we can call it  $\bar{r}_x$  and transforming the other vector so that it lies in the  $xy$  plane, call it  $\bar{r}_{xy}$ . Figure 3.8 on the facing page shows the line of next nearest neighbours along the  $\bar{r}_x$  direction and figure 3.9 on the next page also shows the line of nearest neighbour atoms along the  $\bar{r}_{xy}$  direction.

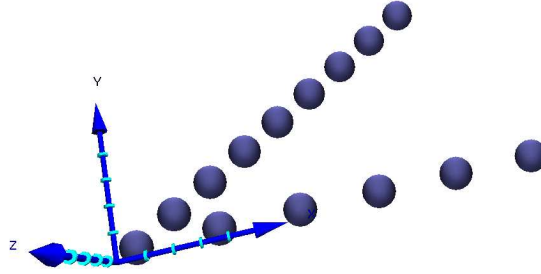
By putting one of the next nearest neighbour lines for each nearest neighbour atom we will now get a full layer as seen in figure 3.10 on the facing page.

### Periodic boundary conditions for the layer

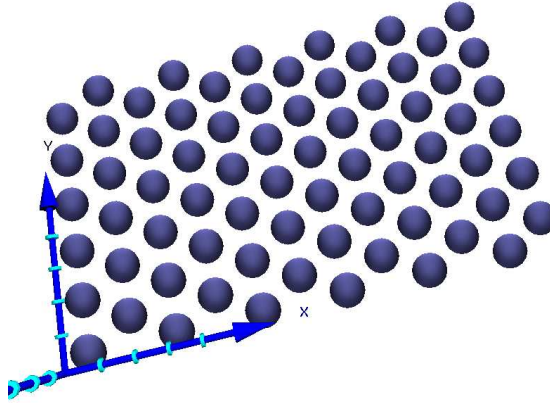
It is very important to satisfy the periodic boundary conditions in the  $x$  and  $y$  direction. For the  $x$  direction this can be done by only letting the  $x$ -size be whole multiples of  $|r_x|$ . For the  $y$  direction we need to find how many lines of atoms ( $N$ ) we need to put before one line is completely identical, regarding the  $x$  positions



**Figure 3.8.** The first line of atoms that are placed along the x-axis using the next nearest neighbours in the surface. This is the  $(1, 1, 1)$  surface with fcc.



**Figure 3.9.** Shows the line with the next nearest neighbours along the x-axis and the diagonal line of nearest neighbours. This is the  $(1, 1, 1)$  surface with fcc.

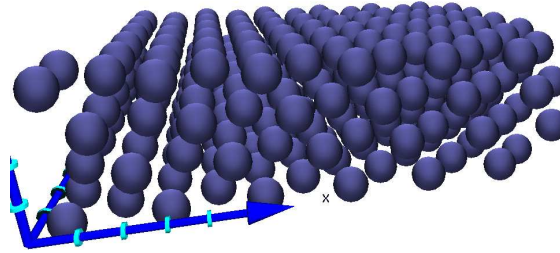


**Figure 3.10.** A  $(1, 1, 1)$  surface fcc layer.

of the atoms, as the line with  $y = 0$ . In figure 3.10 we can see that this period is equal to two. The required  $y$  size will be multiples of  $N(\bar{r}_{xy} \cdot \hat{y})$ .

### Stacking layers

To build a bulk from more than one layer we need to find the sequence of layers before we are back right on top of the first layer. We also need descriptive vectors of the translation between the new layer in relation to the one before. All this can be gotten from the test bulk using similar techniques as when building a layer. First the period of the layers needs to be found by going through layers until the full sequence of layers is found. It is then enough to find the position of one of the atoms in the layer to build it using the method described above. There are no periodic boundary conditions when building this surface so it is a lot easier in this direction. Figure 3.11 shows the final surface box for a (1,1,1) surface.



**Figure 3.11.** A 4 layer (1, 1, 1) surface with periodic boundary conditions in the  $x$  and  $y$  directions

## 3.11 Summary

The main idea in the chapter is modularity, it has been the main focus during the design of this software package. In this chapter we have described several different frameworks that have been designed to accomplish this modularity. In the next chapter we will give you an example of how this framework and software can be set to work.

## Chapter 4

# Simulation Example

In this chapter we give a brief example of how the software can be used. For more explanation on the different settings used see the User Guide in appendix A on page 43. A small bombardment simulation of a (1,1,1) platinum surface is configured. Then the simulation is run and the results are presented. At the end there is a discussion about the performance of the software.

### 4.1 Initiation

In this simulation the Embedded-atoms method is used for simulating the interaction between the platinum atoms.

#### 4.1.1 Simulation parameters

The simulation is set to run for 10000 time steps at a rate of 1 fs per time step and status should be reported every hundred time step. A backup should be taken every five hundred time step. The temperature is held at 300 degrees Kelvin with an Andersen thermostat at an activity of 0.01. In the z-direction the periodic boundary conditions are turned off with no extra space for neighbours.

In the calculations section only the temperature is left *on*, since the others do not make any sense for a surface simulation. In the next section the visualization is turned *on*, the storing frequency is set to five and the *store velocities* button is checked so that all data is stored.

#### 4.1.2 Potential parameters

For the potential parameters for the simulation we have used those published by Johnson[6] as seen in table 4.1 on the following page. The cutoff is set to about 0.86 times the lattice constant, some where in between the nearest neighbour and the next nearest neighbour, of platinum.

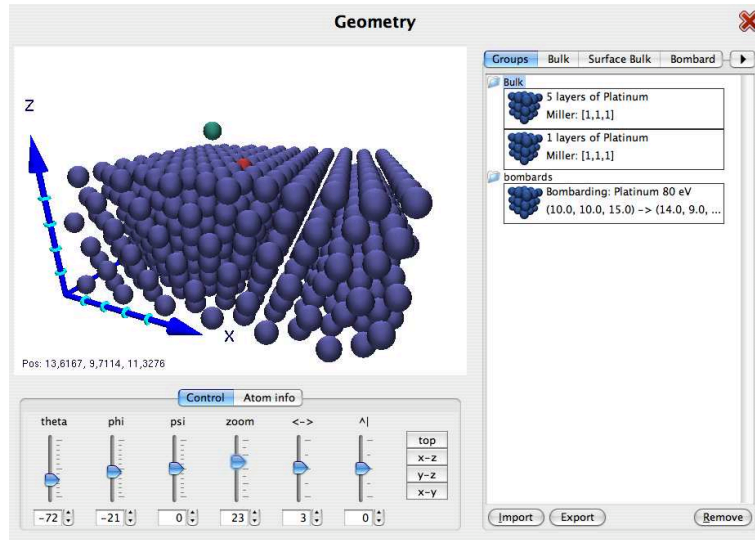
Mass [u]	a [ $\text{\AA}$ ]	$\Omega$ [ $\text{\AA}^3$ ]	$E_c$ [eV]	$E_{UF}$	$\Omega_B$ [eV]	$\Omega_G$ [eV]	$\beta$
195.078	3.925	15.06	5.77	1.6	26.6	6.12	6.69

**Table 4.1.** The Johnson[6] parameters for platinum.

### 4.1.3 Geometry

In the geometry a surface of platinum is configured with the (1,1,1) surface in the  $z$  direction, see figure 4.1. The bottom layer of the bulk is locked to maintain the position of the surface.

On top of the bulk we add a bombarding platinum atom with a kinetic energy of 80 eV. It is set to emerge 500 fs into the simulation from the point (10, 10, 10)  $\text{\AA}$ . Aiming at the point (14,9,11.3) located on this side of the bombarding atom in the figure.



**Figure 4.1.** Simulation geometry configuration. One platinum atom bombarding a platinum (1,1,1) surface.

## 4.2 Simulation

The simulation is started by simply clicking the simulate button. During the simulation information about the simulation is presented in the log panel see figure 4.2 on the facing page. The current temperature, total energy and potential energy are presented.

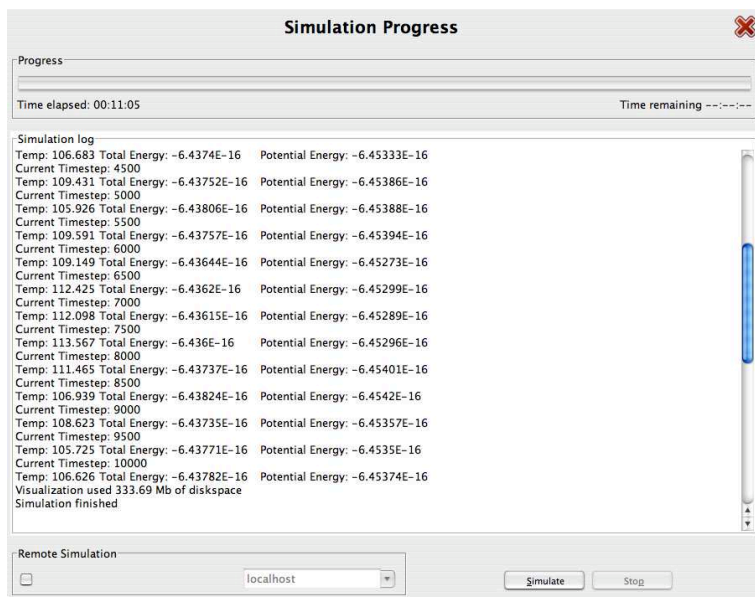


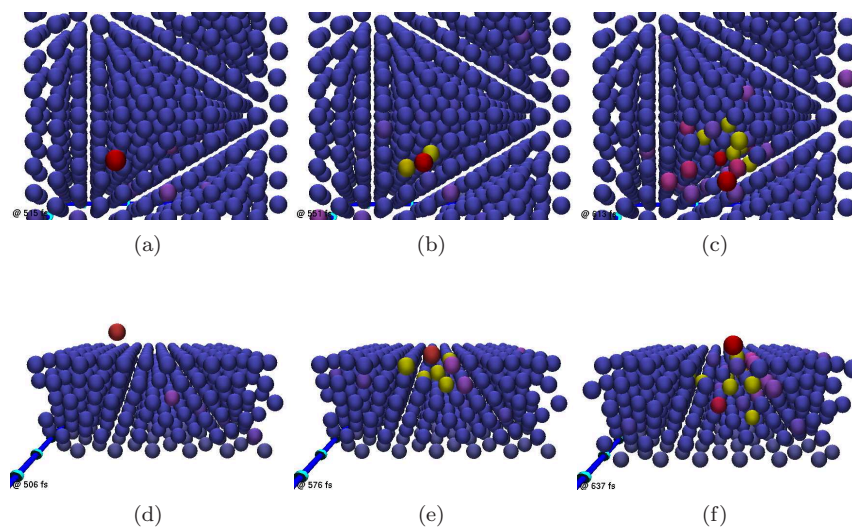
Figure 4.2. View of the simulation progress

## 4.3 Results

The simulation results are presented in the *results*, *plot* and *visualization* panels. In this simulation the most interesting result is in the visualization panel. In figure 4.3 on the next page two different views of the impact of the bombarding atom are shown.

## 4.4 Summary

In this chapter a short presentation of how a simulation could look in our software is given. In the next chapter we go on discussing different performance aspects of the program. A small benchmark is presented confirming the good linearity of our implementation.



**Figure 4.3.** Visualization of a bombarding platinum atom (red) on to a (1,1,1) platinum surface. Figure a to c from the top view, figure d to c from in a cross section view. (For interpretation of colour references please refer to the web version of this report.)



# Chapter 5

## Performance

Simulation performance is one important factor in any molecular dynamics software. Although in our implementation the main focus has not been on high performance but rather good modularity and flexibility. However in this chapter performance aspects of the software are discussed and a small benchmark is presented.

### 5.1 Benchmark

The benchmark has been carried out on a Dell workstation PWS450 with a Intel Xeon 2.4 GHz CPU and 2 GB of RAM. For the simulation the embedded-atom method is used and a cubic block of platinum. Each simulation has been carried out over 1000 time steps with a length of 5 fs.

#### 5.1.1 Time per atom and time step

The following measurements has been made, see table 5.1 on the following page and figure 5.1 on the next page. The most important feature to notice in this figure is perhaps the good scalability of the program. As can be seen the software scales almost linear with the number of atoms used in the simulation.

Using the linear approximation in figure one finds that on average a time step for one atoms takes about 35  $\mu s$ .

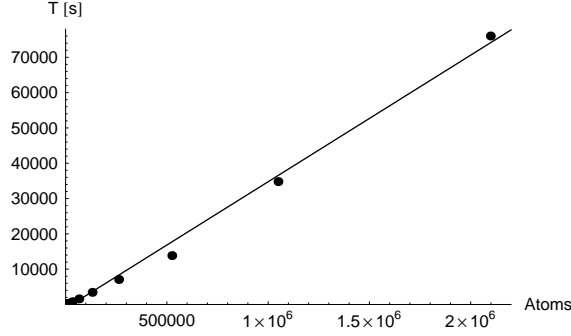
### 5.2 Cutoff length

The number of interactions between atoms in a simulation is strongly dependent on the length of the cutoff  $r_c$ . In a simulation using neighbour cells with the average number of atoms per volume  $\rho$  the number of interactions per atom  $\kappa$  is approximately

$$\kappa \approx \rho(2r_c)^3 = 8\rho r_c^3 \quad (5.1)$$

Atoms	time [s]
2048	55
4096	97
8192	195
16238	385
32768	781
65538	1588
131072	3444
262144	7075
524288	13846
1048578	34797
2097152	76041

**Table 5.1.** Performance test of a Embedded-atom simulation



**Figure 5.1.** Time as a function of the number of atom in the simulation. Simulated using the embedded-atom method and for 1000 time steps

In a simulation with  $N$  atoms, the total number of interactions  $\Gamma$  would then be

$$\Gamma \approx N\kappa = 8N\rho r_c^3. \quad (5.2)$$

The number of interactions is in turn directly proportional to the computational time  $T_{sim}$  required to run the simulation. So the important result is

$$T_{sim} \propto r_c^3. \quad (5.3)$$

### 5.2.1 Three-body Potentials

In a three-part potential the number of interaction increase drastically. Every pair inside the cutoff has to interact with every other atom inside the cutoff. Hence the total number of interactions for one atom is

$$\kappa \approx \rho(2r_c)^3 \times \rho(2r_c)^3 = 64\rho^2 r_c^6. \quad (5.4)$$

And the total number of interactions

$$\Gamma \approx 64N\rho^2r_c^6. \quad (5.5)$$

Thus the total time is proportional to

$$T_{sim} \propto r_c^6. \quad (5.6)$$

This shows how important is to carefully choose a value for the cutoff.

### 5.3 Tabulation

In many cases performance can be improved by using tabulation of computational expensive mathematical expressions. The main limits in using tabulation are memory consumption and initialization time. In a molecular dynamics simulation the time to initialize the tables is usually negligible. Although the memory consumption might not be, especially if the function to tabulate depends on more than one variable, as is common in many three-body potentials. Say for example a function  $f(r_i, r_j, r_k)$  is to be tabulated in the range  $[0, r_c]$  with 1000 points for each dimension. Such a tabulation would result in  $1000^3 * 8 = 8$  GB of memory, something that most computers today are not capable of handling.

### 5.4 Numerical vs. Analytical Derivatives

One way to speed up the development of a new potential can be to use numerical derivatives instead of analytical derivatives. The numerical calculation may often require a bit more calculation due to the fact that the potential must be evaluated on many more points to get a result that are as good as with an analytical derivative. Even so it can be useful in many cases due to the often much faster implementation of the potential. At least in the early stages development and testing.

### 5.5 Data generation

The result in the previous section showed that in average the software simulates one new movement for about 27000 atoms every second. Taking into account that to store all the information generated for one atom in one time step, position and velocity, the program uses about 49 bytes. Combining this lead to that on the computer used the program generates about 5 GB of data every hour.

### 5.6 Summary

In this chapter some aspects of the performance of the software has been discussed. The most important things might be that the software scales linear with the number of atoms and how important it is to choose a good value for the cutoff. In the

following, and final, chapter in the report the future of the software is discussed and some conclusions are made.

## Chapter 6

# Discussion

This chapter treats the future possibilities of the project. Shines some light on the lessons learned during the design and implementation. Finally some conclusions are drawn from the work.

### 6.1 Future

For the future there are many aspects that can be improved, here we present some of the ideas that we have come up with.

#### 6.1.1 Plugin based potentials

One of the most important parts of any molecular dynamics program is the potential code. This is also a part that is likely to change a lot. Often one wants to develop and test new potentials for different systems. And then, it is of course preferable if this can be done with a minimal effort.

In MDSinecura the potential code is very modular and easy to modify, see the information on the potential framework in section 3.7.1 on page 24, but one has to recompile the whole program to test any change in a potential. We think that this could be implemented in a more user friendly way.

Using potential plugins one can change the potential code in runtime. Only the potential code has to be compiled for each test, not the whole program. And it would be easy to maintain a big library of different potentials. One could exchange different potential implementations without having to exchange the whole program. This would certainly result in a much faster and more efficient way to develop new potentials.

Implementing this into the current code would not require much work since most of the existing framework still could be used and the only part that has to be written from ground up is the plugin loading facility. However this code could be reused in other parts of the program, so the time required might be well spent here.

### 6.1.2 Plugin based sample calculations

In a molecular dynamics simulation there is a huge amount of information generated. One of the hardest tasks is to extract the important bits of information, see section 2.5 on page 14 about what properties can be calculated. Often these properties are calculated during the simulation. This could of course also be calculated after the simulation is finished although this will require that all the data generated during the simulation is stored, this will often require a huge amount of storage space, therefore it is often preferred to do the calculations during the simulation even though the simulation will consume more time.

Mainly for the same reasons as for the potential there are good incentives for making the sample calculation framework plugin based. One would then be able to develop, test and exchange different sample codes in a much easier manner.

To implement plugin based sample calculations are probably harder then the above mentioned potential plugins mainly because it is harder to modularize the code in a good way and still not loose too much speed. Still, we do not see it as a impossible task.

### 6.1.3 Modified embedded-atom method

One of the main aspects for further development would be implementing the modified embedded-atom method developed by Baskes[3]. It has many interesting aspects. Perhaps the most fascinating is the ability to simulate as many as 26 different elements:

fcc	Cu	Ag	Au	Ni	Pd	Pt	Al	Pb	Rh	Ir
bcc	Li	Na	K	V	Nb	Ta	Cr	Mo	W	Fe
dia	C	Si	Ge							
gas	H	N	O							

This would of course enable one to do many interesting simulations. The potential formulation uses only simple analytical mathematical functions so it should not be too computationally expensive.

### 6.1.4 Samples

Our work has so far mostly been focused on implementing the different potentials that we have had in mind and building a good framework for doing the simulations. There has not been so much time to write code for calculating more than basic physical quantities. With the design we have these can be easily incorporated into the package.

## 6.2 Conclusion

We have successfully been able to design a molecular dynamics package that is modular, flexible and easily extendible. Almost every design choice we have made has been towards flexibility instead of optimizing for speed, but despite this we

can show good performance and the possibility to make both large and long simulations. On top of this, we have been able to build a user friendly and intuitive graphical interface with advanced features for configuration and visualization.

Second to the design of the modular framework our focus has been on implementing a few fundamental potentials that open up possibilities for a wide range of simulations. This intention has been accomplished with the implementation of the four potentials described in the report. The ease to implement these potentials has also verified the successful design of our potential framework.





# Appendix A

## User Guide

This guide provides information needed to use MDSinecura. The main parts that are described include: configuring the simulation to your needs, building a basic geometry, running the simulation and analysing the results.

### A.1 System Requirements

To run the software package MDSinecura the following requirements have to be met:

**Java 1.5 runtime environment** The software is developed in Java 1.5, also known as Java 5.

**Java OpenGL bindings** have to be supported on the platform used.

The following optional dependencies for extra functionality exist:

**Java 1.5 jdk** The Java development kit allows for more control over the Java runtime environment and allows for use of the `-server` switch presented below.

**Quicktime for Java** A part of the Quicktime<sup>1</sup> program. Required to use the movie capture feature in the visualization.

The software package will run on any platform that meets these requirements. Although it has only been tested on Windows and Mac OS X so far. The memory and CPU power required is highly dependent on the simulation configuration.

### A.2 Starting the program

To start the program simply click the executable MDSinecura.exe in Windows or the MDSinecura.app bundle on Mac OS X. Another way to start the program would be to run the MDSinecura.jar directly:

---

<sup>1</sup><http://www.apple.com/quicktime/>

---

**Example A.1**

---

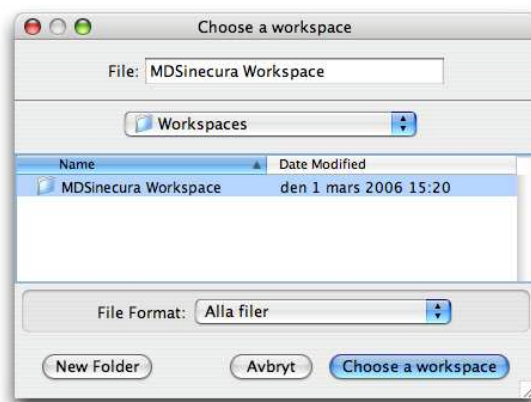
```
java -jar MDSinecura.jar
```

For better performance some more switches can be useful:

- server** Puts the Java virtual machine in server mode making it perform better. It requires that the Java development kit is installed.
  - Xmx100M** Will set the maximum size of the Java heap size to 100Mb, For running large simulation this should be set to the largest allowed value.
- 

### A.2.1 Select a Workspace

The first time the program starts it will display a dialog, see figure A.1, and ask you for a *workspace*. This is where all the simulations will be stored, so make sure there is enough space. A simulation may use anything between 1MB of space up to and beyond 100GB. This typically depends on how long the simulation is and the number of atoms simulated and also on the amount of data saved during the simulation. A general rule is that the visualization requires about 25-50 bytes of data per atom and time step depending on the configuration used. For more information see A.3.3 on page 47. The workspace directory can be changed at a later time using the workspace item under the file menu.



**Figure A.1.** Dialog for selecting a workspace.

### A.2.2 Create a new project

The first thing to do when MDSinecura has launched is to create a new *project*. Pressing File -> New -> New Project will bring up a dialog, see figure A.2, to create a new project. A project is used to organize similar simulations into folders. One may for example have a project called “Silver” with several simulations of silver with different lattice constants.

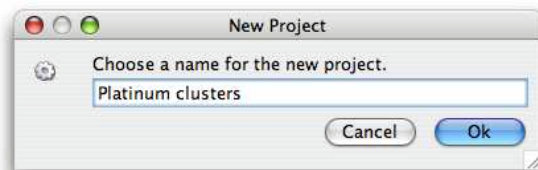


Figure A.2. Dialog for creating a new project.

### A.2.3 Create a new simulation

Once a project is created one can make a new *simulation*. To do this, choose File -> New -> New Simulation. This will bring up a dialog, see figure A.3, asking for the potential to use. The following potentials can be used as of now.

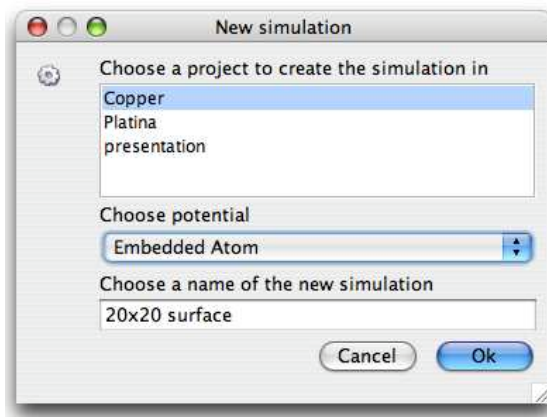


Figure A.3. Dialog for creating a new Simulation.

**Lennard Jones** The most primitive potential implemented, useful for noble gases Argon, Neon, Krypton and Xenon.

**Embedded Atom Model** Useful for metallic compounds of Silver, Gold, Copper, Nickel, Palladium and Platinum.

**Stillinger Weber** A potential that can handle the covalent bond of Carbon, Silicon and Germanium.

**Tersoff** A potential the are able to handle compounds of Carbon, Silicon and Germanium.

## A.3 Configure the simulation

The first thing to do when starting a new simulation is to open the “Config” item where all the simulation configurations presented.

### A.3.1 Simulation

This view presents some general simulation parameters.

**Time step** The size of each time step to be used by the simulation integrator. As a general rule you need a shorter time step with increasing energy. In normal cases a time step between one and five femtoseconds will work well.

**Number of time steps** The number of iterations that should be simulated.

**Steps between status** How often to output simulation status to the log window.

**Store backup** Store data for recovery of a crashed simulation. With this a simulation is restartable. It will demand some extra space and time during the simulation.

**Steps between backup** How often to store backups of the simulation.

**Thermostat** What thermostat to use, if any. Andersen, velocity scaling or none can be chosen.

**Andersen activity** The probability that an atom interacts with the heat bath during a time step. If it is 1 then all atoms will interact each time step and get a new random velocity.

**Scaling activity** The number of time steps between the scaling of all velocities.

**Simulation temperature** The temperature that the simulation is initiated at and that to which the thermostat strives.

**Turn of pbc in z-direction** If a surface is simulated there should not be any periodic boundary condition (pbc) in the z-direction.

**Extra z-size for neighbour cells** The neighbour cells are created to fill up the bulk of atoms. If there are many atoms above the bulk (e.g. surface structures or surface growth) then extra neighbour cells can be added above the surface to increase the speed of the simulation. Warning! Don't use this unless there are a lot of atoms above the surface.

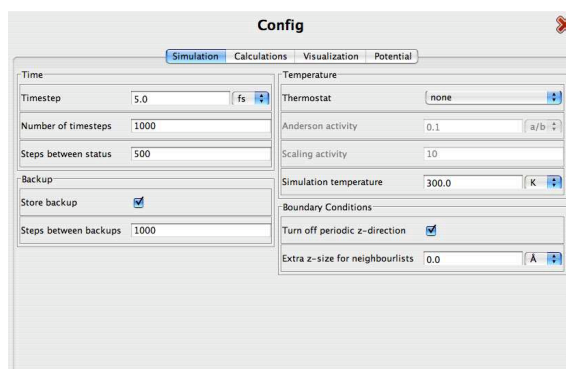


Figure A.4. All general simulation parameters are configured in this view.

### A.3.2 Calculations

A presentation of the different physical properties that can be calculated:

**Temperature** Calculates the instant temperature of the simulation.

**Specific heat (cv)** Calculates the specific heat of the system. Only useful in bulk simulations.

**MSD** Calculates the Mean Square Displacement.

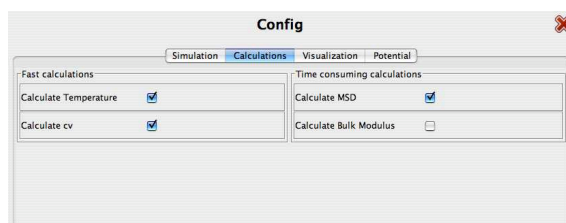


Figure A.5. In the calculation configuration view the properties to calculate are set.

### A.3.3 Visualization

The visualization has a few parameters that can be configured here:

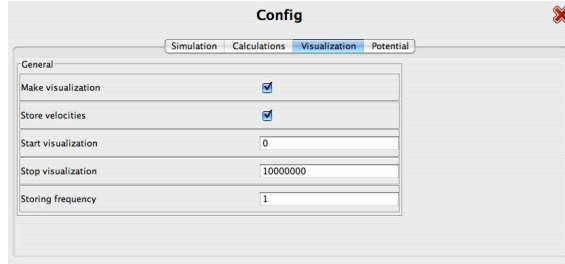
**Make visualization** Turn on or off all visualization. The visualization consumes a lot of space, about 25 bytes per atom and time step.

**Store velocities** Velocities can be shown in the visualization but then they have to be saved during the simulation and that will require 24 bytes of extra storage per atom and time step.

**Start visualization** At which time step to start storing visualization data.

**Stop visualization** At which time step to stop storing visualization data.

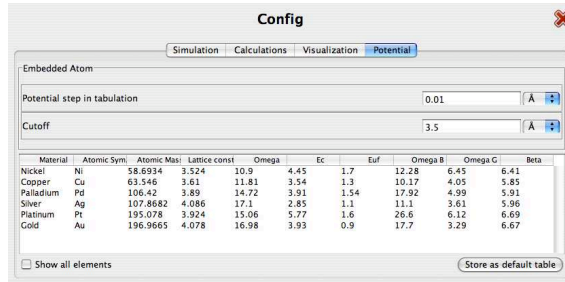
**Storing frequency** How often to store visualization data.



**Figure A.6.** In the visualization configuration view different parameters for the visualization are set.

### A.3.4 Potential

In this view all the potential dependent parameters are entered. Which parameters, that are requested, depends on which potential is used, but the field for “Material”, “Atomic Symbol” and “Atomic Mass” is always present. In the normal case only the materials with filled in parameters are shown. If another material is to be added, clicking the “Show all elements” check box will reveal all the elements of the periodic table.



**Figure A.7.** In the potential configuration all the different potential parameters are set.

## A.4 Setting up a geometry

In the geometry view, figure A.8 on the facing page, the initial positions and element types of all atoms are specified. Special properties are set i.e. bombarding atoms energies.

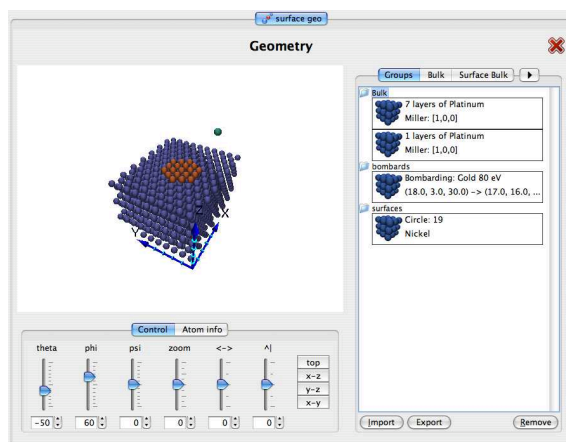


Figure A.8. The geometry view used to build the simulation box

#### A.4.1 Control

There are six sliders used to navigate in the view, seen in figure A.9. From left to right: the first three are for rotation, the last three for translation. There are also field for manually entering positions. At the right end there are four buttons for quick access to the most common views.

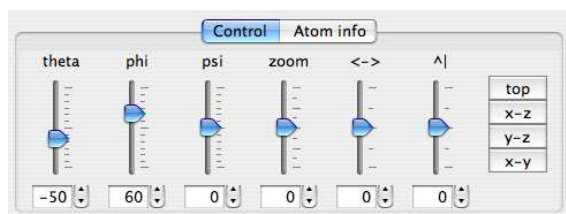


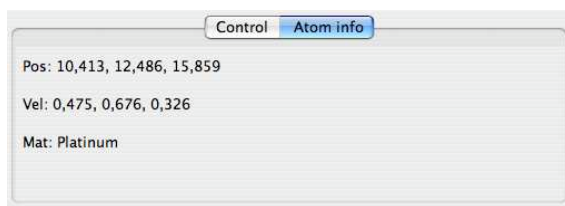
Figure A.9. The 3D view is controlled from this view

#### A.4.2 Atom Info

If an atom is highlighted, by being clicked on, information about that atom will be presented in this panel, figure A.10 on the following page.

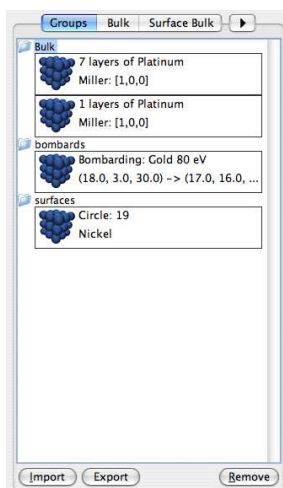
#### A.4.3 Groups

The group panel, see figure A.11 on the next page, holds an overview over the different groups of atoms that are used. The main categories are bulk atoms, surface atoms and bombarding atoms. The different groups can be highlighted by



**Figure A.10.** Clicking on a atom in the 3D view will result in information being presented about the atom in this view

clicking on them. Double-clicking will bring the group into edit mode. The group can also be removed when highlighted by clicking the remove button.



**Figure A.11.** The Group panel in the geometry view, gives a summery of all the different part of the geometry.

## Import

The import button brings up a dialog, see figure A.12 on the facing page, asking you for a file. After that you will be asked to set the quantity and unit for each column in the file. The result will be imported in the bulk.

## Export

The export button brings up a dialog, seen in figure A.13 on the next page, asking for a file to export the atomic configuration to. A second dialog will ask what unit to use when writing to the file and in which order to write the data.



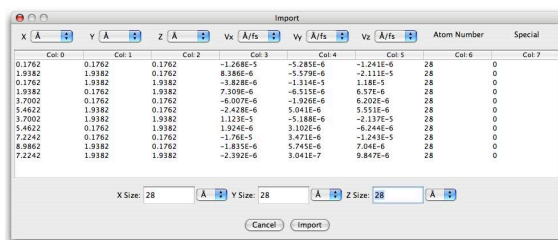


Figure A.12. Dialog for importing atoms positions and velocities to the geometry.

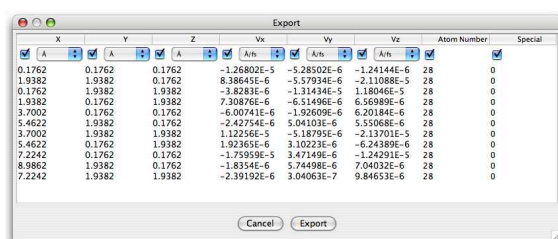


Figure A.13. Dialog for exporting atom positions and velocities from the constructed geometry.

#### A.4.4 Bulk

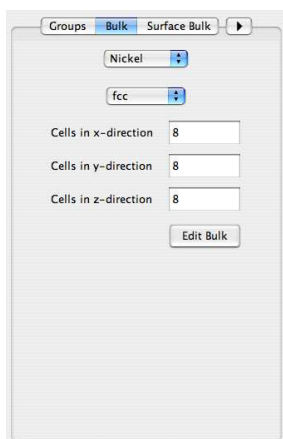
The bulk panel, see figure A.14 on the following page, is used for generating an atomic configuration for a bulk system of one element in either fcc or diamond structure. The inputs needed are the number of unit cells in each direction. When generated each atom is assigned a Gaussian velocity. All atoms will be of the “common” type, see 3.3 on page 21.

#### A.4.5 Surface Bulk

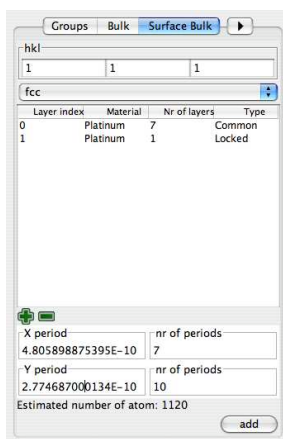
The surface bulk panel, see figure A.15 on the following page, is used for generating atomic configurations for surfaces and it supports an arbitrary orientation of the surface specified in Miller indexes. The different properties such as locked atomic positions and scaled temperatures can be set at a per layer basis. For more information see 3.10 on page 27

#### A.4.6 Atom Bombard

A simple panel, see figure A.16 on page 53, for adding a bombarding atom to the simulation. Supply a starting point, a starting time, an energy and a target point.



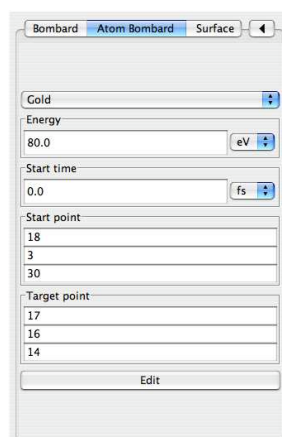
**Figure A.14.** The basic bulk generation panel.



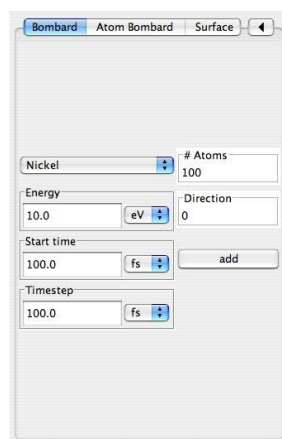
**Figure A.15.** The surface bulk panel used for generating a slab of atoms to do different kinds of surface on.

#### A.4.7 Bombard

The bombard panel, see figure A.17 on the next page, is used for adding bombarding atoms to the simulation. The atoms will be positioned randomly over the surface with a velocity parallel to the normal of the surface and speed accordingly to the chosen energy. The bombardment start time, energy, number of bombarding atoms and the time in between the atoms can be specified.



**Figure A.16.** The atom bombard panel is used to add one bombarding atom stating at a point, aiming at an other point, with a specified energy.



**Figure A.17.** The bombard panel is used to add many bombarding atoms in a fast and easy way.

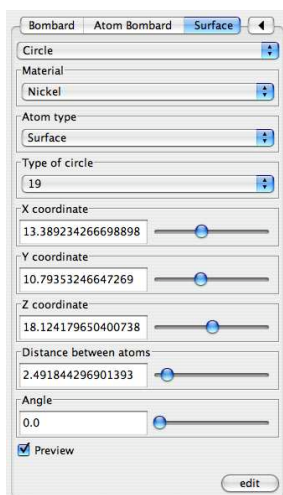
#### A.4.8 Surface

The surface panel, see figure A.18 on the following page, is used to build different surface structures. Four kinds of predefined structures exists:

**Single Atom** Used for adding single atoms to the surface.

**Line** Used for adding lines of atoms, the number of atoms, the spacing and the relative angle of the line can be specified.

**Circle** Used for adding a hexagonal shape of atoms, two variants are available the seven atom cluster and the 19 atom cluster. The atom spacing and the



**Figure A.18.** The surface panel is used to add small structures on the surface of a bulk.

relative angle of the cluster can be specified.

**Triangle** Used for adding triangular shapes of atoms to the surface. Filled triangles with 3 to 21 atoms are supported. The atom spacing and the relative angle of the triangle can be specified.

## A.5 Running the simulation

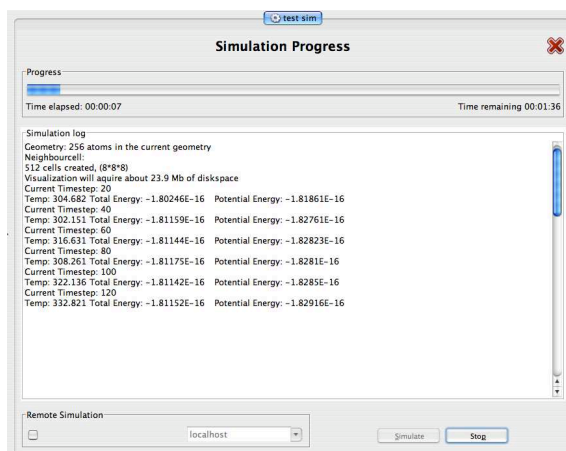
In the simulation view, see figure A.19 on the next page, simply pressing the *Simulate* button will start the simulation. The progress bar will provide information on the progress of the simulation. The log view will give instant information from the running simulation, at a variable rate set in A.3.1. A line of information worth mentioning is the prediction of space required by the visualization. You have to make sure there is enough space available or the simulation might abort. Pressing the *Stop* will make the simulation come to a halt in a controlled way, the simulation will still finish the current time step and store all already generated data.

### A.5.1 Remote Simulation

Enter the address to the remote server on which the simulation should run.

## A.6 Viewing the Results

A simulation will generate a lot of data, there are three main ways of viewing the result. The basic results table A.6.1, the plots A.6.2 on the facing page and the visualization A.6.3.



**Figure A.19.** The simulation view where the simulations are started and stopped and information about the progress of the simulation is presented.

### A.6.1 Results

In this view, see figure A.20, calculated quantities are presented.

**Simulation Result**

Quantity	Value	Unit
Temperature	277.5496054019537	K
Cohesive energy	-5.609158000081203	eV

**Figure A.20.** The result view, here different calculated quantities presented.

### A.6.2 Plots

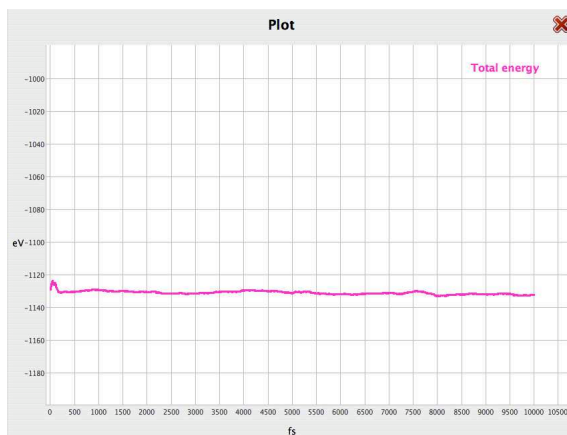
In this view, see figure A.21 on the following page all plots generated by the simulation is presented. The plots can be viewed one by one or several together.

### A.6.3 Visualization

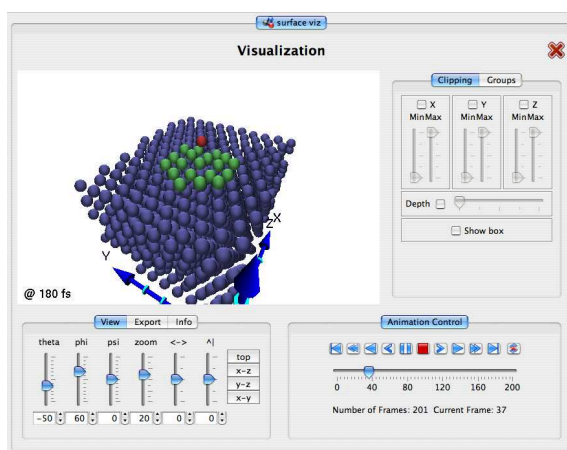
In this view, see figure A.22 on the next page all of the simulation can be viewed if configured in the configuration A.3.3. There are several different tools to use to make the most out of the data.

**View** This panel provides six slides to navigate in the simulation.

**Export** This panel provides options to export images and movies from the simulation.



**Figure A.21.** The plot view is used to view different data collected during the simulation.



**Figure A.22.** The visualization view can present the time development of the simulation in a user-friendly way.

**Info** In this panel information about highlighted atoms is presented. To select a secondary atom use ctrl-click.

**Clipping** In this panel limits on what regions of the simulation box to show can be set.

**Groups** In this panel a group of atoms can be assigned a colour gradient according to their velocity.

**Animation Control** This panel works like an ordinary media controller allowing animation of the atomic positions.

# Bibliography

- [1] M. P. Allen and D. J. Tildesley. *Computer simulation of liquids*. Clarendon Press, New York, NY, USA, 1989.
- [2] Hans C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of Chemical Physics*, 72(4):2384–2393, 1980.
- [3] M. I. Baskes. Modified embedded-atom potentials for cubic materials and impurities. *Physical Review B*, 46:2727–2742, August 1992.
- [4] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. Academic Press, Inc., Orlando, FL, USA, 2001.
- [5] Herbert Goldstein, Charles P. Poole, and John Safko. *Classical Mechanics*. 3 edition, 2002.
- [6] R. A. Johnson. Alloy models with the embedded-atom method. *Physical Review B*, 39:12554–12559, June 1989.
- [7] P.-S. Laplace. *A Philosophical Essay on Probabilities*. Dover, New York, 1951.
- [8] F. H. Stillinger and T. A. Weber. Computer simulation of local order in condensed phases of silicon. *Physical Review B*, 31:5262–5271, April 1985.
- [9] J. Tersoff. Modeling solid-state chemistry: Interatomic potentials for multi-component systems. *Physical Review B*, 39:5566–5568, March 1989.
- [10] Wikipedia. Moore’s law — wikipedia, the free encyclopedia, 2006. [Online; accessed 1-February-2006].

## List of Figures

1.1	Moore's law, and actual development. . . . .	1
2.1	The MD loop with only the most important parts. . . . .	6
2.2	Lennard-Jones potential . . . . .	8
2.3	Two-body SW interaction . . . . .	10
2.4	The Tersoff potential function . . . . .	11
2.5	The Tersoff cutoff function . . . . .	11
2.6	The Tersoff potential function with environment dependence . . .	12
2.7	The angular dependence of the Tersoff potential . . . . .	12
3.1	The MDSinecura main loop. . . . .	17
3.2	Different approaches to handling neighbours . . . . .	18
3.3	Neighbour cells using half cutoff sized cells. . . . .	19
3.4	Example of atom types. . . . .	22
3.5	Overview of the potential framework. . . . .	25
3.6	A small fcc bulk with the side length $2a$ . . . . .	27
3.7	An example of how to cut out a rotated surface. . . . .	28
3.8	First line of atoms. . . . .	29
3.9	Nearest and next nearest neighbour lines. . . . .	29
3.10	A (1, 1, 1) surface fcc layer. . . . .	29
3.11	A 4 layer (1, 1, 1) surface. . . . .	30
4.1	Example geometry configuration. . . . .	32
4.2	View of the simulation progress. . . . .	33
4.3	Visualization of a bombarding atom. . . . .	34
5.1	Time as function of the number of atoms . . . . .	36
A.1	Dialog for selecting a workspace. . . . .	44
A.2	Dialog for creating a new project. . . . .	45
A.3	Dialog for creating a new Simulation. . . . .	45
A.4	Simulation configuration view. . . . .	47
A.5	Calculation configuration view . . . . .	47
A.6	Visualization configuration view . . . . .	48
A.7	Potential configuration view . . . . .	48
A.8	The geometry view . . . . .	49
A.9	The control view . . . . .	49
A.10	Atom info view . . . . .	50
A.11	The Geometry group panel. . . . .	50
A.12	The geometry import dialog. . . . .	51
A.13	The geometry export dialog. . . . .	51
A.14	The geometry bulk panel. . . . .	52
A.15	The geometry surface bulk panel . . . . .	52
A.16	The geometry atom bombard panel . . . . .	53
A.17	The geometry bombard panel . . . . .	53



A.18 The geometry surface bulk. . . . .	54
A.19 The simulation view. . . . .	55
A.20 The result view. . . . .	55
A.21 The plot view. . . . .	56
A.22 The visualization view . . . . .	56

## List of Tables

3.1	The Atom types . . . . .	22
4.1	The Johnson[6] parameters for platinum. . . . .	32
5.1	Performance test of a Embedded-atom simulation . . . . .	36

## List of Examples

A.1	. . . . .	43
-----	-----------	----

## List of Listings

3.1	Using neighbours for two body potential . . . . .	20
3.2	Tabulation . . . . .	26

## Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

## Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>