ATOMISTIC STUDY OF STRUCTURAL CORRELATIONS AT A MODEL SOLID/LIQUID-METAL INTERFACE

ADHAM HASHIBON

ATOMISTIC STUDY OF STRUCTURAL CORRELATIONS AT A MODEL SOLID/LIQUID-METAL INTERFACE

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

ADHAM HASHIBON

SUBMITTED TO THE SENATE OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY

AV, 5762 HAIFA AUGUST, 2002

THIS RESEARCH THESIS WAS SUPERVISED BY DR. JOAN ADLER, DR. WAYNE D. KAPLAN AND PROF. S.G. LIPSON IN THE DEPARTMENT OF PHYSICS

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my advisors, Dr. Joan Adler, Dr. Wayne D. Kaplan, and Prof. S. G. Lipson, for their excellent guidance and support during this research.

I wish to thank Prof. Michael W. Finnis for stimulating discussions and support during all stages of this research.

I am grateful to Dr. Furio Ercolessi for providing the aluminium interaction potential.

I would like to thank my numerous friends at the Technion, especially my fellow students in the Computational Physics Group, and the Metal Ceramic Interfaces Group, for their encouragement and assistance.

I am grateful to my family for always being there for me. They were a constant source of encouragement.

THE GENEROUS FINANCIAL SUPPORT OF THE LILLIAN AND MILTON EDWARDS FOUNDATION, THE ISRAEL MINESTRY OF SCIENCE, AND THE GERMAN ISRAELI FOUNDATION (GIF) IS GRATEFULLY ACKNOWLEDGED

Dedicated to my mother Nadia.

Contents

Abstract **1**

1 Introduction **4**

2 The liquid-solid interface structure **7**

2.1 Analytical models 9

2.2 Computational studies 16

2.3 Experimental studies 22

3 The objectives of the research **25**

4 Methodology **27**

4.1 The molecular-dynamics method 27

4.2 The interatomic interactions 29

4.3 Boundary conditions 34

5 Description of the simulation systems **35**

5.1 The structure of the solid substrates 37

5.2 The width of the solid substrates 39

5.3 The initial configurations 43

5.4 System notation 44

5.5 Systems with face-centered-cubic substrates 44

5.6 Systems with body-centered-cubic substrates 46

6 Description of the simulations and calculation of properties 49

6.1 Density profiles 50

6.2 Potential energy 51

6.3 The static structure order parameter 51

6.4 The two-dimensional radial distribution function 52

6.5 Equilibration 53

6.6 Sampling 55

7 Results 57

7.1 Results for the face-centered-cubic systems 57

7.1.1 Quantitative measure of ordering 66

7.1.2 Effect of substrate widths and system size 70

7.1.3 In-plane ordering 73

7.2 Results for the body-centered-cubic systems 75

7.2.1 In-plane ordering 79

8 Discussion 86

8.1 In-plane ordering 94

8.2 The interlayer spacing and density variations across the interface . . 97

9 Summary and conclusions 103

A Density functional method 106

B Description of the simulation programs 112

B.1 Source code 117

References 163

Hebrew Abstract מ

List of Figures

2.1 Various approximations for the wall-fluid interaction as used by Bonissent and Abraham [1]. (Figure is from reference [1]). 11

2.2 Comparing the density profiles obtained with the various approximations for the wall-fluid interaction as used by Bonissent and Abraham [1]. (Figures are from reference [1]). 14

2.3 The averaged density profiles for the bcc 100 (a) and 111 (b) crystal melt interfaces as obtained by Haymat and Oxtoby. (Figures are from reference [2]). 15

2.4 Experimental evidence of liquid layering at a solid-liquid interface by Huisman *et. al.* [3, 4]. The top panel shows the averaged electron density profile in liquid Ga normal to the solid interface, and the lower panel shows a suggested structure for the interface. Large symbols represent Ga atoms, bonds between Ga atoms represent dimers. (Figure is from reference [3]). 24

4.1 The three functions constituting the optimized glue potential for Al. (Figure is from reference [5]). 31

4.2 Evolution of an aluminium liquid drop on a substrate of the same material. The drop on the (110) surface spreads completely into the substrate, while that on the (111) surface does not. (Figure is from reference [6]). 33

5.1 Sketch of the model system, and the width of the solid, W_s, should be larger than the cutoff distance of the potential, $W_s > R_c$. The extent of the liquid region perpendicular to the interface plane, W_l, should be large enough that the free surface does not interact with the solid at the other end of the simulation cell. 36

5.2 A symmetric unit cell for a fcc lattice (a), and a bcc (b). The unit vectors correspond to those in equations 5.2 and 5.3 38

5.3 Top view of the (100) fcc substrate. The last two planes are shown, green is the top most, and red is the layer beneath. 40

5.4 Top view of the (111) fcc substrate, the three top layers are shown, blue is the top most followed by red and green. 40

5.5 Top view of the (110) fcc substrate showing the two terminating planes, (green on top, followed by red). 41

5.6 Top view of the bcc (100) system. Two terminating planes are shown in red and green. 42

5.7 Top view of the bcc (110) system. Two terminating planes are shown in red and green. 42

6.1 An illustration of the division of the simulation cell into bins or slices normal to the interface direction $\hat{n}$, each of width Δz. Property calculations are carried out for each individual bin separately. 50

6.2 An example of an annealing schedule for the fcc(100)-A system (see Table 5.3). The state of the system at the final (target) temperature is compared with the initial state. The schedule is repeated until the system is in equilibrium, that is when the energy and density profile in the final state and initial state are the same. 53

6.3 The profiles of the mean number of atoms $\langle N \rangle$ (top panel), the potential energy per particle u_p, the kinetic energy E_k, and the static structure order parameter $S(\mathbf{K})$ in each bin. Data is from system fcc(111)-B at $T = 1100K$. Typical error bars are indicated. 54

7.1 The density profile $\rho(z)$ as a function of distance z from the hard wall is shown for a (100) fcc system at three temperatures: 1000K, 1200K, and 1400K. Vertical lines mark the locations of the density peaks at 1000 K. Data is from a simulation on system fcc(100)-C described in section 5.5. 58

7.2 The density profile $\rho(z)$ as a function of distance z from the hard wall is shown for a (111) fcc system at three temperatures: 1000K, 1200K, and 1400K. Vertical lines mark the locations of the density peaks. Data is from a simulation on system fcc(111)-B described in section 5.5. . . 59

7.3 The density profile $\rho(z)$ as a function of distance z from the hard wall is shown for a (110) fcc system at three temperatures: 1000K, 1200K, and 1500K. Vertical lines mark the locations of the density peaks. Data is from a simulation on system fcc(110)-C described in section 5.5. . . 60

7.4 Comparison of density profiles for the fcc systems at a temperature of 1000K. The vertical dashed lines mark the positions of the peaks for the (100) system. d_p indicates the interlayer spacing as calculated from the peak data and d_v is the interlayer spacing calculated from the minima, or valleys of the density profile. 61

7.5 The interlayer spacing from the peak data d_p, for the fcc (100) substrate at $T = 1000$ K, $T = 1200$ K, and $T = 1400$ K. 62

7.6 The interlayer spacing from the peak data d_p, for the fcc (111) substrate at $T = 1000$ K, $T = 1200$ K, and $T = 1400$ K. 63

7.7 The interlayer spacing from the peak data d_p, for the fcc (110) substrate at $T = 1000$ K, $T = 1200$ K, and $T = 1400$ K. 64

7.8 The interlayer spacing from the peak data d_p for the three fcc systems at $T = 1000$ K. 64

7.9 The interlayer spacing from the peak data d_p, and from the valley's data d_v, calculated at $T = 1000$ K for the three fcc faces as indicated in the figure. 65

7.10 (a) The envelope of the density profile obtained for a (111) fcc sample at $T = 1250K$, and (b) test for the fitting (see text). 68

7.11 The disorder parameter κ vs. temperature for the three fcc surfaces. The dashed lines are meant to guide the eye only, and are not fits to the data. 69

7.12 Comparing density profiles for the different fcc (100) systems studied at 1100K. 70

7.13 Comparing density profiles for the different fcc (111) systems studied at 1300K. 71

7.14 Comparing density profiles for the different fcc (110) systems studies at 1000K. 72
7.15 The disorder parameter κ vs. temperature for all fcc systems. 72
7.16 The two dimensional radial distribution functions for the fcc(100) system at $T = 1000K$. The RDFs for the different layers are stacked one above the other. 73
7.17 The two dimensional radial distribution functions for the fcc(111) system at $T = 1000K$. The RDFs for the different layers are stacked one above the other. 74
7.18 The two dimensional radial distribution functions for the fcc(110) system at $T = 1000K$. The RDFs for the different layers are stacked one above the other. 75
7.19 Comparing density profiles for the (100) bcc system at different temperatures. 76
7.20 Comparing density profiles for the (110) bcc system at different temperatures. 77
7.21 The profile of the bcc (100) system showing an extra peak very near to the solid peak, which is due to the adsorbed liquid layer. 78
7.22 The interface region at a bcc (100) substrate at $T = 1000K$. The left panel is a side view and the right panel is a top view. The darkest atoms represent the solid, medium and light gray, the liquid. On the right, different sizes indicate layer position: layer (a) is represented by large spheres, layers (b) and (c) by normal size, and layer (c) by a smaller size. Liquid atoms (dark gray) in layer (c) are adsorbed into the solid, they lie in the *face-centered* sites on the bcc (100) plane, but positioned slightly above the (100) plane as seen from the left panel. 79

7.23 The d-spacing for bcc (100) and bcc (110) substrates; in both cases the interlayer spacing inside the liquid is compatible with the periodicity of (111) planes in an fcc aluminium metal. 80

7.24 The disorder parameter κ for bcc substrates. The results are very similar for both the (110) and (100) substrates at $T \leq 1300K$. 81

7.25 (a) Stacking of layers in the bcc (100) system at $T = 1000K$. Light gray circles represent atoms in the terminating bcc (100) plane, dark gray circles the adsorbed layer, diamonds, squares, and triangles represent the first three layers in the liquid. The square connects four atoms of the bcc (100) plane, and the dark circle in the middle indicates the additional site of the fcc (100) plane, occupied by an adsorbed atom. (b) Stacking of layers in a cooled down bcc (100) system at $T = 600K$. Symbols are the same as in (a), but the metallic region has now solidified, forming a bcc (100)/fcc (100) interface. 83

7.26 The two dimensional radial distribution functions for the bcc(100), top panel, and bcc(110), lower panel, at $T = 1000K$. 84

7.27 The interface region at a bcc (110) substrate at $T = 1000K$. The left panel is a side view and the right panel is a top view. The darkest atoms represent the solid, medium and light gray, the liquid. Different sizes indicate layer position as can be seen from the left panel. In contrast to the (100) system, there is no adsorption of the liquid to the solid substrate. 85

8.1 The disorder parameter κ for all interfaces studied. The dashed lines are meant to guide the eye, and are not related to the data obtained from the simulations. 87

8.2 The disorder parameter κ for closed packed fcc substrates (fcc(111) and fcc(100)) and for the bcc substrates. The only difference between the current figure and Figure 8.1 is the scale allowing a closer inspection of the differences between the different systems. 94

8.3 The de-wetting of the high density bcc(100) substrate. A snapshot from the simulation on system bcc(100)-C after 20000 MD steps. . . . 96

8.4 The variation of the local layer bulk density at T=1000K. Liquid layers have indices larger than zero. 99

8.5 The variation of the in-plane density at T=1000K. Liquid layers have indices larger than zero. 100

8.6 The mean number of particles per layer at T=1000K. Liquid layers have indices larger than zero. 102

A.1 Atom/solid surface interaction energy according to the Lennard-Jones 10-4 potential, two cases are shown, (1) for one crystallographic plane, and (2) for two crystallographic planes. 110

A.2 The equilibrium density profile for a non-interacting fluid adjacent to a soft 10-4 LJ wall. Note that the peaks is higher for a two layered substrate, indicating a stronger interaction with the substrate. 111

B.1 An example of a cva2 and xyz files used for storing coordinates, velocities and accelerations. In this case solid atoms are denoted by 'S', and liquid atoms by 'L'. 113

B.2 An example of the input control file of the simulation program. 115

List of Tables

4.1 Some experimental and calculated values with the Ercolessi-Adams glue potential for equilibrium lattice spacing, cohesive energy, bulk modulus, vacancy formation and migration energies, surface energy (Estimates for an "average" orientation), melting temperature, latent heat and volume change on melting. All the energies are at $T = 0$ and include relaxation effects. (Data from reference [5]). 32

5.1 A list of the different solid substrate symmetries, and planar orientations used. The d-spacing and in-plane densities $\eta = \rho d$, where ρ is the bulk density, of each substrate is indicated for a value of the lattice parameter $a_0 = 4.1$Å. 41

5.2 The required number of atomic planes of the solid for each symmetry and orientation, calculated with a lattice parameter of $a = 4.1$ Å. . . 43

5.3 Parameters for the systems with an interface parallel to an fcc (100) plane. System notation is described in section 5.4. 45

5.4 Like table 5.3 for the systems with an interface parallel to an fcc(111) plane. 45

5.5 Like table 5.3 for the systems with an interface parallel to an fcc(110) plane. 46

5.6 Parameters for the systems with an interface parallel to a bcc (100) plane. The lattice parameter for systems A and B is 4.1Å, while that for system C is 2.05Å. 48

5.7 Like table 5.6 for the systems with an interface parallel to a bcc(110) plane. 48

8.1 Calculation of the quantity $\alpha = \kappa d^2$ for various systems used in this study. 92

8.2 Ideal surface formation energies for aluminium as calculated with the Ercolessi-Adams potential [7] and with LDA [8]. 92

Abstract

Despite its great relevance to technology and materials science, the understanding of the structure at solid-liquid interfaces is still incomplete. To be more specific, the correlation between the structure of the solid, and the structure in the liquid near the interface, have not been fully addressed. Historically, this stems from the technical difficulties faced in experimentally accessing the structure of a buried interface bound by two dense phases. Atomistic simulation techniques offer, in principle, direct access to the local structure at the interface. The main limitation of atomistic models is the complexity of the bonding interactions at the interface. However, simplified models can be used to obtain basic qualitative insight, and enhance the understanding of the structure and ordering at solid-liquid interfaces.

In this study a model system for a solid-liquid metal heterophase interface is introduced. The solid is assumed to be composed of static atoms positioned in an ideal lattice with either face-centered cubic (fcc) or body-centered cubic (bcc) symmetry, and the liquid is introduced adjacent to several different terminating planes of the solid. The atomistic molecular dynamics simulation technique was applied to study structural correlations between the liquid metal and the solid on systems of up to 4500 atoms. The density profile at the interface was investigated for the different substrate crystallographic orientations and temperatures ranging from 970K to 2000K. An exponential decay of the density profile within the metal was observed,

$\rho(z) \sim \exp\{-\kappa z\}$, where z is the distance normal to the interface, leading to the definition of κ as a quantitative measure of *disorder* normal to the solid-liquid interface.

A strong correlation between the amount of ordering, as given by the inverse of κ, in the liquid phase and the underlying substrate orientation was found, the ordering being largest at the close packed fcc solid surfaces. It was also found that κ depends on the strength of the solid-liquid interaction; when the bonding is stronger, the layering is more pronounced, leading to a smaller κ. The in-plane ordering within each of the layers was studied via the two dimensional radial distribution function (2D RDF). It was found that for the fcc substrates the in-plane structure within the layers resembles that of the underlying solid structure. For the bcc substrates, two distinct phenomenon were obtained: (i) for the bcc (100) open surface, atoms from the liquid metal adsorbed into the vacant face centered sites, transforming the structure of the terminating plane of the bcc solid into fcc (100) like, and (ii) in the case of the bcc (110) substrate, no in-plane ordering was observed in any of the layers, although substantial oscillations in the density profile indicate large ordering normal to the interface.

It was also concluded that the ordering of the liquid at the interface is governed by both the structure of the solid substrate, and the properties of the liquid metal. If good wetting exists between the solid and the liquid materials, the liquid will form layers adjacent to the interface. If, in addition, the structure of the solid and the structure of the metal upon freezing are compatible, substantial in-plane ordering within the liquid layers occurs, whereas in the case of incompatible materials the layers will be liquid like.

List of symbols and abbreviations

d	the spacing between two consecutive planes in a lattice or in liquid layers
ρ	bulk density in atoms/Å^3
η	surface density in atoms/Å^2
T	temperature in [K]
a_0	lattice parameter at T=0 K
a_{lat}	lattice parameter
x, y, and z	the three Cartesian coordinates
R_c	the cutoff radius of the inter-atomic potential
Δz	slice width
$\vec{\mathbf{K}}$	reciprocal lattice vector
$S(\vec{\mathbf{K}})$	static structure factor
$g(r)$	the radial distribution factor
$\rho(z)$	density profile
u_p	potential energy per particle in eV
k_p	kinetic energy per particle in eV
L	layer index
k_B	Boltzmann factor
L_x, L_y	lateral dimensions of the simulation cell
L_z	the dimension of the simulation cell normal to the interface plane

Chapter 1

Introduction

Solid-liquid interfaces are important in many processes of technological interest, such as solidification, wetting, liquid phase joining, crystal growth, and lubrication. As a consequence, the study of solid-liquid interfaces has received a substantial amount of attention [3,9–14]. The characterization of an interface, and in particular that between a solid and a liquid is a very challenging problem. From the experimental point of view, the information about the solid-liquid interface, being confined between two dense phases, is very difficult to obtain by most experimental methods known to date. Experiments therefore rely on indirect measurements of the interfacial properties. From the theoretical point of view, two very different phases, the ordered solid and the homogeneous liquid, have to be treated within the same theoretical framework.

Solid-liquid interfaces can be cast into two categories. In the first, both the solid and the liquid belong to the same substance as in a crystal melt interface. Here the main problem is to account for structural changes between the solid and the liquid phases. In the second, the solid and liquid belong to different substances. Here the different thermodynamical behavior and chemical composition further complicates the study of the interfaces. For this reason, most analytical studies of solid-liquid interfaces deals with crystal melt interfaces, see for example [1, 15–17]. The general

question about what happens at the solid-liquid interface between two *dissimilar* materials did not gain much attention, despite its great relevance to technology.

Atomistic computational methods offer in principle a relatively straightforward way to model both phases and the interface between them, provided that a suitable representation of the inter-atomic interaction potentials, along with sufficient computer power, are available. Nevertheless, most atomistic studies available deal with models of either a hard-sphere or Lennard-Jones solid in contact with its melt. Both cases do not represent realistic materials in a satisfactory manner, such as in the case of an interface between two different metals (e.g. copper/aluminium) or between a metal and a ceramic (e.g. Al/Al_2O_3). Only recently, have there been a few studies on systems with completely different materials, one done in the context of reactive wetting between a metal and a ceramic [18, 19], and one study on the interface between solid copper and liquid aluminium [20].

In the case of crystal melt systems, both computational and phenomenological (analytic) theoretical studies, have shown that ordering occurs in the first layers of the liquid adjacent to the crystal surface. The same result emerges from experimental studies of solid-liquid interfaces [3, 14, 21]. A particularly interesting study with respect to the current project was conducted by Huisman *et al* [3], who investigated the interface structure of liquid gallium in contact with a diamond (111) surface. They observed pronounced layering of the liquid metal density profile which decays exponentially with increasing distance from the diamond. Moreover, the interlayer spacing was equal to the repeat distance of (100) planes of upright gallium dimers in solid α-Ga. Thus it appears that the liquid near the surface assumes a solid-like structure similar to the α-phase.

There is no clear understanding of the way the substrate structure may affect the structure of the liquid. To be more specific, the correlation between the structure of

the solid and the structure in the liquid at the interface have not been fully addressed. This study introduces a model system for a solid-liquid metal heterophase interface. The atomistic molecular dynamics simulation technique is applied to study structural correlations between the liquid metal and the solid. In this model the solid is assumed to be composed of static atoms positioned in an ideal lattice with either face centered cubic (fcc) or body centered cubic (bcc) symmetry. The liquid will be positioned adjacent to several different planes of the solid. The correlations in terms of the structure between the solid and the liquid will be investigated. Although this system does not represent a realistic solid material, it does serve as a *generic* model for the study of a solid-liquid interface. The main motivation for choosing such a model stems from the fact that to date, there are no suitable potentials for the simulation of a realistic solid-liquid interface, which can offer the speed required for a large scale, long time simulation of a solid-liquid interface.

The structure of this thesis is as follows. In chapter 2, a literature review of the current state of research of solid-liquid interfaces is given. This will be followed by a chapter describing the objectives of this research in detail. In chapter 4 the atomistic MD simulation method is explained. The description of the simulation systems used in this study will be given in chapter 5, followed by a chapter describing the simulations and the sampling of properties. The results of this research will be presented in chapter 7. In chapter 8 the results are discussed. Finally chapter 9 concludes and summarizes the results of this research.

Chapter 2

The liquid-solid interface structure

The information regarding the structure of the liquid and how it may be affected by the structure of the solid is hard to obtain from experimental methods. Historically, most studies of the solid-liquid interface are based on either numerical or phenomenological considerations. From the theoretical point of view, two approaches are possible; phenomenological (analytical) and computational (atomistic). In both approaches it is not enough to come up with a description of each of the two bulk phases, the theory has to include the possibility of having a phase in transition, i.e. a phase which is neither solid nor liquid, but somewhere in between. Indeed, the earliest theoretical models developed for the study of an interface between a solid and its melt [22, 23] simply ignored the third component by assuming that the liquid is unaltered up to the dividing surface.

The first attempt to take the liquid structure at the interface into account was made by Turnbull [24]. In this model the liquid was allowed to adjust itself to the crystal surface; the resulting decrease in the entropy of the system is used to estimate the surface tension. However this argument was based on the assumption that the liquid is no more than a random packing of the atoms, and therefore did not use

any structural information. Spaepen [25] extended this idea to include some structural information on the liquid. Here the main assumption was that inside the liquid the principle governing the structure is that of "maximum short range density". He essentially used a stick and ball model to construct by hand, random packing configurations of atoms near a closed packed surface of a crystal, under the constraint imposed by the above statement. The density of the liquid was calculated by enumerating the number of polyhedra produced by the packing. The most interesting feature of this model is the discovery of a decaying layered structure for the density profile of the liquid at the interface. It was also found that the number of atoms in the first interfacial layer was $\frac{3}{4}$ that in the crystal plane, i.e. the first layer has a packing fraction of at least 75% of the bulk crystal. Furthermore, the density in each of the phases remains constant up to the dividing surface, in other words, it was found that the bulk density of the layers is constant hence there is no density deficit at the interface. However, no information could be obtained beyond the first two liquid layers, nor information regarding the structure inside the layers.

These results were further confirmed by Bonissent and Mutaftschiev [26] in another study of the same nature, i.e. in which a random close packing of hard-spheres was used to model the solid-liquid interface (although this time they used a computer to construct their model instead of ping-pong balls and plasticine). However, in this study it was found that the density of the liquid near the interface is lower than bulk liquid, i.e. there is a nonzero density deficit. This discrepancy in the density of the liquid at the immediate neighborhood of the interface was related to the fact that Bonissent's model did not take relaxation into account.

2.1 Analytical models

One of the first attempts to tackle the problem of a fluid in contact with a wall in an analytic form was made by Henderson *et al* [27]. The basic idea here was to utilize the Ornstein-Zernike equation for a binary fluid mixture in the limiting case in which one of the species becomes infinitely dilute and then infinitely large in size, in other words, one of the species is just one huge atom. In this way they obtained the direct correlation function (DCF) for a hard sphere fluid in contact to a hard structure-less wall. The DCF pointed at some ordering in the fluid near the wall, however no substantial information regarding the nature of this ordering was deduced.

This problem was later adapted by Abraham and Singh [10, 28] who treated a hard sphere fluid in contact with a soft repulsive structure less wall. They used a perturbative expansion of the grand canonical potential, assuming it is a functional of the Boltzmann factor $\exp(-\beta\phi_w)$, where $\phi_w(z)$ is the interaction potential between the fluid and the wall. The basic idea behind this type of expansion will be evident later once we consider density functional methods (DFT). They were able to obtain the density profile of the fluid near the wall, which showed that the fluid is layered with a density which decays with distance z from the wall. However, the main result of this study was to indicate that the layering of the fluid near the wall depends rather strongly on both the repulsive and attractive parts of the fluid-wall interaction, in contrast to the structure of bulk liquid, which according to Van-der Waals theory, is dominated by the repulsive inter-particle interactions.

The above work did not take into account the actual structure of the hard wall and the effect that this would have on the structure of the liquid adjacent to it. Bonissent and Abraham [1] applied the Abraham-Singh perturbation theory for the case of the interface between an fcc crystal and its melt. However, the theory still required an interaction potential between the wall and the fluid which only depends

on the normal distance between the fluid particle and the wall, z. The theory did not allow for the explicit two dimensional structure of the interface. The starting point was the same: a hard sphere fluid adjacent to a hard wall. It was assumed that the repulsive nature of $\phi_w(z)$ dictates the structure of the liquid (neglecting the point of the previous study). However, the structure of the solid substrate was accounted for to some extent explicitly, by using the approach of Steele [29], whereby the interaction between a fluid atom and the atoms comprising the wall is taken as a lateral average of the interactions with the individual atoms comprising each of the crystal planes. This results in different magnitudes for the potentials of different faces. However the form of the interaction potential was the same. Two schemes were proposed in order to obtain an effective one-dimensional interaction between the wall and the fluid. In the first, a two dimensional averaged ϕ_w for a uniform distribution of surface sites at the crystal face, and in the case of a Lennard-Jones 6-12 potential, this resulted in a 10-4 power law dependence for the fluid-wall interaction (see appendix A, equation A.16). While in the second, the same averaging was considered but with a Boltzmann weight:

$$\exp[-\beta\phi_w^B(z)] = 1/A \int_A \exp\left(-\beta\phi_w(x,y,z)\right) dxdy \tag{2.1}$$

where A is the area of the unit cell of the solid surface, $\phi_w(x,y,z)$ is the interaction potential between a fluid particle and a single wall particle, and $\phi_w^B(z)$ is the resulting Boltzmann weighted effective interaction potential (see Figure 2.1) for a presentation of these potentials for fcc (111) and (100) faces). The density profiles obtained for the fcc (111) and (100) systems showed pronounced layering (see Figure 2.2), decaying with distance. The profiles for the (111) and (100) faces were very similar. The main result here is that the weighted density approach gives results in accordance with simulation data.

Haymat and Oxtoby [2, 17, 30] introduced a molecular theory for the solid-liquid interface based on a perturbative density functional (DF) method (the basis of the

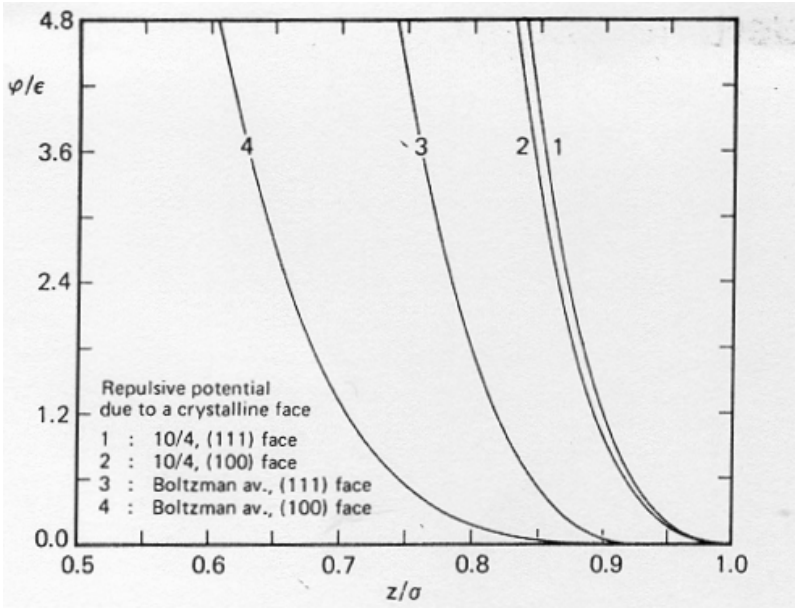


Figure 2.1: Various approximations for the wall-fluid interaction as used by Bonissent and Abraham [1]. (Figure is from reference [1]).

DF theory is explained in appendix A). In the heart of this theory is the assumption that the solid is described as a perturbation about a uniform liquid, i.e. the solid is liquid with imposed ordering. An ansatz for the density is then used to specify the crystal symmetry and the location of the freezing point of the solid. In other words this theory assumes *a-priori* a certain symmetry for the freezing liquid at the interface, hence it is only suitable to study simple crystals in equilibrium with their own melt phase. Furthermore, this is only under the assumption that the solidifying liquid falls directly into the structure of the underlying solid phase.

The density of a nonuniform solid-liquid system is then expanded in its Fourier components with wave vectors $\mathbf{K}_n$ of the solid phase reciprocal lattice. A typical form for this ansatz is [30]

$$\rho(\mathbf{r}) = \rho_0(1+\eta) + \rho_0 \sum_n \mu_n \exp(i\mathbf{K}_n \cdot \mathbf{r}) \tag{2.2}$$

where $\{\mathbf{K}_n\}$ is a set of reciprocal lattice vectors of the solid, and η is the fractional

density change on freezing. The Fourier coefficients μ_n are the "order parameters" of the theory. In practice not all the reciprocal wave vectors could be used and the expansion had to be cut off for the first set of nearest neighbor reciprocal lattice vectors. For this the application of the theory was limited in practice to crystals having open real space structure, such as a bcc lattice, which has 12 nearest neighbor reciprocal lattice vectors, allowing for fast convergence of the series. For a simple cubic system with only six reciprocal lattice vectors, the theory did not converge, and for an fcc lattice a larger set, beyond the first nearest neighbors is needed. Again these requirements posed an immense computational challenge to the perturbative approach rendering it impractical in many cases [2]. Another drawback of the DFT approach is the lack of input from the actual interaction models, only geometrical considerations are used (the reciprocal lattice vectors) and the effect of the interaction energy between the solid and the liquid, or the interaction within the liquid are not taken into account.

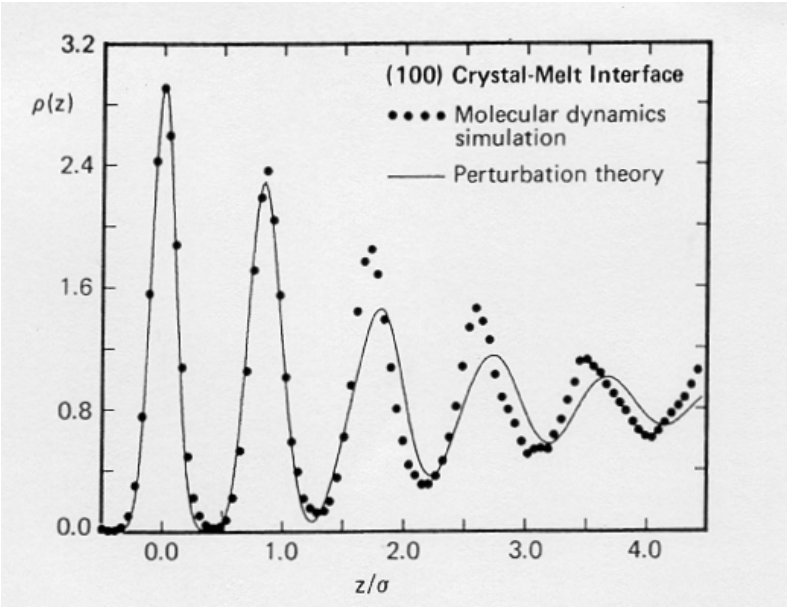
The first main result of this study was to predict different profiles for different crystal faces [30]. Based on this theory Haymat and Oxtoby performed explicit calculations of the interfacial structure of the body centered cubic (bcc) (100) and (111) crystal-melt interfaces (Figure 2.3 is a reproduction of these density profiles). These calculations suggested the existence of a structured liquid region at the interface characterized by a liquid-like averaged density, but solid like lattice ordering. The interfacial transition region was quite broad, 16-20Å in width, and comprising 10-15 layers. A comparison between the density profiles of the (100) and (111) faces showed a strong correlation between the periodicity and magnitude of the layering in the liquid and those in the the solid.

The form of decay of the density oscillations was studied by Tarazona [31] within a DFT treatment. The grand potential function was expanded in the difference

$\Delta\rho = \rho(\mathbf{r}) - \rho_l$, where $\rho(\mathbf{r})$ is the density of the fluid, and ρ_l is the bulk liquid density. This expansion is valid as long as $\Delta\rho \ll 1$, i.e very close to the freezing point. Furthermore, under the assumption that the density difference $\Delta\rho$ is slowly varying, it was found that the density oscillations decay exponentially with the normal distance from a smooth hard wall, with a decay length ξ related to the first peak of the structure factor in the bulk fluid phase. This suggests that the ordering in the liquid is inherently related to the properties of the bulk liquid, with the presence of the solid acting as a catalyst for this ordering.

Another density functional study of the solid-liquid interface is that of Curtain [15] who avoided a perturbative expansion of the free energy functional and used instead a weighted-density approximation which yields non-local, non-perturbative functional of the density $\rho(\mathbf{r})$. In addition, a density profile parameterization was introduced which takes into account the full interface width and the broadening of the density peaks. The surface free energy and density at the interface were calculated for a hard sphere (100) fcc solid-liquid interface. The interface was found to be fairly narrow (5-6 layers wide). However no information regarding the form of the decay of the oscillations in the density or the relation to the solid and bulk fluid properties was found.

(a)



(b)

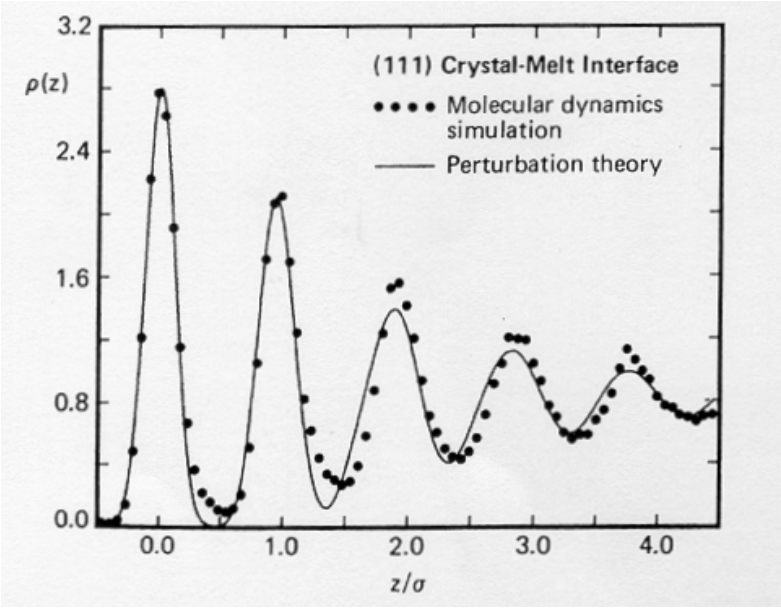


Figure 2.2: Comparing the density profiles obtained with the various approximations for the wall-fluid interaction as used by Bonissent and Abraham [1]. (Figures are from reference [1]).

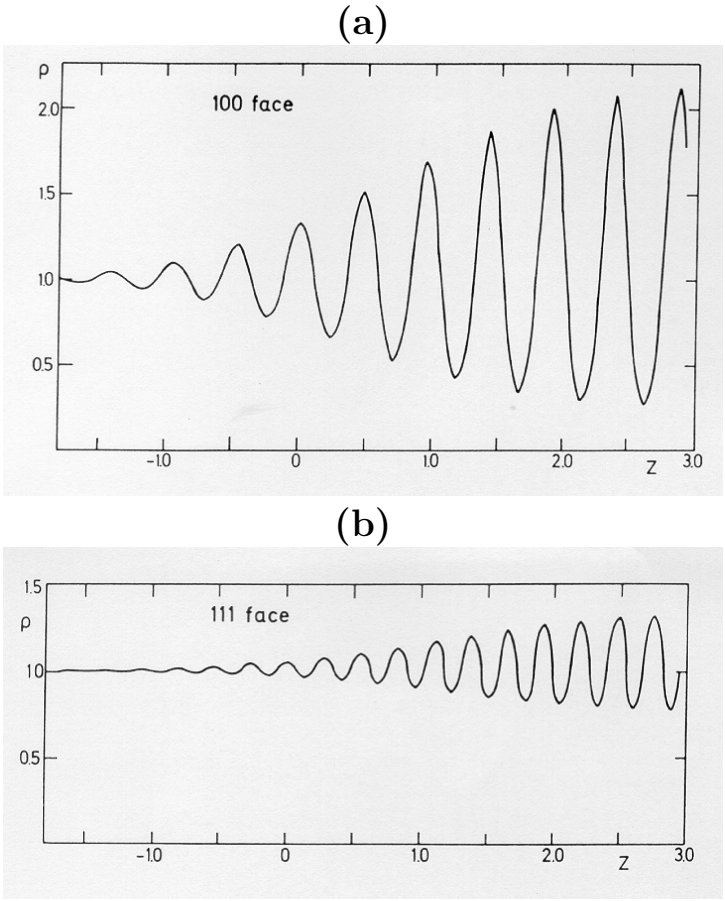


Figure 2.3: The averaged density profiles for the bcc 100 (a) and 111 (b) crystal melt interfaces as obtained by Haymat and Oxtoby. (Figures are from reference [2]).

2.2 Computational studies

Atomistic simulations, such as Molecular Dynamics (MD) or Monte Carlo (MC) are in general more flexible than analytic methods, and they offer direct access to the microscopic details of the interface. The main requirement of atomistic simulations is that appropriate interaction models are available for the solid and liquid phases, and for the different materials considered. A simulation of an interface is in general more demanding than that for a simple bulk system [32, 33]. The presence of an extended defect, such as an interface, requires that the lateral dimension of the simulation cell (parallel to the plane of the interface) to be large enough, as not to bound the interface, and therefore change its properties. This problem is usually solved by imposing periodic boundary conditions parallel to the interface (see section 5).

The main difficulty however, is related to the interatomic interactions across the interface [18, 34]. For a crystal melt interface, where both solid and liquid belong to the same material, one transferable interatomic potential is sufficient (transferable means it can be used for a liquid and for a solid state). For an interface such as in the case of a liquid-metal/solid-ceramic system, a single potential is obviously not sufficient. In this case in addition to a potential model describing the liquid (metal), and a potential model describing the ceramic, another potential model is needed for the cross interaction between the metal and the solid at the interface, an interaction that includes both short range terms, as well as long range interactions due to the ionic charges. The model has also to take into account that chemical compositions might change at the interface, further complicating the interaction model. This interaction is often not a trivial problem, and in fact to date, there are no clear ways of how to accurately model this interaction in an atomistic simulation [34]. For this reason, studies of interface systems often rely on simplified interaction models.

The most widely used interaction models for the study of the solid-liquid interfaces,

are the Lennard-Jones (LJ) interaction potential (see section 4.2 for more details of the potential), which is suitable for the description of rare gases, but is usually considered appropriate for dense simple fluids and crystals which have a fcc lattice structure. Another popular choice is the hard-sphere model, which although being a "cartoon" of real interatomic interactions, is interesting for two main reasons: First, due to its relative simplicity (either ∞ when the two particles overlap, or 0 otherwise), this system lends itself well to theoretical studies (most of the DFT studies of crystal melt interfacial structure involve hard sphere systems, see for example [15, 35]). Second, the structure and freezing dynamics of most dense simple fluids is governed, for the most part, by the repulsive part of the interaction potential.

The vast majority of atomistic studies of solid-liquid interfaces deal with the equilibrium crystal melt interface, where both solid and liquid belong to the same material, and are in thermodynamic equilibrium conditions. Studies of non-equilibrium liquid-solid interfaces usually describe the effect of the solid on the fluid by an effective external potential, the lateral structure of the interfaces is either implicitly included, or a smooth surface is assumed.

The overall picture of the interface that emerges from these studies conforms with that from phenomenological studies, namely the density profiles show an oscillating structure, gradually decaying into a straight line representative of the density of a uniform liquid phase. The main advantage of the atomistic studies is the ability to reveal a microscopic picture of the ordering at the solid-liquid interface, and the study of the influence of the solid orientation and symmetry on the ordering.

For an fcc LJ crystal in equilibrium with its melt, the interface was investigated in detail by Broughton *et al* [36–41], and more recently by Huitema *et al* [42], for the (111), (100) and the (110) face. Measurements of the lateral diffusion coefficients, radial distribution functions, and nearest neighbor fractions, all indicate an interfacial

width of about 3 atomic diameters, irrespective of the crystal orientation. The density profiles, on the other hand, showed a different behavior in that the (111) and (100) orientations have large density oscillations far beyond the range of 3 atomic diameters, whereas the (110) orientation does not show these oscillations. The interlayer spacing, i.e. the distance between consecutive minima (or maxima, either which give the same result) in the oscillations, for the (110) and (111) orientations, was found to be equal to the d-spacing of the underlying solid phase planes. For the (100) orientation on the other hand, the interlayer spacing in the liquid shows a shift toward that of a (111) plane. Broughton *et al* explained this based on the low energy cost for the creation of density oscillations in the liquid with a periodicity equal to the d-spacing of a (111) planes: variations in the liquid that are induced by the presence of the interface will be enhanced if the wavelength is consistent with the position of the first maxima in the liquid structure factor $S(q)$, which for a LJ fluid occurs approximately at a wavelength of $q_0 = 2\pi/d_{111}$. The inter-planar distance of the (100) orientation is close enough to $d_{111} = a/\sqrt{3}$ (a is the lattice parameter), to make the change from $d_{100} = a/2$ to d_{111} in the melt more favorable than the switch from $d_{110} = a/\sqrt{8}$ to d_{111} (see section 5.1).

Rull and Toxvaerd [43] studied the structure at a solid-fluid interface by performing MD simulations for a LJ fluid up to static fcc(100), fcc(111) and bcc(100) surfaces. The LJ was split into purely short ranged repulsive and purely long range attractive terms. The calculations were carried out with an interaction composed of a pure repulsive term and an attractive term scaled by a positive factor λ. The authors argue that in this way the system asymptotically approaches the triple point of a LJ systems as λ approaches unity. It was found in this study that the in-plane ordering in the interface is insensitive to λ, and therefore is mainly determined by the repulsive part of the interaction. The layering in the quasi-liquid, on the other hand, increased

with λ. It was also observed that the density oscillations decays exponentially with distance from the interface approximately as $\exp[-\alpha z/\Delta z^2]$ where Δz is the mean interlayer spacing, and α is a parameter, which Rull *et. al.* argue, based on the simulations data, that it is a universal value for all interfaces. However, this study suffers from two main drawbacks. The first is related to the very small system sizes used, 840 particles in total, and second is related to the small equilibration times used (5000 MD steps). This resulted in large errors, therefore the assumption of universality of α is not entirely justified based only on the results of these simulations. The interlayer spacing in the quasi-liquid for all systems considered in this study was reported to be compatible with the underlying substrate. In other words no expansion in the interlayer spacing in the (100) direction was observed, in contrary to other studies on systems with similar nature [36–42, 44, 45]. This might be attributed to the very short times of the simulations.

The hard-sphere crystal-melt system was the focus of a number of recent studies. Kyrlidis and Brown [35] have performed Monte Carlo simulations for the (111), (110), and (100) interfaces. The number of particles was about 3000 for each interface studied. The results for the density profiles indicate interfacial widths of about 4-5σ, 5-6σ, and 4σ (where σ is an atomic unit of spacing) for the (100), (111), and (110) interfaces, respectively. Similar to the studies of a Lennard-Jones (LJ) system, an increase in the density profile spacing of the (100) interface, as one moves normal to the interface from solid to fluid, toward a value very close to the interlayer spacing at a (111) interface was observed, suggesting the possibility of a (111)-like ordering in the liquid layers.

In another study, Mori *et al.* [46] used a constant energy molecular dynamics simulation to study the same hard-sphere interface, also with about the same number of particles. Here the estimated widths of the interfaces was about 5.5σ for all interfaces

(these widths are estimated directly from the density profile graphs in [46], and were not given explicitly by the authors). However, the simulation runs of this study were extremely short (of the order of 700 collisions per particle, including equilibration), the density profiles were therefore subject to significant statistical errors, especially in the liquid phase, where oscillations are seen to extend through out the bulk-fluid region. These are probably undissipated remnants of the ordered fcc structure from which the fluid phase is produced.

A molecular-dynamics study on of larger system, comprising approximately 10^4 hard spheres, and for longer times (a few thousand collisions per particle) was performed by Davidchack and Laird [44, 45] on the (100) and (111) fcc interfaces. In agreement with previous hard-sphere and LJ studies, an increase in the (100) interlayer spacing was observed. Analysis of the orientational in-plane order perpendicular to the interface indicated that this phenomenon is not due to (111)-like ordering of the liquid near the (100) interface, as has been speculated by Kyrlidis [35]. This supports the suggestion of Broughton *et al.* [40], that is, this transition is actually correlated with the properties of the bulk liquid. Analysis of the in-plane ordering further suggests that the transition from solid to fluid occurs within 2-3 atomic layers, which is much narrower than the width indicated by the density profiles of about 7-8 layers.

The equilibrium crystal melt interface involving a more realistic potential was conducted by Chen *et al.* [12]. An embedded atom method potential for nickel was employed in a molecular dynamics simulation of about 1200 particles, using both constant energy and constant temperature runs on (111) and (100) interfaces, for times of up to about 50 pico seconds. The same phenomenon of the increase of the interlayer spacing of the (100) face toward that of the (111) face was observed as well. In addition, it was observed that the averaged density within each layer for both the

(100) and (111) systems is 0.086 Å^{-3} in the solid and 0.081 Å^{-3} in the liquid. Correlating this result with the expansion of the (100) interlayer spacing, suggests that for the (111) face the density decrease associated with the transition from crystal to fluid is achieved by maintaining the spacing between the layers along the direction normal to the interface, while the number of particles in layers decreases. In contrast, for the (100) face, the crystal-liquid transition is achieved by a gradual increase of the interlayer spacing, while the number of particles remains constant. It was also observed in this study that the (100) interface was more diffuse than that at a (111) interface.

A heterophase interface between liquid aluminium and solid copper has been recently studied by Geyserman and Gorse [20]. The system studied is made of crystalline copper with surface terminations (100) and (111) and liquid aluminium, both modeled with an *n*-body semi-empirical potential, and the cross interactions were obtained by fitting the mixing enthalpy of the two species to experimental values. It should be noted that the dissolution process between the two phases is very slow at the time scale of the molecular dynamics simulations, while the structure of the liquid reaches equilibrium on a much faster time scale. The simulations were conducted at a temperature of $T = 1000$ K, which is below the melting point of copper, and above that of aluminium. It was found that at the interface the liquid aluminium forms layers with a fixed interlayer spacing of a constant value of 2.4 ± 0.1 Å, no matter what the orientation of the crystal was. Therefore no obvious correlations exists between the interplanar distance in the crystal, which for a (100) is $d_{100} = 1.84$ Å, and for (111) is $d_{111} = 2.13$ Å, and the layering in the liquid. The local bulk density in the layers of the liquid was found to be typical of bulk liquid aluminium. It was also found that the first aluminium layer possessed crystalline order, the aluminium atoms were epitaxially stacked above adatoms sites of the copper terminating planes,

although there is a substantial misfit between the atomic diameters in copper and aluminium. This misfit is accommodated by the creation of well localized vacancies trapped at the interface region, which help relaxing the misfit strain. Geyserman and Gorse [20] also considered the interfaces between a single crystalline or amorphous copper layer, and a bulk amorphous static copper substrates. In the models with a single layer, both crystalline or amorphous, it was found that the density oscillations get narrower and better separated. However, when the liquid aluminium is adjacent to a bulk amorphous static copper substrate, the layering in the liquid aluminium was greatly suppressed.

2.3 Experimental studies

Howe [21] has avoided the direct study of a solid-liquid interface by studying a crystal amorphous interface produced by partially crystallizing a $Pd_{80}Si_{20}$ amorphous sample. The interface structure in the resulting specimens are then studied using high resolution electron microscopy. Assuming that the structure of the liquid and the amorphous solid are similar, Howe showed that density oscillations exist at the crystal-amorphous interface. This experimental study was the first to strongly suggests the layering of liquid at the solid liquid interface.

More direct evidence of layering of the liquid at the solid-liquid interface has been given by Huisman *et. al.* [3, 4] in an *in situ* X-ray diffraction study of the interface created by deposition a liquid gallium drop on top of a diamond (111) substrate. The choice of materials in this study is crucial. The Ga metal has a relatively low melting temperature, $T = 300$ K, and diamond is essentially transparent to X-rays at the photon energy used (17 keV), enabling the incident beam to reach the interface from the diamond side. The diamond (111) substrate, having a high surface free energy, is effectively a hard wall, which at 300 K is nonreactive toward Ga. The analysis of

the scattering data indicate pronounced layering of the liquid metal density profile which decays exponentially with increasing distance from the diamond, see Figure 2.4. Moreover, the interlayer spacing was equal to the repeat distance of (100) planes of upright gallium dimers in solid α-Ga. Thus it appears that the liquid near the surface assumes a solid-like structure similar to the α-phase of gallium, which is nucleated on freezing at lower temperatures.

Reichart *et. al.* [14] studied the lateral structure of liquid lead in contact with a Si (001) substrate by means of a well defined synchrotron high energy X-ray microbeam (E=71.5 keV, 8 μm size). The beam penetrates the interface from the solid side, at incident angles below the critical angle of total internal reflection the incident beam is totally reflected. Non-propagating (evanescent) X-rays are created within the liquid adjacent to the solid wall while the X-ray field is absent in the bulk liquid. The in-plane scattering from the evanescent X-ray field gives direct access to the local liquid structure in contact with the wall. It was found that the bulk structure of liquid Pb is preserved to a large extent at the interface region. The most interesting discovery of this study is the existence of non-crystalline modulations of the liquid scattering intensity, suggesting the existence of five-fold and four-fold symmetry building blocks within the liquid. The four-fold symmetry is imposed by the interaction between the Si(001) substrate and the liquid atoms, while the five-fold symmetry is due to the bulk liquid interactions. This is the first evidence of a five-fold symmetry building blocks in a liquid, which has important implications in understanding under-cooling and solid-liquid transitions (see [47]).

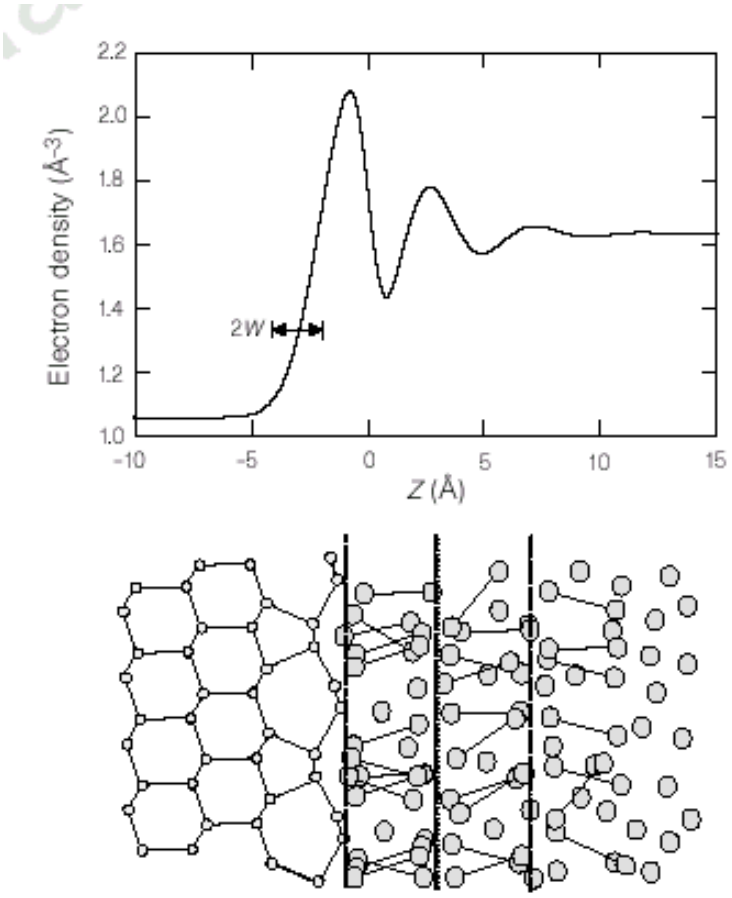


Figure 2.4: Experimental evidence of liquid layering at a solid-liquid interface by Huisman *et. al.* [3, 4]. The top panel shows the averaged electron density profile in liquid Ga normal to the solid interface, and the lower panel shows a suggested structure for the interface. Large symbols represent Ga atoms, bonds between Ga atoms represent dimers. (Figure is from reference [3]).

Chapter 3

The objectives of the research

The two main questions that are addressed in this research are the following:

1. How the layering in the liquid at the interface is affected by the structure of the underlying solid substrate?

2. What kind of ordering, if any, exists within the layers?

In most studies of solid-liquid interfaces, these questions were addressed in the context of coexisting crystal-melt conditions. It is found in general that the layering in the liquid is that typical of freezing conditions. In other words, the liquid in the first few layers is ordered in a structure dictated by the solid, and the interlayer spacing at the interface in the liquid is equal or very close to the d-spacing of the solid planes parallel to the interface. It is also found in most cases that the component of ordering normal to the interface, as indicated by the density profiles, extends more into the liquid than that indicated by the in-plane ordering. What kind of ordering will exist far from these conditions, specifically at a heterophase interface between to incompatible materials, and what will be the effect of temperature and solid-liquid interaction on this ordering are open questions.

A full simulation of a general heterophase solid-liquid interface between two different materials, such as between a ceramic and a metal, requires three different interaction potentials to be available, describing the interaction between the liquid atoms, between the solid atoms, and the cross interaction between the solid and liquid atoms. While fairly accurate potentials exist for the description of bulk interactions, to date, no simple potentials exist to describe the cross interaction, making large scale atomistic simulations of these interfaces an impractical option [34], (although these limitations are expected to fade away in time). Nevertheless, simplified models can offer an insight into the basic physics of the problem. Especially in the context of the metal-ceramic interface, where the ceramic is usually much "tougher" than the metal, and has a melting temperature of approximately double that of the metal. Hence, it is reasonable to assume that a model system in which the solid is composed of a set of atoms pinned to their lattice positions and the liquid atoms are free to move, will serve as a model for a liquid-metal/solid-ceramic system.

In this research a model solid/liquid-metal interface is proposed which allows a controlled study of structural correlations between the solid and the liquid metal at the solid-liquid interface. The solid substrate in this model system is composed of a simple rigid structure with a defined symmetry, either fcc or bcc. The liquid, on the other hand, is assumed to be a metal substance, aluminium. In the current research, atomistic simulations have been conducted to study the density profile and structure of the liquid-metal/hard-wall interface as a function of temperature and substrate structure. In particular we show that the decay of the density profile is quantitatively and qualitatively related to the underlying structure of the substrate, as well to the bulk properties of the liquid.

Chapter 4

Methodology

The Molecular Dynamics (MD) simulation technique will be used as the primary tool in this study. Therefore this section begins with a brief review of the method, followed by a description of the interaction potential, and finally a general description of the boundary conditions.

4.1 The molecular-dynamics method

In a MD simulation the time evolution of a system of atoms is followed by integrating the classical equations of motion,

$$\mathbf{F}_i = m_i \mathbf{a}_i \tag{4.1}$$

for each atom i in the system, where F is the force, m_i the mass of each atom, and a is the acceleration. The MD simulation is a deterministic technique: given an initial set of positions and velocities, the subsequent time evolution is, in principle, completely determined. In practice this is limited to available numerical accuracy of integrating algorithms and the computer's roundoff error. The main ingredient in an MD simulation, just like in a Monte Carlo (MC) simulation, is the potential

function describing the interaction between the atoms, $\mathcal{U}(r^N)$, from which the forces are derived as the gradients with respect to atomic displacements,

$$\mathbf{F}_i = -\nabla_{r_i}\mathcal{U}(r^N). \tag{4.2}$$

The integration algorithms are based on finite difference methods, where time is discretized on a finite grid, with the time step Δt, being the distance between consecutive points on the grid. Given the velocities and positions (and perhaps accelerations, depending on the integration scheme used) of all atoms at time t, the positions and velocities are calculated at time $t + \Delta t$ via an integration scheme [32, 33, 48]. As an example, we consider the velocity-Verlet algorithm. The velocities v, positions r, and accelerations a at time $t + \Delta t$ are obtained from the same quantities at time t in the following way:

$$\begin{aligned} r(t+\Delta t) = &\quad r(t) + v(t)\Delta t + (1/2)a(t)\Delta t^2 \\ v(t+\Delta t/2) = &\quad v(t) + (1/2)a(t)\Delta t \\ a(t+\Delta t) = &\quad -(1/m)\nabla\mathcal{U}(r(t+\Delta t)) \\ v(t+\Delta t) = &\quad v(t+\Delta t/2) + (1/2)a(t+\Delta t)\Delta t \end{aligned} \tag{4.3}$$

The simulations start by supplying an initial configuration including positions, and velocities. For the simulation of bulk liquid, the initial configuration might be that of an ideal lattice, with a lattice parameter such that the density of this "solid" is close to the liquid phase density. The initial velocities may be chosen from a random distribution around an average value corresponding to a certain temperature. This introduces some randomness into the initial conditions, but the dynamics themselves are deterministic. The same considerations of boundary conditions apply as in the case of MC simulations.

The above integration of the equations of motion preserve the energy E, number of particles N, and volume V of the system, and hence describe a trajectory of a

system in the microcanonical ensemble. The temperature is related to the average kinetic energy K via the equipartition theorem of statistical mechanics:

$$K = \frac{3}{2} N k_B T \tag{4.4}$$

hence the temperature in the sample may be controlled by altering the velocity of the particles. In order to produce a canonical trajectory in the MD simulation, Andersen [49] applied occasional stochastic "collisions" with a heat bath, i.e. a particle is chosen at random and its velocity altered. This transforms the system from one energy level to another. In between the collisions the system evolves along a constant E trajectory. Another method due to Nosé and Hoover [50] produces a trajectory in the canonical or the isobaric-isothermal ensembles. They apply certain modifications to the Lagrangian equations of motion as to preserve either the temperature or both the pressure and temperature. This method does not use any stochastic procedure and hence produces trajectories which are deterministic.

4.2 The interatomic interactions

The two body potentials, such as the Lennard-Jones potential,

$$\Phi_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \tag{4.5}$$

are useful for realistically simulating rare gas atoms, where bonding is mainly due to van der Waals forces. In and of themselves they are not accurate for other systems although they do provide excellent "toy"systems. This form of a potential, is also an important ingredient in many-body potentials. This is especially true for metals, where they account for the core-core repulsive interaction, while the bonding due to the electrons is accounted for by an additional many-body term,

$$V = \frac{1}{2} \sum_{i \neq j = 1}^{N} \Phi(r_{ij}) + \sum_{i=1}^{N} U(n_i) \tag{4.6}$$

where $\Phi(r)$ is the two-body part, and $U(n)$ is a function giving the energy of an atom as a function of "generalized coordinations" n. For a given atom, n is in turn constructed as a superposition of contributions from neighboring atoms,

$$n_i = \sum_{j \neq i} \rho(r_{ij}) \tag{4.7}$$

where $\rho(r_{ij})$ is a short-ranged, decreasing function of distance, and the sum is over all atoms. This general form is common to all many-body schemes for metals, including the *embedded atom method* (EAM) [51], the *Finnis-Sinclair potentials* [52], the *effective medium theory* [53], and the *Glue* model [5]. The main physical point behind these models is that the metallic bonds become weaker when the local environment becomes more crowded. These schemes differ in the procedures used to build the functions $\Phi(r)$, $\rho(r)$, and $U(n)$. In the EAM scheme n_i is interpreted as the local charge density at site i which is constructed by a rigid superposition of atomic charge densities ρ. The embedding energy $U(n)$ of an atom at site i is then assumed to be, to a certain approximation, the same as if it were in a uniform electron gas of that density, which can be calculated and tabulated. In the class of Finnis-Sinclair potentials, the form of $U(N)$ is chosen to mimic the tight binding result of $U(n) \sim \sqrt{n}$. In general, the forms are chosen on the basis of computational convenience and empirical fitting.

In the glue model, $\rho(r)$ is defined as a measure of an "atomic density", and $U(n)$ as a "glue function", which decreases with increasing atomic density. A glue potential for aluminium was designed by Ercolessi and Adams (EA) [5]. The functions constituting the EA potential are defined as third-order polynomials (cubic splines). These are fitted to a large set of data produced by first-principles calculations, including fitting the potential to *ab initio* atomic forces of many atomic configurations, including surfaces, clusters, liquids and crystals at finite temperature. This potential reproduces many of the properties of solid and liquid aluminium, such as shown in table 4.1.

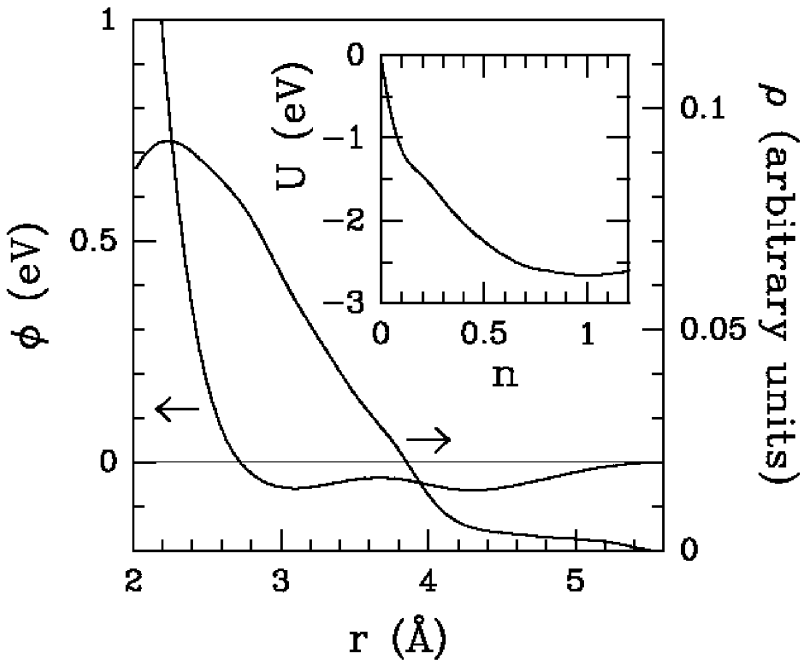


Figure 4.1: The three functions constituting the optimized glue potential for Al. (Figure is from reference [5]).

The EA potential was used in an interesting realization of wetting in an atomistic study by Di Tolla *et al* [6] to simulate the wetting behavior of liquid aluminium on top of an aluminium crystal surface near the melting point. Two distinct wetting behaviors were found depending on the orientation, as shown in Figure 4.2. For the (110) surface, at temperature of $0.99T_m$, i.e. 1% below the melting point, the drop of liquid aluminium wets completely, while for the (111) surface, which undergoes overheating beyond its melting point, the drop does not spread into the metal substrate. The drop in this simulation consisted of 861 particles, and the substrate was composed of 16 layers, with the bottom three layers kept fixed at the ideal locations.

	exp.	calc.
$a_\circ(\mathring{A})$	4.032	4.032
E_c(eV/atom)	3.36	3.36
B(MBar)	0.809	0.809
$E^f_{\rm vac}$(eV)	0.66	0.69
$E^m_{\rm vac}$(eV)	0.62	0.61
$E_{\rm SF}$ (meV/$\mathring{A}^2$)	7.5–9.0	6.5
T_m(K)	933.6	939 ± 3
L_m(eV/atom)	0.1085	0.1054 ± 0.0003
$\Delta V_m(\%)$	6.5	8.4 ± 0.1

Table 4.1: Some experimental and calculated values with the Ercolessi-Adams glue potential for equilibrium lattice spacing, cohesive energy, bulk modulus, vacancy formation and migration energies, surface energy (Estimates for an "average" orientation), melting temperature, latent heat and volume change on melting. All the energies are at $T = 0$ and include relaxation effects. (Data from reference [5]).

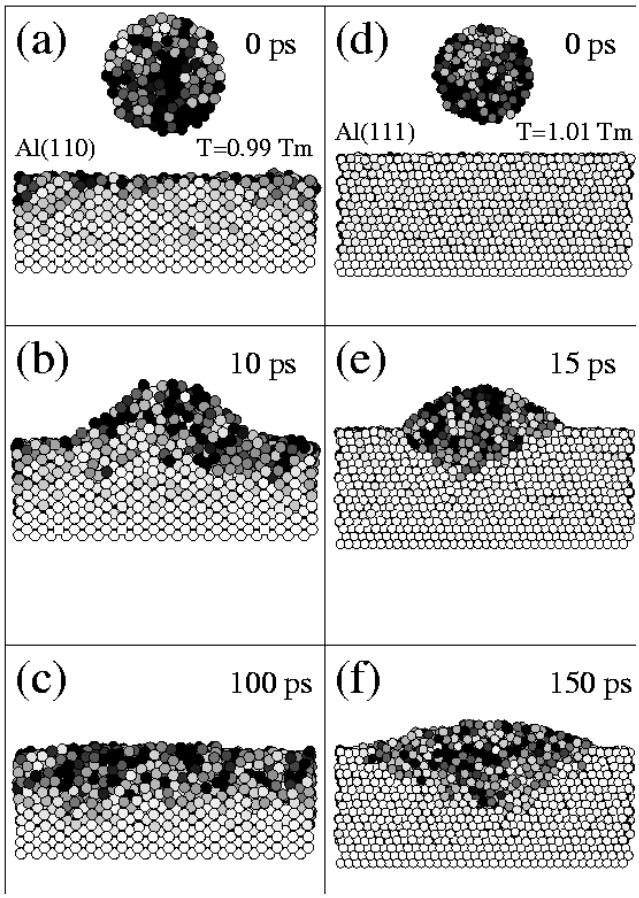


Figure 4.2: Evolution of an aluminium liquid drop on a substrate of the same material. The drop on the (110) surface spreads completely into the substrate, while that on the (111) surface does not. (Figure is from reference [6]).

4.3 Boundary conditions

There are several ways to define the boundary conditions of the system. The simplest way to deal with the boundaries of the system is to leave them free and this is the so called *free boundary conditions* (FBC). In this case atoms at the boundary will have less neighbors to interact with. These conditions are suitable for simulating free surfaces. For simulating a bulk system, *periodic boundary conditions* (PBC) are applied. In this case particles are usually enclosed in a cubic box of a fixed side-length L. The box is then "replicated" to infinity by rigid translations in the three Cartesian directions, producing infinite "images" of each particle in the box. In this case, a particle in the box would not only interact with the rest of the particles in the same box, but with all the images in the nearby boxes, provided that they lie within the range of interaction. Periodic boundary conditions can be combined with free boundary conditions to simulate free surfaces. In this case the PBC are applied, say in the x and y directions, while in the z direction (perpendicular to the interface) they are free. Fixed boundary conditions (FBC) may also be applied, especially to simulate the effect of rigid walls.

Chapter 5

Description of the simulation systems

The aim is to model a system composed of an interface between two different materials, in which one of the components is solid and the other is liquid. In this study the solid material is modeled as a configuration of atoms, pinned to their lattice sites. The liquid atoms are allowed to move according to the equations of motion dictated by the interatomic potential between them and with the solid substrate. Since an interface is in general a spatially extended feature of the system, it may contain a substantial amount of particles, and extend over many unit cells. A description that allows this feature is then needed. The most straightforward way to build an interface is to adjunct the liquid to the solid substrate and to apply periodic boundary conditions parallel to the interface plane, say in the x-y directions (see sketch in Figure 5.1). In this way the interface is not confined in principle to the area of the original simulation cell.

Boundary conditions perpendicular to the interface (in the z direction), are expected to simulate the bulk media on either side of the interface. On the rigid (substrate) side where atoms are fixed to crystalline positions, it is sufficient to require

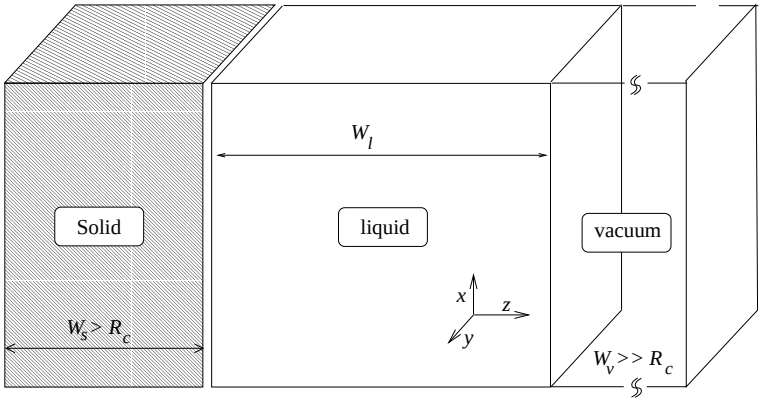


Figure 5.1: Sketch of the model system, and the width of the solid, W_s, should be larger than the cutoff distance of the potential, $W_s > R_c$. The extent of the liquid region perpendicular to the interface plane, W_l, should be large enough that the free surface does not interact with the solid at the other end of the simulation cell.

that the extent of the solid region in the z direction, W_s, to be larger than the cutoff of the interaction potential: $W_s > R_c$. In this way liquid atoms near the interface do not "see" the end of the rigid layer, hence it acts like a semi infinite bulk system. On the liquid (metal) side the situation is more difficult. If periodic boundary conditions are applied, the liquid and solid atoms at both ends of the simulation cell will interact forming a second interface in the system. These two interfaces can then interact with each other unless the dimension in the z direction is sufficiently large so that the interaction is negligible.

In addition, due to the confinement of the simulation cell, the system will not be able to respond to volume changes caused by stress at the interface. One way to overcome this difficulty is to use a model in which the sample is enclosed by two moving rigid walls as proposed by Lutsko *et al.* [54], where to allow for volume deformations, the z dimension was considered as a dynamic variable and a formulation introduced to allow a flexible length of the sample normal to the interface in response to stresses in the simulation cell. In this study we take a simpler route. A liquid layer

is deposited above the solid layer, and then a vacuum region is inserted with periodic boundary conditions in all directions. In this way we have one free liquid interface and one internal solid-liquid interface. Provided that the extent of the liquid layer, W_l, is large enough in the z direction, there will be no interaction between the free liquid surface and the internal rigid-liquid interface, giving rise to interesting "interference" effects [55]. The system will also be free to respond to stresses at the interface, since there is nothing to limit the liquid from expanding in the z direction.

Since periodic boundary conditions are also applied in the z direction, there are two requirements on the extent of the vacuum region, W_v. Firstly, it should be larger than the cutoff distance of the interatomic potential, otherwise atoms at the surface of the liquid will interact with atoms of the solid at the other end of the simulation cell. Secondly, if liquid atoms evaporate from the free surface and enter into the vacuum region, there is a chance that these vapor atoms would reach the end of the vacuum and interact with the solid on the other end via the periodic boundary condition. However, as will be shown in section 6.6, there are practically no vapor atoms during the simulation time. Hence it is sufficient to have a large enough vacuum based on the cutoff distance alone. In practice we chose a vacuum region of 15 times the extent of the rigid and liquid regions in the z direction, that is $W_v = 15 \times (W_s + W_l)$.

5.1 The structure of the solid substrates

Two basic types of symmetries for the solid substrates are used: face-centered-cubic (fcc) and body-centered-cubic (bcc). For sake of completeness we present here the lattice parameters of the bcc and fcc lattices found in any elementary solid state text book. The primitive fcc unit cell shown in Figure 5.2(a) contains four atoms. A

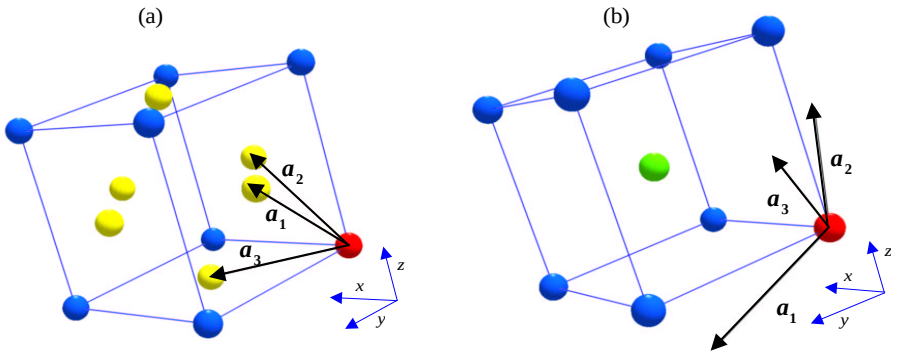


Figure 5.2: A symmetric unit cell for a fcc lattice (a), and a bcc (b). The unit vectors correspond to those in equations 5.2 and 5.3

symmetric set of primitive vectors for the fcc lattice is

$$\begin{aligned} \mathbf{a}_1 &= \tfrac{a}{2}(\hat{\mathbf{y}} + \hat{\mathbf{z}}) \\ \mathbf{a}_2 &= \tfrac{a}{2}(\hat{\mathbf{z}} + \hat{\mathbf{x}}) \\ \mathbf{a}_3 &= \tfrac{a}{2}(\hat{\mathbf{x}} + \hat{\mathbf{y}}) \end{aligned} \tag{5.1}$$

where a is the lattice parameter. The bcc primitive unit cell, see Figure 5.2(b), contains two atoms in each primitive unit cell. A symmetric set of primitive vectors for the bcc lattice is

$$\begin{aligned} \mathbf{a}_1 &= \tfrac{a}{2}(\hat{\mathbf{y}} - \hat{\mathbf{x}} + \hat{\mathbf{z}}) \\ \mathbf{a}_2 &= \tfrac{a}{2}(\hat{\mathbf{x}} - \hat{\mathbf{y}} + \hat{\mathbf{z}}) \\ \mathbf{a}_3 &= \tfrac{a}{2}(\hat{\mathbf{x}} + \hat{\mathbf{y}} - \hat{\mathbf{z}}) \end{aligned} \tag{5.2}$$

The lattice parameter in most cases is fixed to a value of $a = 4.1$ Å, which is close to the value obtained for aluminium at the melting point ($T = 940$ K) with the potential model of Ercolessi-Adams used here [5]. For the fcc substrates, three orientations for the terminating planes are used: (100), (111), and (110), see Figures 5.3, 5.4, and 5.5. For the bcc system, two orientations for the terminating substrate plane are used:

(100) and (110), see Figures 5.6, and 5.7. The main difference between these two systems is that the fcc is a native configuration of the potential, i.e. the aluminium liquid upon freezing would naturally fit into an fcc structure, while the bcc is a non-equilibrium structure for the aluminium, so it will provide additional insight into the case when the interface is between two dissimilar materials. In general then there are three closed packed surfaces: the fcc (111), (100), and bcc (110), and two open surfaces: the bcc and fcc (100) planes.

With a lattice parameter of $a_0 = 4.1$ Å, the bulk densities of each substrate are: $\rho(\text{fcc}) = 0.0580$ atoms/Å^3, and $\rho(\text{bcc}) = 0.029$ atoms/Å^3 for the fcc and bcc substrates, respectively. Table 5.1 lists the different lattice symmetries and orientations used for the solid phases, along with the inter-planar spacing d, and the in-plane density η, in each case. A note is due regarding the inter-planar spacing, normally the d-spacing of a family of planes with Miller indices (hkl) is given by: $d_{hkl} = a_{latt}/\sqrt{h^2 + k^2 + l^2}$. For example in the case of (110) planes in the fcc lattice: $d_{110} = a_{latt}/\sqrt{2}$. However, this is not the smallest distance between two consecutive planes having the same in-plane structure as the (110) plane. The family of planes that includes all planes with a (110) structure is in fact the (220) family. Hence in calculating the d-spacing for the (110) surface, we used d_{220}. Similarly for the (100) family, we used d_{200}.

5.2 The width of the solid substrates

The width of the solid substrate (the rigid layers) should be larger than the total interaction range of the EA potential. The range of effective interactions in an EAM type of potential is actually twice the cutoff radius of the pair potential part, which for the EA potential is 5.558Å, so the total interaction range is $R_c = 11.116$ Å. For example, with a lattice constant of $a = 4.1$Å, and a fcc (220) inter-planar distance (or

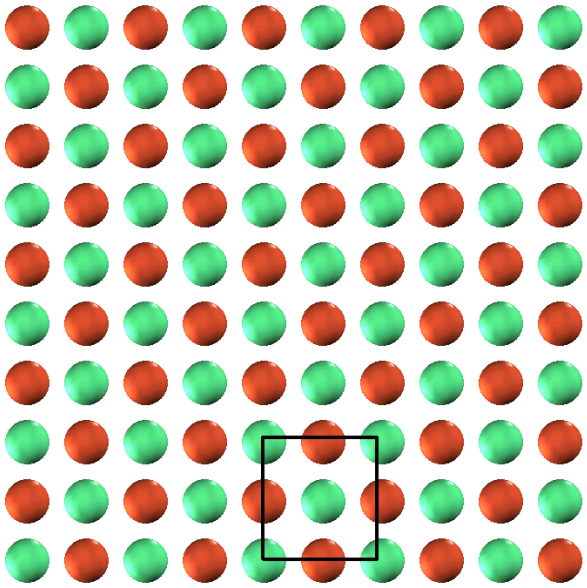


Figure 5.3: Top view of the (100) fcc substrate. The last two planes are shown, green is the top most, and red is the layer beneath.

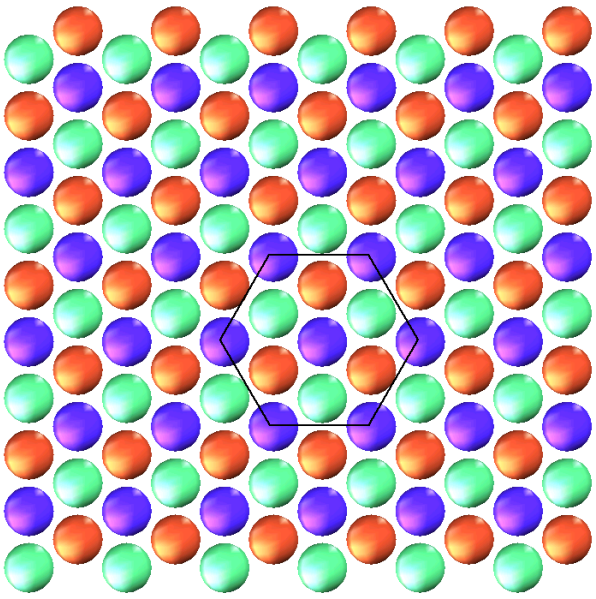


Figure 5.4: Top view of the (111) fcc substrate, the three top layers are shown, blue is the top most followed by red and green.

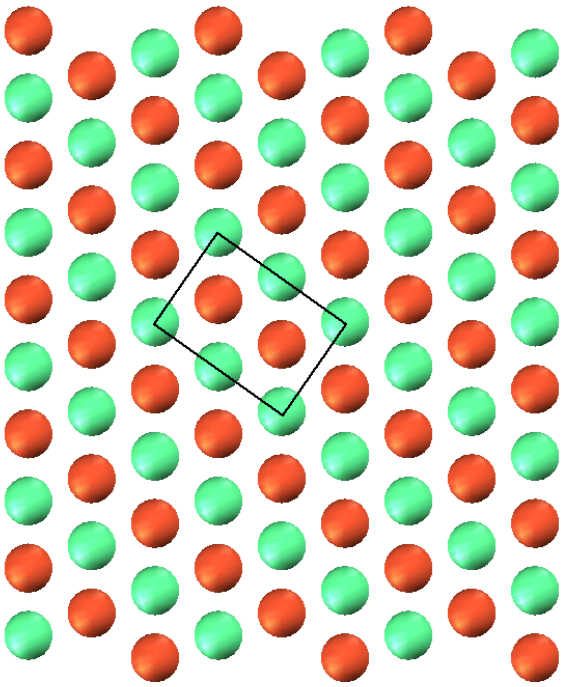


Figure 5.5: Top view of the (110) fcc substrate showing the two terminating planes, (green on top, followed by red).

symmetry	face	d		d Å	η atom/Å^2
fcc	(100)	$a/2$	=	2.05	0.1189768
	(110)	$a/\sqrt{8}$	=	1.4495689	0.0841293
	(111)	$a/\sqrt{3}$	=	2.3671361	0.1373826
bcc	(100)	$a/2$	=	2.05	0.0594884
	(110)	$a/\sqrt{2}$	=	2.8991378	0.0841

Table 5.1: A list of the different solid substrate symmetries, and planar orientations used. The d-spacing and in-plane densities $\eta = \rho d$, where ρ is the bulk density, of each substrate is indicated for a value of the lattice parameter $a_0 = 4.1$Å.

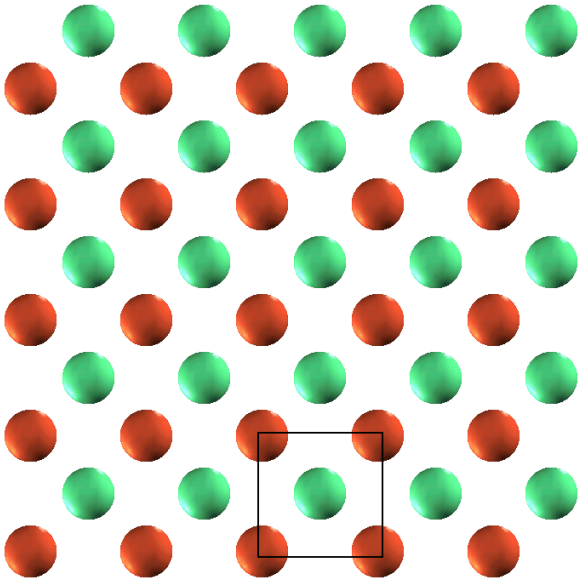


Figure 5.6: Top view of the bcc (100) system. Two terminating planes are shown in red and green.

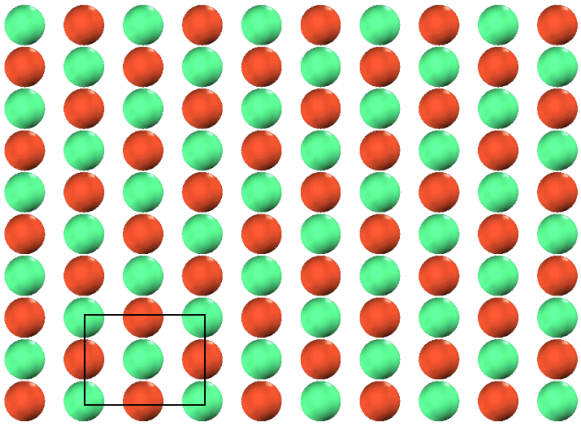


Figure 5.7: Top view of the bcc (110) system. Two terminating planes are shown in red and green.

symmetry	orientation	No. of layers
fcc	(100)	6
	(110)	8
	(111)	5
bcc	(100)	6
	(110)	4

Table 5.2: The required number of atomic planes of the solid for each symmetry and orientation, calculated with a lattice parameter of $a = 4.1$ Å.

d-spacing) of $d_{220} = a/\sqrt{8}$, at least eight (220) atomic planes should be introduced into the rigid region. For the fcc (111) direction, with an inter-planar distance of $d_{111} = a/\sqrt{3}$, five planes are required. In this case, the "liquid" atoms do not interact with the bottom of the rigid sample, which is effectively a semi-infinite bulk. A simple way of modifying the substrate liquid interaction is to reduce the number of substrate planes from this value. The required width for each substrate is summarized in Table 5.2.

5.3 The initial configurations

The initial configuration for the simulations is created by joining the solid substrate to the liquid part, which is initially given as an ordered lattice. Since the solid atoms are pinned to their respective lattice positions, the whole simulation cell can be heated above the melting point of the liquid, so that a liquid-solid interface is obtained.

For the systems with fcc substrates, both the solid and liquid part had initially the same fcc symmetry, for the bcc (100) the solid had a bcc symmetry, but the liquid was initially composed of fcc (100) planes. For the bcc (110) systems on the other hand, both solid and liquid were initially constructed from bcc (110) planes. Once the liquid part had melted, it should not matter what was the initial configuration

of the atoms, hence the simplest construction method was applied in each case. In section 6.5 a detailed description is given of the annealing process each system was subjected to in order to reach an equilibrium steady state, which is independent of the initial configuration.

5.4 System notation

To facilitate the distinction between the different systems, each system is denoted in a special way as "SYMM($\vec{n}$)-X", where SYMM stands for the substrate symmetry (either fcc or bcc), $\vec{n}$ designates the orientation of the solid surface plane, and X stands for an alphabetic label. For example, fcc(100)-A is a system with an fcc substrate oriented such that its surface is a (100) plane. The second fcc system with the same surface orientation will be denoted as fcc(100)-B, etc.

5.5 Systems with face-centered-cubic substrates

A total of eight systems with fcc solid substrates were simulated, three with a (100) interface plane, two with a (111), and three with a (110). The lattice parameter of the solid in all systems was $a = 4.1$ Å. The number of atoms and the width of the solid in each system are presented in Tables 5.3, 5.4 and 5.5, for the (100), (111), and (110) systems, respectively. The fcc(100) systems are aligned such that the Cartesian x,y, and z-axes are along the [100], [010], and [001] lattice directions, The fcc (111) and (110) systems on the other hand, are aligned such that the x,y, and z axes are parallel to the [111], [$0\bar{1}1$], and [$\bar{2}11$] lattice directions, respectively. The interface in the (100) and (111) systems is therefore perpendicular to the x-axis of the simulation cell, while in the (110) system it is perpendicular to the y-axis.

The first two (100) systems: fcc(100)-A, and fcc(100)-B, share the same solid, but

System notation:	fcc(100)-A	fcc(100)-B	fcc(100)-C
Total No. of atoms:	2156	3920	2450
No. solid atoms:	294	294	588
No. liquid atoms:	1862	3626	1862
No. solid planes:	3	3	6
Width of solid (in Å):	6.15	6.15	12.3

Table 5.3: Parameters for the systems with an interface parallel to an fcc (100) plane. System notation is described in section 5.4.

System notation:	fcc(111)-A	fcc(111)-B
Total No. of atoms:	1944	3456
No. solid atoms:	324	1512
No. liquid atoms:	1620	1944
No. solid planes:	3	14
Width of solid (in Å):	7.10	33.13

Table 5.4: Like table 5.3 for the systems with an interface parallel to an fcc(111) plane.

the number of liquid atoms is different. In both systems the solid is composed of 3 (100) fcc planes, each containing 98 atoms. The resultant width of the substrate in these systems is 6.15 Å, about half the cutoff distance of the potential. The liquid part in system (A) is initially composed of 22 planes, while that of system (B) of 40 planes, each containing 98 atoms. The third system, fcc(100)-C, has a larger substrate composed of 6 planes leading to a solid width of 12.3 Å which is larger than the cutoff distance of the potential. In the lateral, x-y directions the systems are identical; there are in each case 14 (010) $\times$ 14 (001) planes.

The two (111) systems simulated are composed in the lateral directions of 18 $(0\bar{1}1) \times 36$ $(\bar{2}11)$ planes, resulting in 108 atoms per fcc (111) plane. The first system, denoted fcc(111)-A has three fixed layers, resulting in a substrate width of 7.10 Å, about half the cutoff distance. The second system, denoted as fcc(111)-B, includes

System notation:	fcc(110)-A	fcc(110)-B	fcc(110)-C
Total No. of atoms:	2160	2448	4320
No. solid atoms:	432	720	864
No. liquid atoms:	1728	1728	3456
No. solid planes:	6	10	6
Width of solid (in Å):	8.7	14.5	8.7

Table 5.5: Like table 5.3 for the systems with an interface parallel to an fcc(110) plane.

six solid planes, giving rise to a solid of width of 33.13 Å, much larger than the cutoff distance.

For the systems with the (110) interface plane, two of the systems are identical in the lateral dimensions, composed of 12 (111) $\times$ 36 ($\bar{2}$11) planes, each containing 72 atoms. In the fcc(110)-A system there are 6 rigid layers, leading to a solid of width 8.7 Å, while in system fcc(110)-B there are 10, leading to a width of $\approx$ 14.5 Å. The number of liquid atoms in both systems is identical; there are 1728 liquid atoms, initially positioned in 24 (110) planes. The last system, fcc(110)-C, has a greater lateral dimension composed of 18 (111) $\times$ 47 ($\bar{2}$11) planes, each containing 144 atoms.

5.6 Systems with body-centered-cubic substrates

A total of five bcc systems, three with a (100) terminated solid, and two with a closed packed (110) terminated solid, were simulated. The number of atoms and the width of the solid in each system are shown in Tables 5.6, and 5.7 for the (100), and (110) systems, respectively. The lattice parameter of the bcc solid substrates is chosen to be equal to the lattice parameter of the fcc systems: $a = 4.1$ Å, except for system bcc(100)-C, where the lattice parameter takes a value of 2.05 Å. The bcc(100) systems are aligned such that the Cartesian x,y, and z-axes are along the [100], [010], and [001]

lattice directions. The bcc (110) systems, on the other hand, are aligned such that the x,y, and z axes are parallel to the [110], [$\bar{1}$10], and [001] lattice directions, respectively. The interface in the bcc (100) and bcc (110) systems is therefore perpendicular to the x-axis of the simulation cells.

The solid part of the first bcc (100) interface (system bcc(100)-A) is composed of 10 (100) bcc planes giving rise to a solid of width 20.5Å, which is almost twice the cutoff distance. The cross section of the solid is composed of 10 (010) $\times$ 10 (001) bcc planes, leading to (100) planes containing 25 atoms. The liquid is initially composed of 40 fcc (100) planes each containing 50 atoms. The cross section of the liquid part is made of 10 (010) $\times$ 10 (001) fcc planes. The solid in the second system (bcc(100)-B) contains only three bcc (100) layers, giving rise to a width of 6.15Å, which is almost half the cutoff distance. In the lateral directions the solid is composed of 14 (010) $\times$ 14 (001) bcc planes, each containing 49 atoms. The liquid part is initially composed of 19 fcc (100) planes, each containing 98 atoms. In the lateral directions the liquid part is initially composed of 14 fcc (010) $\times$ 14 fcc (001) planes.

For the bcc (110) interface, two systems were simulated. In both systems the solid is composed of 6 (110) bcc planes, having a total width of approximately 17.4 Å. System (B) contains more liquid atoms and has a wider cross sectional area perpendicular to the interface plane. The solid part in system (A) is composed of 6 (110) $\times$ 10 ($\bar{1}$10) $\times$ 10 (001) bcc planes, and the liquid atoms are initially composed of 20 (110) $\times$ 10 ($\bar{1}$10) $\times$ 10 (001) bcc planes. In system (B) the solid part is made of 6 (110) $\times$ 14 ($\bar{1}$10) $\times$ 14 (001) bcc planes, and the liquid atoms are initially composed of 40 (110) $\times$ 14 ($\bar{1}$10) $\times$ 14 (001) bcc planes.

System notation:	bcc(100)-A	bcc(100)-B	bcc(100)-C
Total No. of atoms:	2250	2009	4214
No. solid atoms:	250	147	2352
No. liquid atoms:	2000	1862	1862
No. solid planes:	10	3	12
Width of solid (in Å):	20.5	6.15	12.3

Table 5.6: Parameters for the systems with an interface parallel to a bcc (100) plane. The lattice parameter for systems A and B is 4.1Å, while that for system C is 2.05Å.

System notation:	bcc(110)-A	bcc(110)-B
Total No. of atoms:	1300	4508
No. solid atoms:	300	588
No. liquid atoms:	1000	3920
No. solid planes:	6	6
Width of solid (in Å):	17.39	17.39

Table 5.7: Like table 5.6 for the systems with an interface parallel to a bcc(110) plane.

Chapter 6

Description of the simulations and calculation of properties

The main purpose of this research is to study the *structure* of the liquid at the interface and to correlate it with the structure of the solid. The structure of the liquid can be characterized by a number of quantities, or order parameters. Due to the presence of the rigid substrate, an axial anisotropy is present in the system near the interface. It is expected then that the properties of the liquid will vary as a function of the (normal) distance from the solid. The most straightforward way to study such a behavior is by dividing the liquid into slices or bins perpendicular to the interface, each of width Δz, and carrying out the property calculation for each individual bin. Figure 6.1 is an illustration of such a binning process. The width of the bins may be uniform or may change from region to region depending on the properties of the liquid. A proper choice of bin width must be made taking into account two factors. First, very small bin widths results in too few particles in each bin, leading to large statistical errors and a large scatter of the data. Second, very large bins will not show the actual dependence of the properties over distance. Hence a balance between those

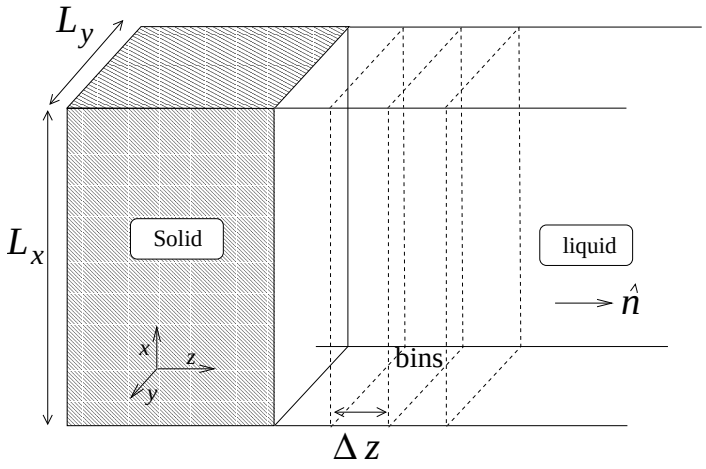


Figure 6.1: An illustration of the division of the simulation cell into bins or slices normal to the interface direction $\hat{n}$, each of width Δz. Property calculations are carried out for each individual bin separately.

two requirements must be achieved. In the analysis of the simulations a number of binning schemes will be employed.

The structural properties of the liquid calculated in this study are the translational structure order parameter $S(\mathbf{k})$, the radial distribution function $g(\mathbf{r})$, and the local density $\rho(\mathbf{r})$, in addition to more general properties such as the average potential energy per particle u_p (or the internal energy), and the kinetic energy per particle k_p. The expressions for these properties are described below.

6.1 Density profiles

The density profile $\rho(z)$ is defined as the average density of particles in a slice of width Δz parallel to the rigid surface and centered around z. The simulation cell is divided into equal layers or bins parallel to the interface. The expression for the density profile is

$$\rho(z) = \frac{\langle N_z \rangle}{L_x L_y \Delta z} \tag{6.1}$$

where L_x and L_y are the x and y dimensions of the cell, respectively, and z is perpendicular to the interface (see Figure 6.1), Δz is the bin width, and N_z is the number of particles between $z - \Delta z/2$ and $z + \Delta z/2$ at time t. The angled brackets indicate a time average.

6.2 Potential energy

The potential energy per particle is:

$$u_p = \sum_{i \in L} u_i / N(L), \tag{6.2}$$

where $N(L)$ is the number of atoms that belong to layer L, and u_i is interaction energy of the i^{th} particle, and the sum is over particles in layer L.

In addition, the temperature of each layer can be evaluated by:

$$\mathcal{T} = 2E_k^L / 3N(L)k_B = \frac{1}{3N(L)k_B} \sum_{i \in L} |\vec{v}_i^2| \tag{6.3}$$

where E_k^L is the kinetic energy in layer L, and $\vec{v}_i$ is velocity of particle i. In the algorithm the units of the simulation are normalized such that the particle mass is unity, hence $\vec{v}_i$ is also the momentum.

6.3 The static structure order parameter

The static structure order parameter is defined as:

$$S(K) = |\sum_{i,j \in L} e^{i\vec{K} \cdot \vec{r}_i}|^2 / N(L)^2, \tag{6.4}$$

where $\vec{K}$ is a vector of the reciprocal lattice, for an *fcc* lattice, we choose $\vec{K} = (2\pi/l) \times (1, -1, 1)$, where l is the length of the unit cell edge, which for our simulation model equal to the lattice parameter a. For an ordered *fcc* lattice, the order parameter is 1 because the compositional components of the $\vec{r}_i$ are all multiples of $a/2$.

6.4 The two-dimensional radial distribution function

The three dimensional radial distribution function (RDF) $g(r)$ is defined as:

$$\rho g(r) = \left\langle \sum_{i,j \in L} \delta(r - r_{ij}, \Delta r)/V(r_i, r, \Delta r) \right\rangle /N(L) \tag{6.5}$$

where r_i is the position of atom i, $r_{ij} = |\vec{r}_i - \vec{r}_j|$, ρ is the bulk density, $V(r_i, r, \Delta r)$ is the volume of a spherical shell centered at r_i with radius r and thickness Δr. The angular brackets $\langle\ \rangle$ denote time averages over simulated trajectories. δ is the usual Kronecker δ function. In practice the RDF can be measured using a histogram of discretized pair separations [48]. Let h_n be the number of atom pairs (i, j) in the n^{th} shell, i.e. atom pairs for which $(n-1)\Delta r \leq r_{ij} < n\Delta r$. Then the RDF at point $r_n = (n - 1/2)\Delta r$ is obtained from:

$$g(r_n) = \frac{V(L)h_n}{2\pi N(L)^2 r_n^2 \Delta r} \tag{6.6}$$

where $V(L)$ is the volume of the layer L. The two-dimensional RDF is defined in the same manner, except that the distances are projected onto the plane of the layer:

$$g(r_n) = \frac{A(L)h_n}{\pi N(L)^2 r_n \Delta r} \tag{6.7}$$

where $A(L)$ is the lateral area of layer L. The normalization factors ensure that $g(r \longrightarrow \infty) = 1$. although in practice the range of r is limited to half the smallest edge of the layer.

The RDF and the structure factor are related through a Fourier transform:

$$S(\mathbf{k}) = 1 + \rho \int g(\mathbf{r}) \exp(-i\mathbf{k}) \cdot \mathbf{r}) d\mathbf{r} \tag{6.8}$$

which for an isotropic system becomes:

$$S(k) = 1 + 4\pi\rho \int \frac{\sin kr}{kr} g(r) r^2 dr \tag{6.9}$$

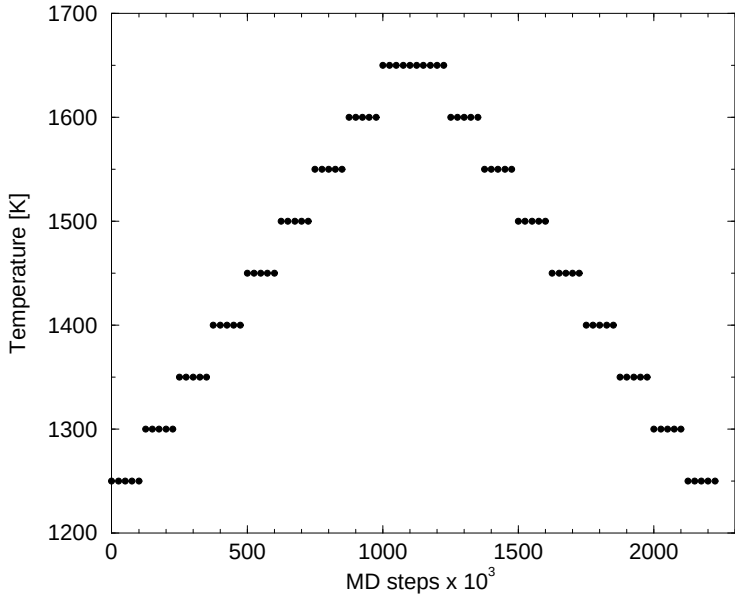


Figure 6.2: An example of an annealing schedule for the fcc(100)-A system (see Table 5.3). The state of the system at the final (target) temperature is compared with the initial state. The schedule is repeated until the system is in equilibrium, that is when the energy and density profile in the final state and initial state are the same.

6.5 Equilibration

For each system a series of simulations are conducted at temperatures ranging from $T = 800$K up to $T = 2000$K in steps of 50K or 100K. To obtain an equilibrated sample at a particular temperature, an ideal configuration is annealed by heating the system in steps to a higher temperature than required, and then cooling down in steps to the target temperature. Several different annealing schedules were employed, featuring different temperature steps, and heating and cooling cycles. Fig. 6.2 shows a typical annealing schedule. The target temperature in this case is $T = 1250$ K, an initial configuration is simulated at this temperature for 20000 MD steps, then the system is heated by 50K, and then equilibrated for additional 20,000 MD steps. This process is continued up to a maximum temperature which in this case is $T = 1400$ K. Then

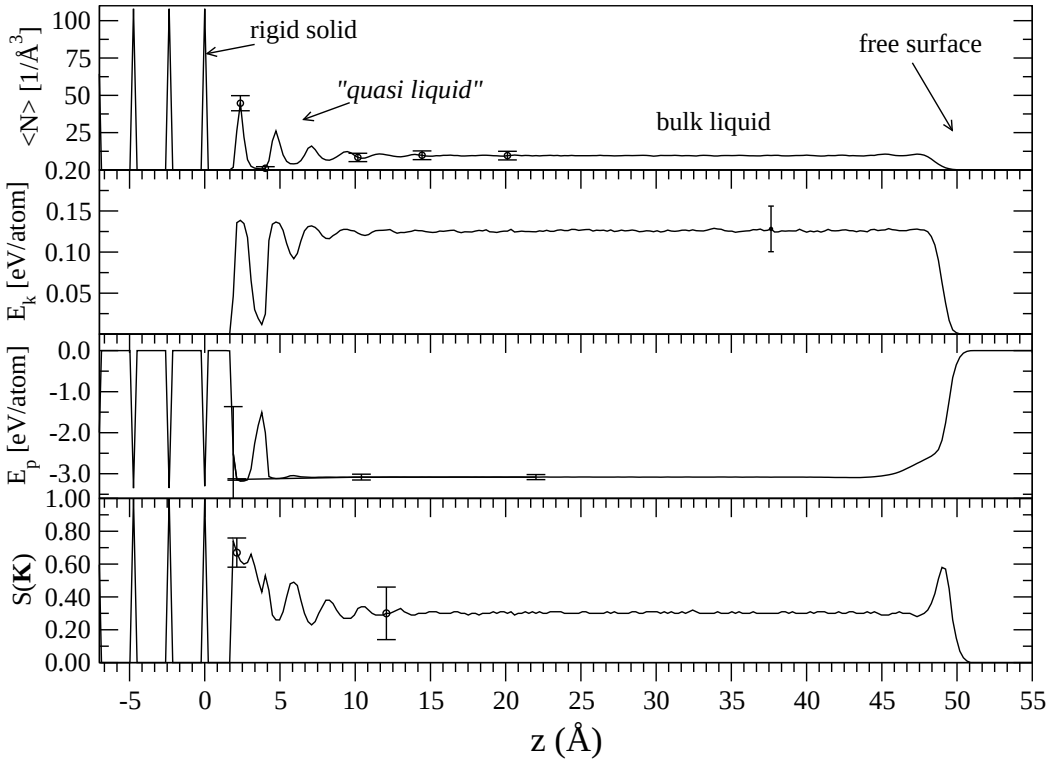


Figure 6.3: The profiles of the mean number of atoms $\langle N \rangle$ (top panel), the potential energy per particle u_p, the kinetic energy E_k, and the static structure order parameter $S(\mathbf{K})$ in each bin. Data is from system fcc(111)-B at $T = 1100K$. Typical error bars are indicated.

a cooling process begins, whereby the system is cooled down in steps of 50 K. The annealing processes is essential to obtain an equilibrium density profile, especially at low temperatures, where the system is liable to get stuck in a metastable condition. Once the final state of the different annealing schedules is the same, the system can be safely regarded in equilibrium. At least two annealing processes were applied on each system.

6.6 Sampling

Following the equilibration stage, the system is further subjected to a number of long MD runs at a constant temperature or constant energy during which sampling of quantities is performed. Measurements of the density profile were obtained by accumulating $\rho(z)$ via equation (6.1) over a number of MD cycles, each containing 20,000 to 50,000 MD steps, then averaging over these cycles to produce a single measurement. In equilibrium, these measurements of $\rho(z)$ do not change substantially in time, and are independent of the annealing schedule.

Typical results obtained from the simulations are illustrated in Figure 6.3, where the profiles of the mean number of atoms in each bin $\langle N \rangle$, the average potential energy per particle u_p, the averaged kinetic energy E_k, and the static structure order parameter $S(\mathbf{K})$ in each bin are shown for fcc(111)-B at $T = 1100$ K. The rigid substrate terminates at $z = 0$, and only three of the 14 solid layers are shown. The width of each bin is $0.1d$, where d is the d-spacing of (111) planes, resulting in a bin width of 0.2367 Å. Each profile is obtained by averaging over 5000 MD steps. The structure of the liquid at the interface is manifested through the oscillations in the profiles, especially from the oscillation of the mean number of atoms $\langle N \rangle$ and the structure factor $S(\mathbf{K})$. The form of the energy profiles correlates with that of the number density. The atoms concentrate in regions with low potential energy, the maxima or peaks of the density profiles are the location of the layers which corresponds to the minima in the potential energy profile. Comparing between the different quantities in Figure 6.3, it is observed that the oscillations in the average potential energy dampen faster than the others; variations in the number density, structure order parameter, and kinetic energy extend further into the liquid.

One advantage of using a more realistic metallic potential (instead of a simpler pair potential such as a Lennard-Jones 6-12 potential), is that it gives the realistic

low vapor pressure characteristic of liquid metals. Simulations that were performed with a LJ potential resulted in evaporation of a large portion of the liquid. In the case of liquid aluminium, the low vapor pressure results in as little as 10^{-8} vapor atoms in a simulation box of cubic side equal to ~ 40Å, therefore, in effect no vapor atoms are observed during the simulations. Note how this is manifested in Figure 6.3, where the number density decays at the free surface to zero, indicating that there is no evaporation in the samples within the time scale of the simulation.

Chapter 7

Results

In what follows the results of the simulations are presented first for the systems with the fcc substrates, with a discussion of some of the technical issues, followed by the results for the bcc systems, with emphasis on the differences between the two structures.

7.1 Results for the face-centered-cubic systems

The number density profiles, that is the average number of atoms in each layer normalized by the volume of the layer $\rho(z)$ (see equation 6.1) of the liquid part are shown in Figure 7.1, 7.2, and 7.3, for the (100), (111) and (110) fcc systems, respectively. These profiles are taken from simulations on the samples having a substrate with a width larger than the cutoff of the potential (the particular systems are indicated in the figures). The density profiles are shown at three temperatures for each system. The dashed vertical lines indicate the locations of the density peaks (or local maxima) at $T = 1000$ K (top panel in the figures) and are extended to the other panels for comparison. The solid substrate terminates at the point $z = 0$ in all systems.

The density profiles show large oscillations corresponding to the layering of the

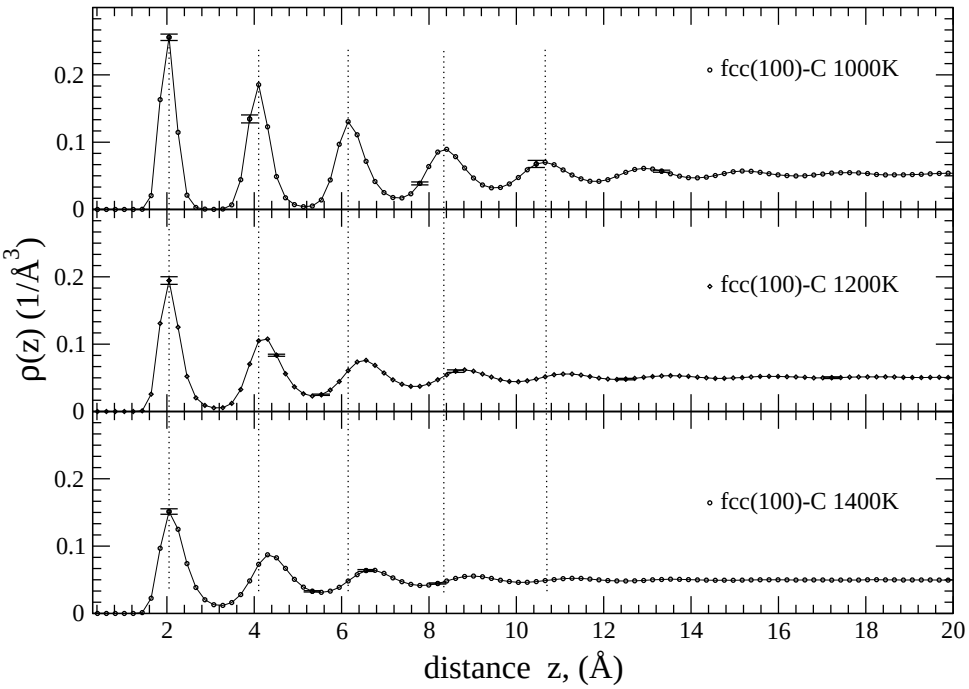


Figure 7.1: The density profile $\rho(z)$ as a function of distance z from the hard wall is shown for a (100) fcc system at three temperatures: 1000K, 1200K, and 1400K. Vertical lines mark the locations of the density peaks at 1000 K. Data is from a simulation on system fcc(100)-C described in section 5.5.

liquid near the hard wall. The oscillations dampen gradually within the interfacial region, until the uniform density of the liquid phase is reached, which is the same in all systems and is approximately equal to: $\rho_l = 0.051 \pm 0.005$ atoms/Å^3, (this density should change slightly with temperature according to the phase diagram of bulk liquid aluminium). It is not yet clear what the state of matter is at the interfacial region, it is neither bulk liquid nor solid. Hence, hereafter it will be referred to as a "quasi-liquid", or equivalently "quasi-solid" region. The damping of the density oscillation is accompanied by an increase in the width of the layers. The location of the first peak of the "quasi-liquid" is constant in temperature for all systems. For the (100) system, the location of the peaks, starting from the second one seems to shift away from the solid substrate, as illustrated in Figure 7.1 by the dashed lines. For the

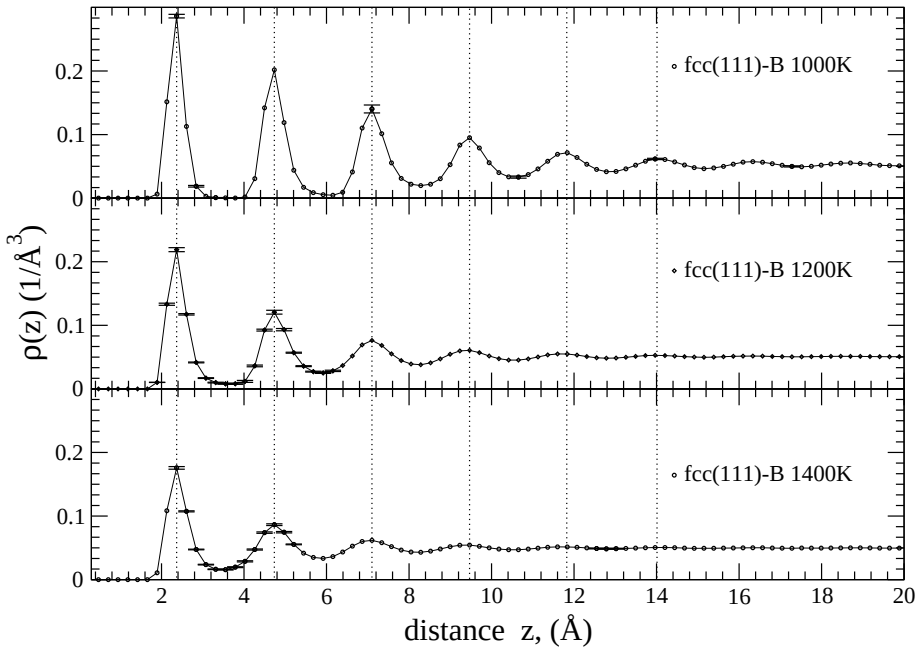


Figure 7.2: The density profile $\rho(z)$ as a function of distance z from the hard wall is shown for a (111) fcc system at three temperatures: 1000K, 1200K, and 1400K. Vertical lines mark the locations of the density peaks. Data is from a simulation on system fcc(111)-B described in section 5.5.

(111) systems on the other hand, the location of the density peaks remains constant as temperature increases (Figure 7.2). For the (110) systems, there is an apparent slight shift in the location of the peaks *towards* the solid substrate, as illustrated in Figure 7.3 by the dashed lines.

The density profiles for the three systems at 1000 K are shown in Figure 7.4. The location of the solid terminating plane is at $z = 0$ as before. The first remarkable result is that ordering at the (111) and (100) interfaces extends further into the liquid than at the (110) interface. However, the number of density peaks is very similar in all directions, while the distance between consecutive peaks and the width of the layers, varies substantially with interface direction. This distance, or interlayer spacing can be identified with the d-spacing of the "quasi-solid" region within the liquid. In

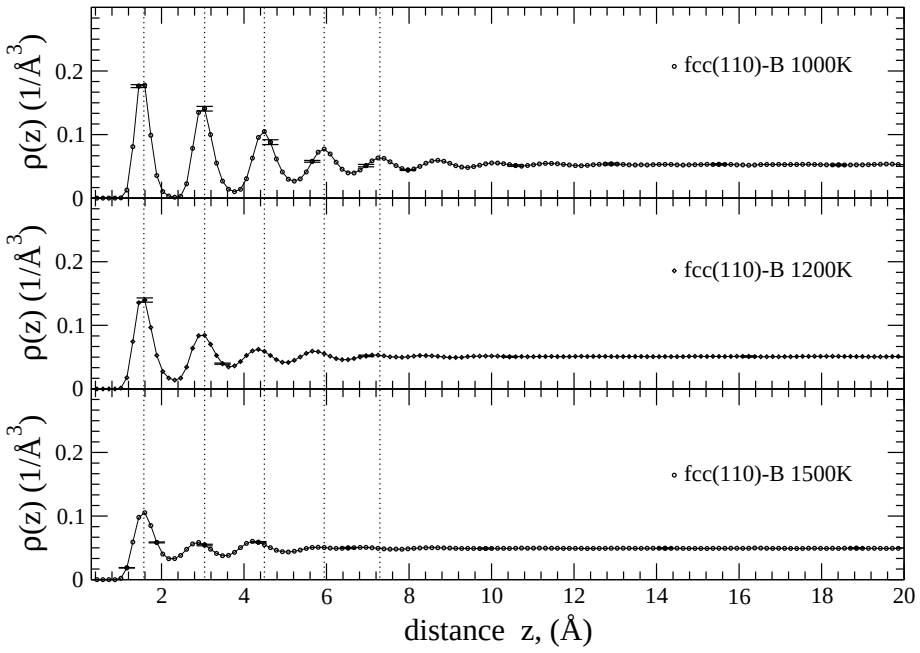


Figure 7.3: The density profile $\rho(z)$ as a function of distance z from the hard wall is shown for a (110) fcc system at three temperatures: 1000K, 1200K, and 1500K. Vertical lines mark the locations of the density peaks. Data is from a simulation on system fcc(110)-C described in section 5.5.

fact there are two measures that can be used for the interlayer spacing. The first is related to the distance between consecutive density peaks, d_p, and the other is related to the distance between consecutive minima (or valleys) in the density, as indicated in Figure 7.4. Both quantities were calculated for all systems at different temperatures.

The interlayer spacing from the peak data d_p for the (100), (111), and (110) systems is presented in Figures 7.5, 7.6, and 7.7, respectively, at $T = 1000$ K, $T = 1200$ K, and at $T = 1400$ K. The dashed horizontal lines in each figure mark the location of the d-spacing of the corresponding solid substrate. The dashed vertical line marks the position of the last layer belonging to the solid substrate. For the (100) fcc (Figure 7.5) system at T=1000K, the interlayer spacing in the "quasi-liquid" is equal to the d-spacing in the solid substrate, $d = 2.05$ Å, in the first four layers, and

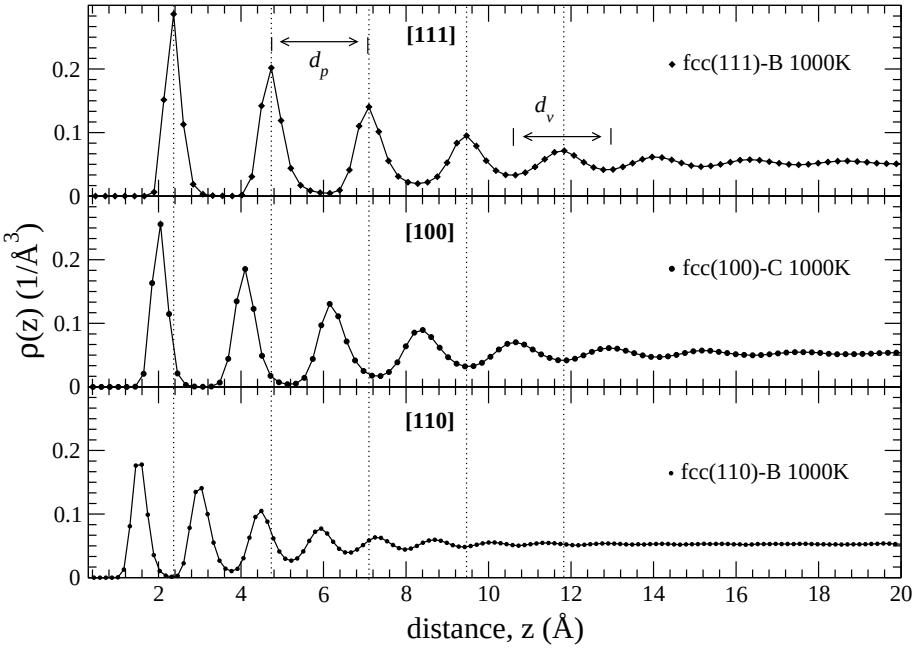


Figure 7.4: Comparison of density profiles for the fcc systems at a temperature of 1000K. The vertical dashed lines mark the positions of the peaks for the (100) system. d_p indicates the interlayer spacing as calculated from the peak data and d_v is the interlayer spacing calculated from the minima, or valleys of the density profile.

then it increases steadily to a maximum value of 2.323Å at the 6^{th} layer. As the bulk liquid region is approached, the distance between the peaks decreases. Well inside the bulk region the calculation of the interlayer spacing is of course, meaningless. The d-spacing at higher temperatures shows similar behavior, except that the increase in the interlayer spacing already occurs at the second peak.

It is important to note that the low value of the d-spacing for the last peak at $T = 1000$ K, and the last three peaks for the higher temperature curves is not a realistic measure, in fact it is an artifact of the calculation algorithm. The calculation of the d-spacing is made with an algorithm that searches for maxima in the density profile, and then calculates the difference in the locations of each consecutive maxima.

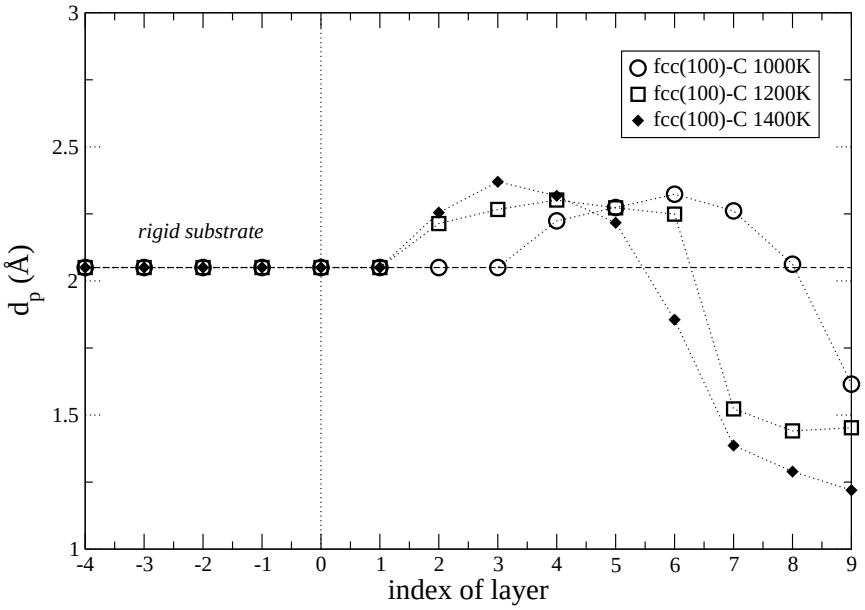


Figure 7.5: The interlayer spacing from the peak data d_p, for the fcc (100) substrate at $T = 1000$ K, $T = 1200$ K, and $T = 1400$ K.

This difference is then averaged over many instances to give one estimate of the d-spacing. The data includes the analysis of the d-spacing even well inside the liquid region, where the "maxima" are in random locations and are due to noise in the data. The average on these gives the low values seen in the figure.

For the (111) fcc system (Figure 7.6), the d-spacing in the "quasi-liquid" is constant, and equal to the d-spacing of the corresponding solid substrate, $d_{111} = 2.367$ Å. There is no substantial deviation of the d-spacing as the liquid is approached, unlike the behavior for the (100) system. At a higher temperature, deviations in the interlayer spacings are observed, albeit very small, and lie with in the error bounds of the measurements.

For the (110) fcc system (Figure 7.7), a similar behavior as for the (111) system is observed. However, the first solid peak is at a distance somewhat larger than the d-spacing of the solid substrate ($d_{110} = 1.449$ Å). This distance is within the bounds

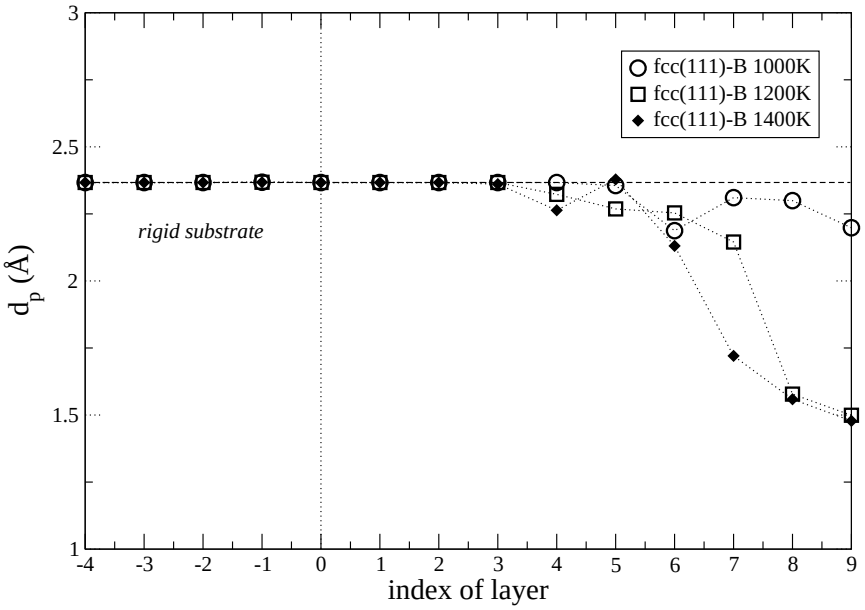


Figure 7.6: The interlayer spacing from the peak data d_p, for the fcc (111) substrate at $T = 1000$ K, $T = 1200$ K, and $T = 1400$ K.

of the statistical and systematic errors in determining the d-spacing, which is related to the width of the bins used. The effect of this systematic error is evident from the small fluctuation of the d-spacing of the last *rigid* layer, as indicated in the figure.

The d-spacings for the three systems are compared in Figure 7.8. It is clear from this figure that the d-spacing in the (100) system moves towards that of the (111) system. Similar behavior was previously observed [40, 45, 55], and is believed to arise from the fact that perturbations within the liquid are more energetically favored if their periodicity is $2\pi/Q_0$, where Q_0 is the wave vector at the first peak of the structure factor $S(Q)$. For an fcc liquid metal this distance is very close to d_{111} [55]. For the (110) case a shift in the interlayer spacing is not observed. This may be attributed to the large difference between d_{110} and d_{111}. The interlayer spacing obtained from the minima or valleys of the density leads to similar results as obtained from the peaks or maxima of the density oscillations, d_p, as shown in Figure 7.9.

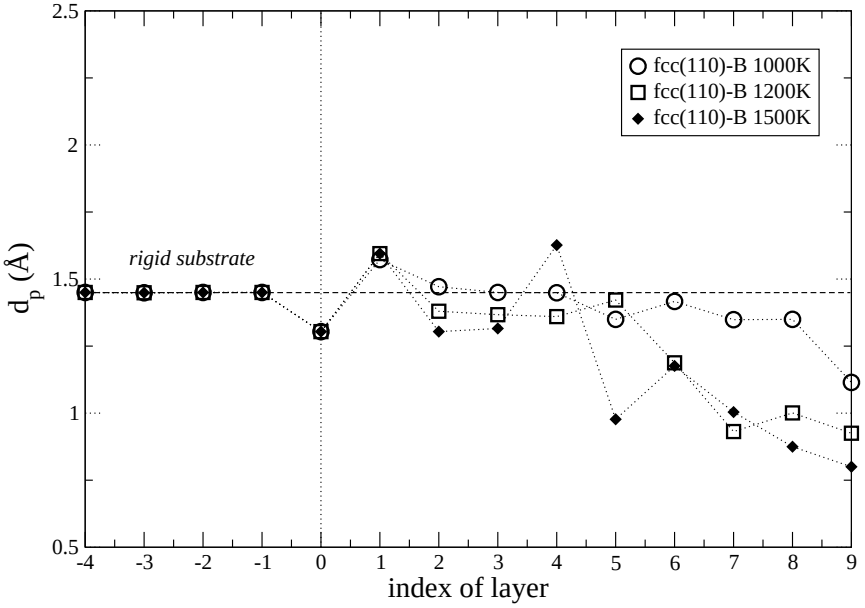


Figure 7.7: The interlayer spacing from the peak data d_p, for the fcc (110) substrate at $T = 1000$ K, $T = 1200$ K, and $T = 1400$ K.

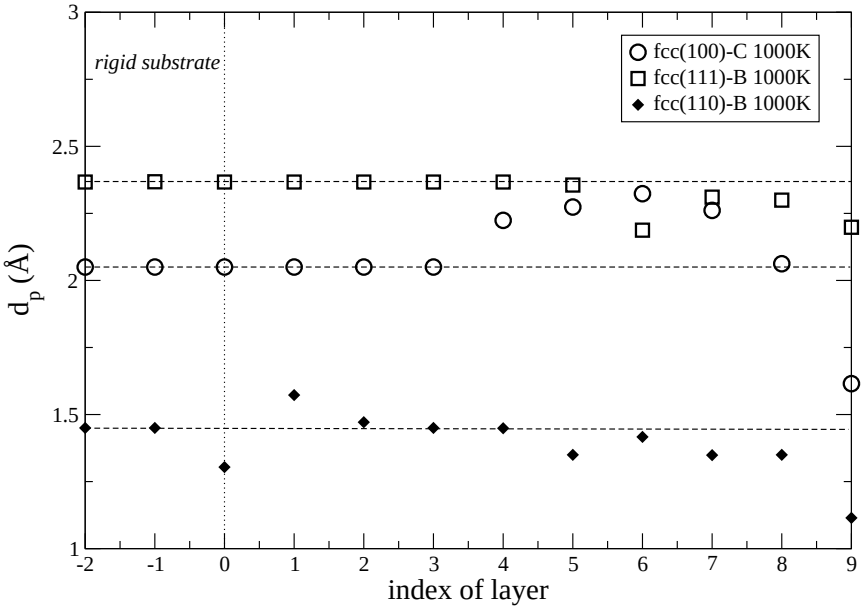


Figure 7.8: The interlayer spacing from the peak data d_p for the three fcc systems at $T = 1000$ K.

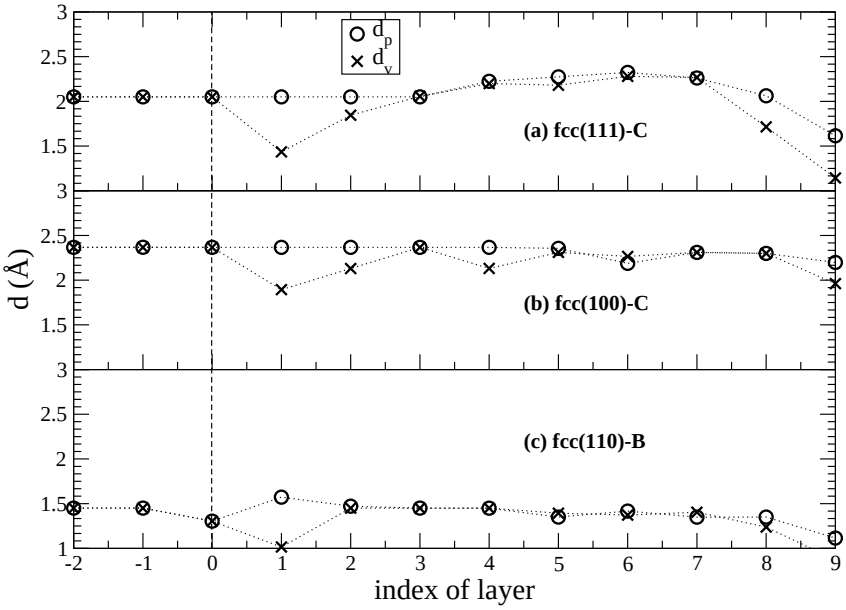


Figure 7.9: The interlayer spacing from the peak data d_p, and from the valley's data d_v, calculated at $T = 1000$ K for the three fcc faces as indicated in the figure.

7.1.1 Quantitative measure of ordering

In order to analyze the decay of the density profile as a function of temperature and substrate orientation, the envelope of the density profile is extracted and plotted against distance from the surface, as shown in Fig. 7.10 (a). The profile is then fitted assuming an exponential decay:

$$\rho(z) \sim \exp(-\kappa z) \tag{7.1}$$

where $\kappa = 1/\xi$, and ξ is the correlation length at the interface. The parameter κ can then be used to *quantitatively* describe the amount of *disorder* at the interface. The exact function that has been used in the fitting has an extra constant term, b, to account for the background liquid density, and a normalization factor, a:

$$\rho(z) = a\mathrm{e}^{-\kappa z} + b \tag{7.2}$$

Note that the abscissa in Figure 7.10 (a) is given as the average number of particles per layer, however the results of the fit for κ are not affected, the other parameters a and b do change, but this is irrelevant to our purpose. This form of decay of the density profile is typically obtained from a mean field treatment of binary fluid interfaces [31]. A good check of the validity of this assumption is that $log(\rho(z) - b)$ is linear in z. A few typical examples are shown in Fig. 7.10 (b). The large scatter for points far from the solid-liquid interface, i.e. well within the normal liquid phase, is due both to the sensitivity of the logarithm to small arguments, since the difference between $\rho(z)$ and b for points inside the liquid is very small, and to the fact that close to the bulk liquid the exponential decay is no longer valid.

The results of the density profile analysis for the three fcc orientations, (111), (110) and (100) are shown in Fig. 7.11, where the disorder parameter κ is plotted against temperature for the three systems; fcc(100)-C, fcc(111)-B, and fcc(110)-B. The amount of disorder, κ, is consistently smaller for the (111) and (100) surfaces

than for the (110) system, as shown in Fig. 7.11. In other words, the ordering at the (111) and (100) interfaces is larger, and extends further into the liquid, than the ordering at the (110) interface. Another remarkable feature is that the disorder in the (110) orientation increases faster than the disorder in the (111) and (100) orientations, and almost linear with temperature. It is not possible to distinguish between $\kappa(fcc(111))$, the disorder in the (111) interface, and $\kappa(fcc(100))$, although for lower temperatures (below 1300 K), $\kappa(fcc(100)) > \kappa(fcc(111))$, while for higher temperatures $\kappa(fcc(100)) < \kappa(fcc(111))$, this change is small compared to the error bounds on κ, which result from both statistical fluctuations in the density peaks, and from the fitting process. The proximity of κ for the two fcc closed-packed surfaces (111), (100) conforms with the picture emerging from the d-spacing, namely that the d-spacing in the (100) shifts to a value close to that of the (111) at the same temperature, therefore the behavior of the density may be similar.

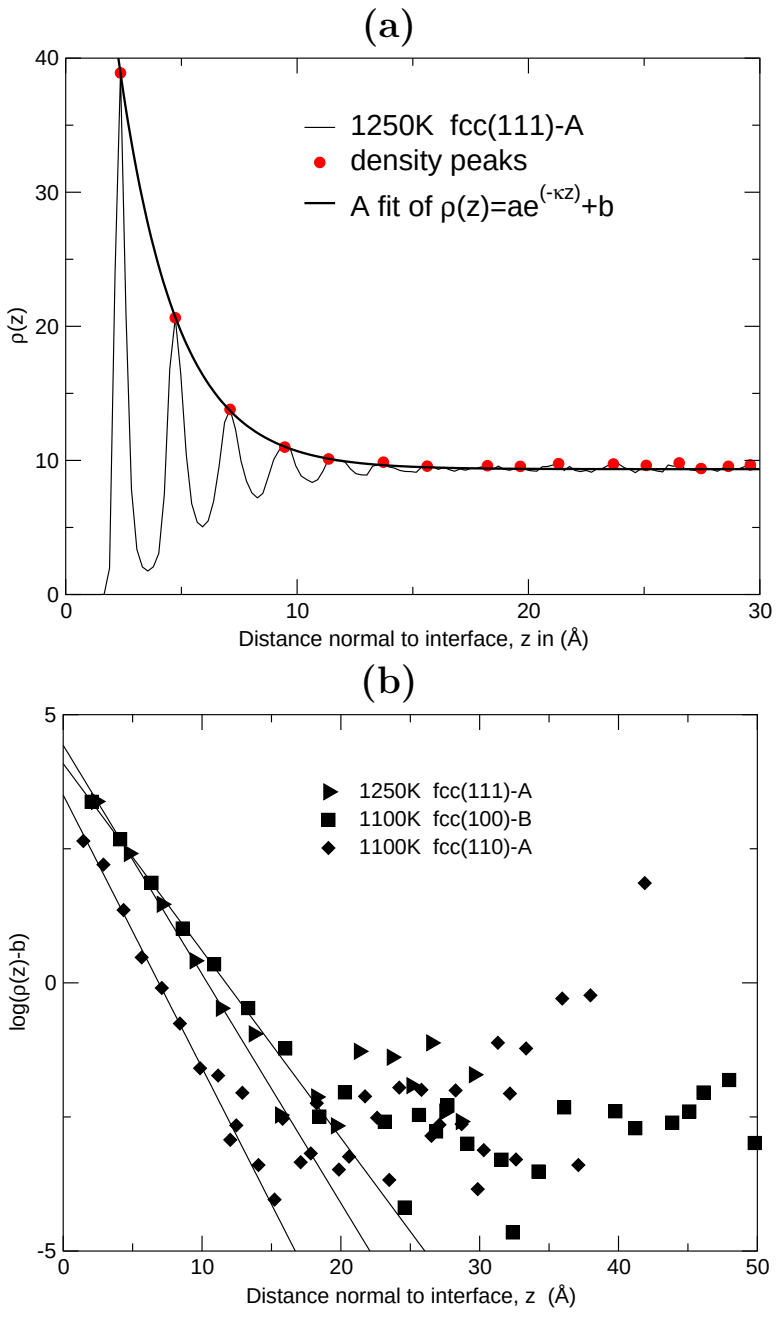


Figure 7.10: (a) The envelope of the density profile obtained for a (111) fcc sample at $T = 1250K$, and (b) test for the fitting (see text).

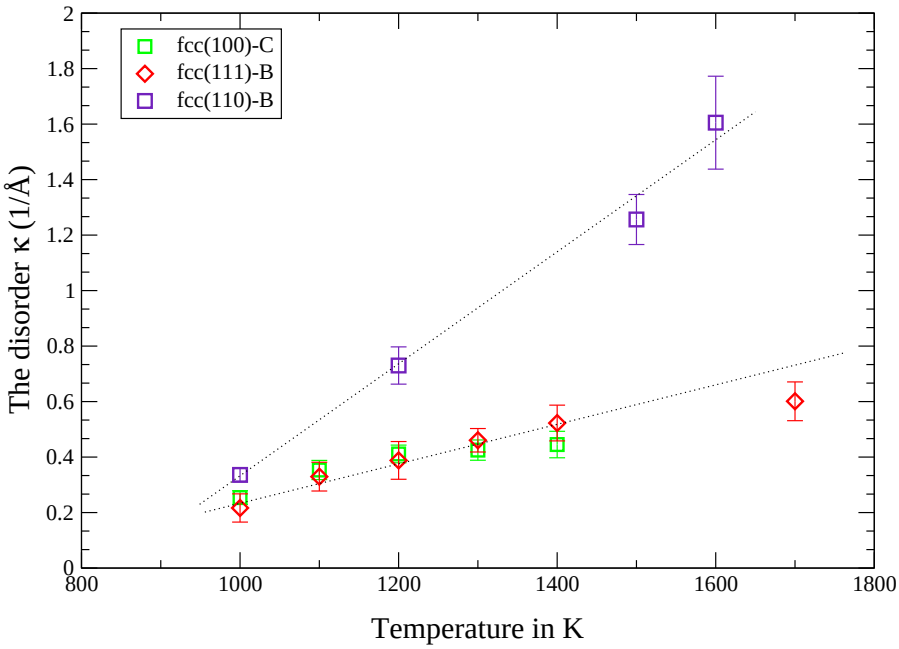


Figure 7.11: The disorder parameter κ vs. temperature for the three fcc surfaces. The dashed lines are meant to guide the eye only, and are not fits to the data.

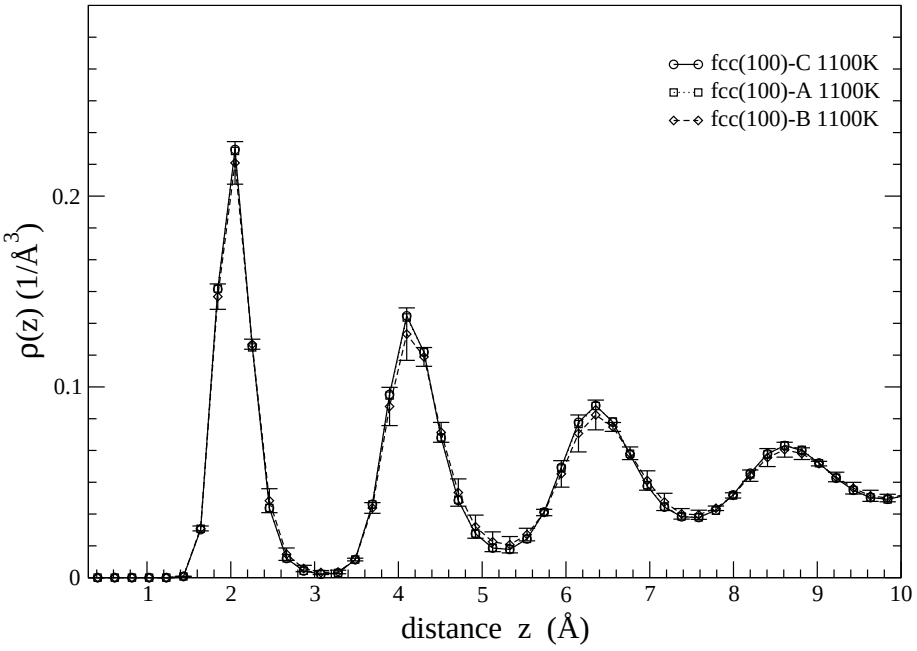


Figure 7.12: Comparing density profiles for the different fcc (100) systems studied at 1100K.

7.1.2 Effect of substrate widths and system size

The density profiles for the three fcc (100) systems studied, namely fcc(100)-A, B, and C, are shown in Figure 7.12 at $T = 1100$ K. Within the statistical error bounds, the three profiles are similar. The width of the substrate in systems fcc(100)-B and A is: 6.15 Å (see table 5.3), almost half the cutoff distance of the potential, and that of the C system is 12.3 Å. Nevertheless, this does not affect the ordering in the "quasi-liquid" region. Systems fcc(100)-A and B, differ in that the extent of the liquid region is larger. The exact profiles obtained for the two systems indicate that there is no interference between the free liquid surface and the solid liquid interface in either system. The same density profile is obtained for the two (111) systems, shown in Figure 7.13, where the solid width of system A is 7.10 Å, and that of system B is 33.1 Å (the latter being larger than required, see Table 5.4).

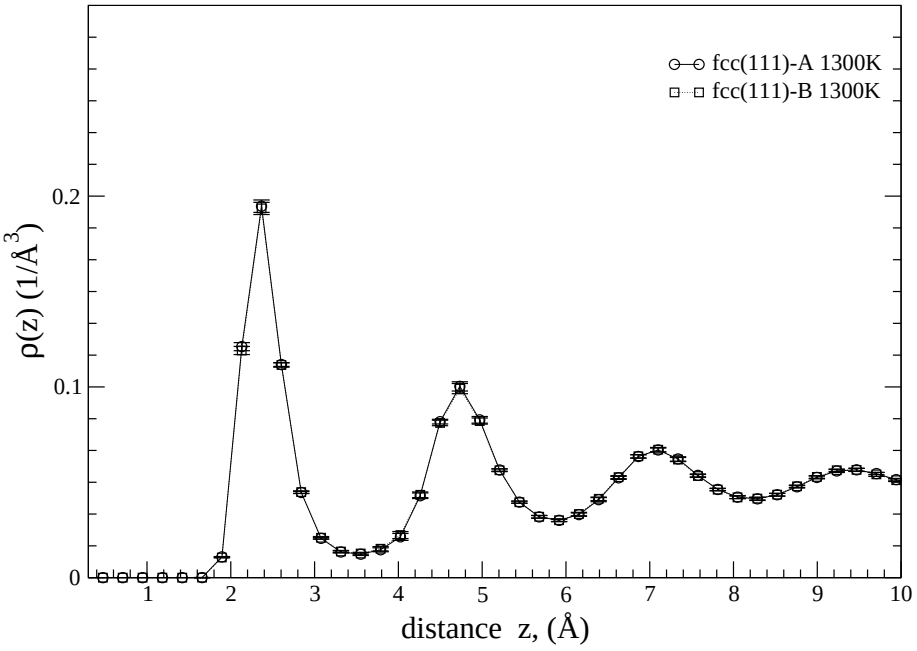


Figure 7.13: Comparing density profiles for the different fcc (111) systems studied at 1300K.

For the (110) system on the other hand (see Figure 7.14), where the solid widths are 8.7, 14.5, and 8.7 Å for the A, B, and C systems respectively (see Table 5.5), the density profiles clearly depend on the width of the solid. The density oscillations for the B system, with the thicker substrate, are displaced further away from the solid substrate with respect to the density profiles of the thinner substrates. Also, the density peaks, or maxima, are smaller, and dampen faster for the thick substrate.

The disorder parameter κ calculated for three fcc (100) systems should then be similar, and also that calculated for the (111) systems, as can be seen in Figure 7.15. The disorder parameter κ for the (110) systems is also shown in the figure. The remarkable result is that ordering for the (110) system with the smaller substrate is *larger* than in the case of a substrate with the full width of the solid.

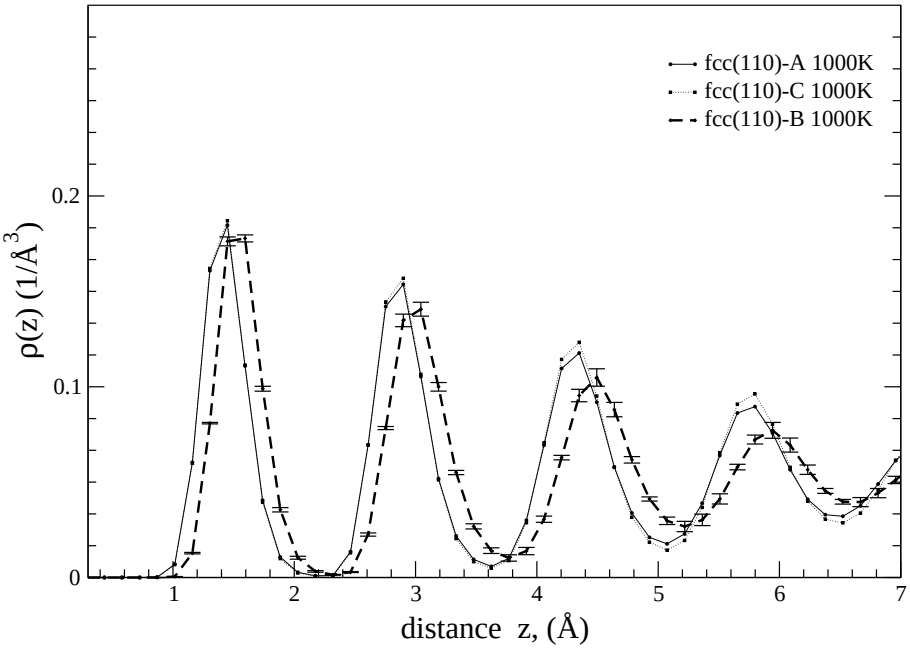


Figure 7.14: Comparing density profiles for the different fcc (110) systems studies at 1000K.

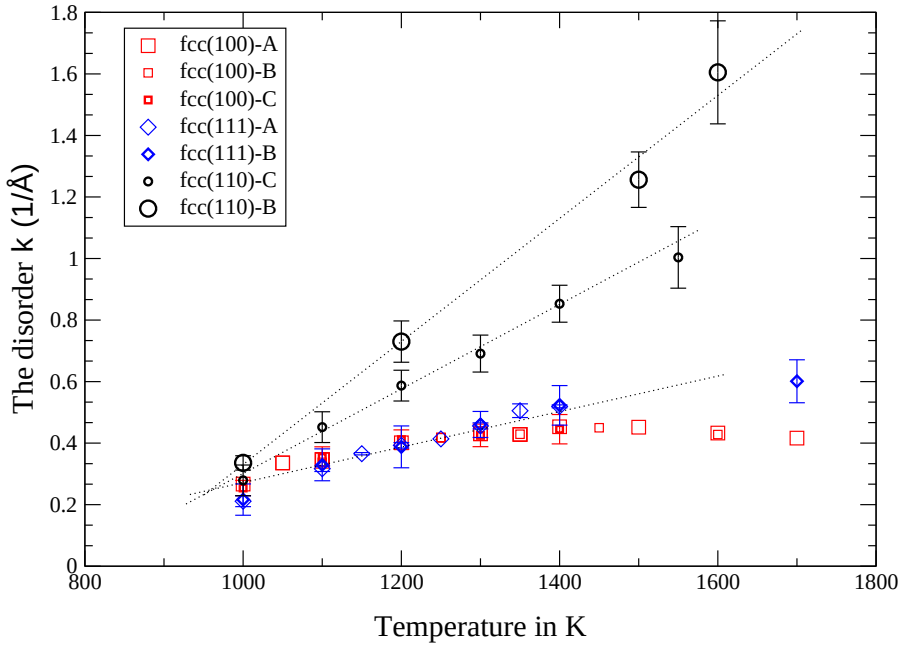


Figure 7.15: The disorder parameter κ vs. temperature for all fcc systems.

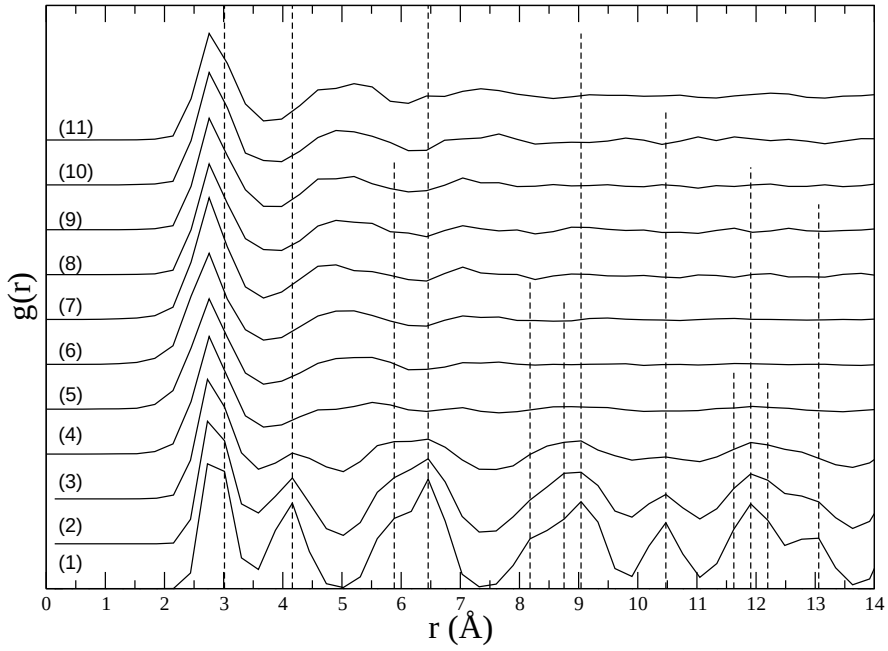


Figure 7.16: The two dimensional radial distribution functions for the fcc(100) system at $T = 1000K$. The RDFs for the different layers are stacked one above the other.

7.1.3 In-plane ordering

The ordering within the layers can be inspected via the two dimensional radial distribution function (2D RDF), $g(r)$, calculated in each layer. The 2D RDFs are shown in Figures 7.16, 7.17, and 7.18 for the fcc(100), fcc(111), and fcc(110) systems, respectively. As seen from these figures the order in the layers decreases in a monotonic manner. The loss of order as the interface is traversed is seen to occur first for the features in the RDF corresponding to the higher neighbors. In addition the height of the peaks and the area under $g(r)$ vary, which reflects the change in density across the interface.

In all orientations the ordering with in the first few layers of the "quasi-liquid" (or "quasi-solid") is compatible with that of the solid substrate. The transition from an ordered layer to a layer with a liquid 2D RDF in the fcc(110) system occurs in

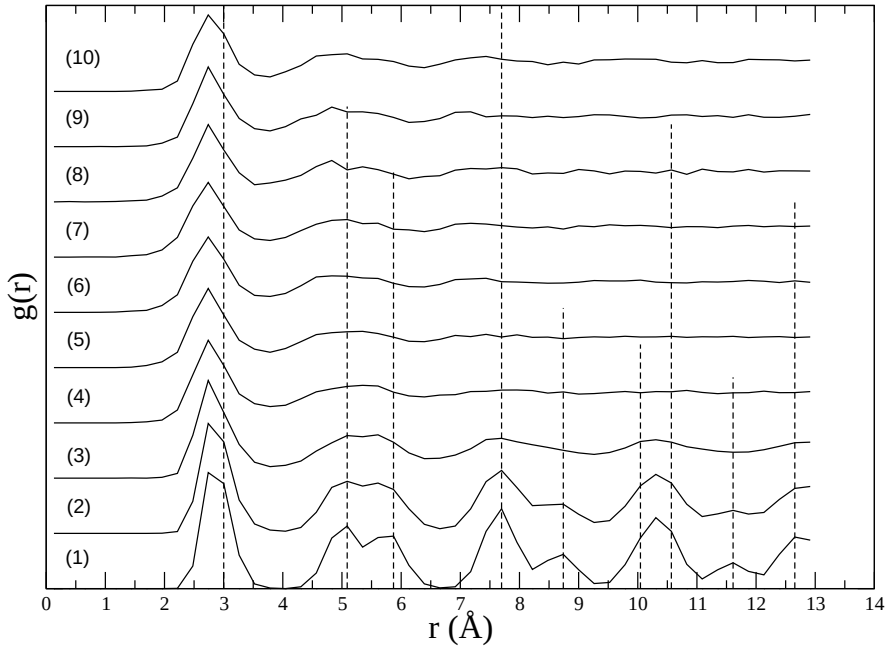


Figure 7.17: The two dimensional radial distribution functions for the fcc(111) system at $T = 1000K$. The RDFs for the different layers are stacked one above the other.

the 7^{th}-8^{th} layers, where a complete loss of the ordered features is observed. In the fcc(100) and fcc(111) direction the transition occurs in the 4^{th}-5^{th} layers. The in-plane ordering as reflected by the 2D RDFs, indicates that it decays faster with distance from the interface than the ordering perpendicular to the interface. The number of pronounced layers in the density profile in each orientation is of the order of 7-8 layers, at least. This indicates that the ordering in the planes and the ordering normal to the planes (i.e. the layering) are not strongly correlated. In other words, far enough from the solid substrate, the metal exhibits a stratified structure, with an interlayer spacing, which for the fcc(111) and fcc(110) systems is compatible with that of the solid, and in the case of the fcc(100) it has a specific interlayer spacing very close to that found in the (111) systems.

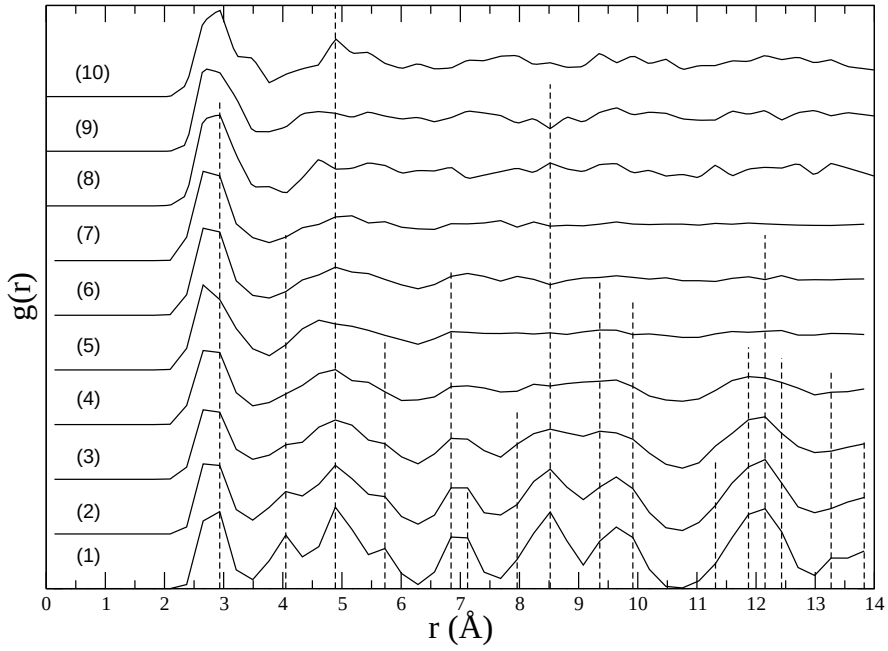


Figure 7.18: The two dimensional radial distribution functions for the fcc(110) system at $T = 1000K$. The RDFs for the different layers are stacked one above the other.

7.2 Results for the body-centered-cubic systems

The density profiles at three temperatures for the bcc(100)-A system are shown in Figure 7.19, and for the bcc(110)-A in Figure 7.20. As before the solid surface terminates at the origin, and the solid layers are not shown in the figures. In both cases the behavior is similar to that obtained for the fcc substrates. The average number density in each layer, decays gradually to the bulk density of the liquid. However, there is an obvious new feature in the case of the bcc(100) interface, where the first peak in the "quasi-liquid" is consistently smaller than the others, and is suspiciously close to the solid, being only 0.615 Å from it. Figure 7.21 shows the density layering in the (100) system, with a closer look at what happens near the rigid substrate. The first two peaks shown correspond to the solid substrate. In addition to the layers well inside the liquid, there is an additional peak in the density profile very close

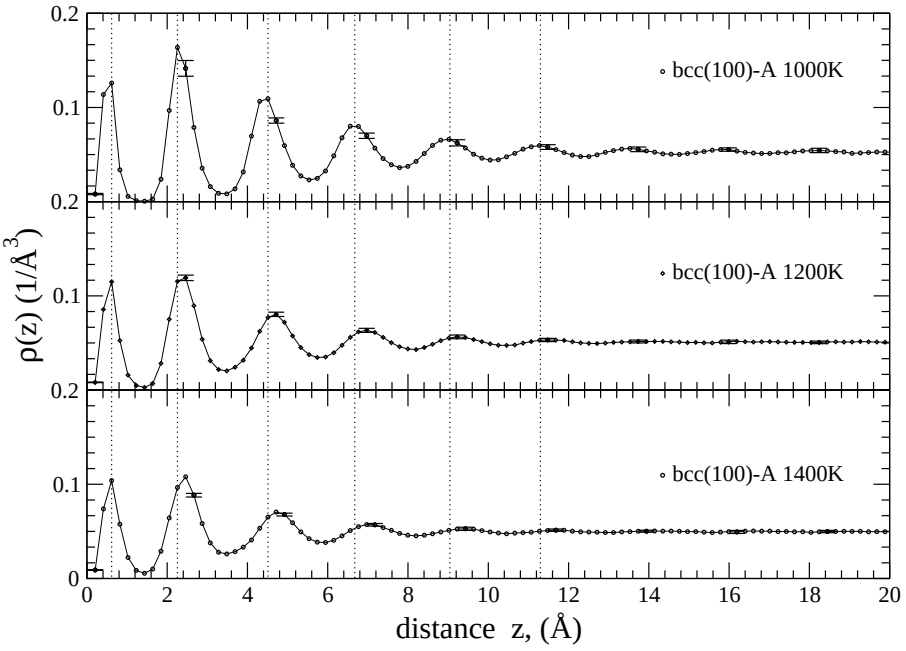


Figure 7.19: Comparing density profiles for the (100) bcc system at different temperatures.

to last peak of the substrate. A better understanding of the source of this peak is achieved by looking at the actual structure of the interface, as shown in Figure 7.22. The left panel is a side view, looking down from the [010] direction, of a part of the interface showing the last two solid layers, labeled (a), and (b), and the two liquid layers adjacent to the substrate, labeled (c) and (d). The right panel is a top view, looking down from the [100] direction. The size of the atoms drawn in the right view was changed to make it easier to discern the different layers. Layer (a) is drawn with larger black spheres, followed by smaller dark spheres for layer (b). It is clear that the extra peak in Figure 7.21 is due to the layer (c) (dark gray atoms in Figure 7.22) formed by the liquid atoms that penetrate, or adsorb, into the terminating plane of the rigid substrate. These atoms fall into the face-centered positions of the bcc (100) plane, immediately above the positions of the atoms in layer (a), in accordance with the fcc interaction potential of the liquid. In this way, the terminating plane of the

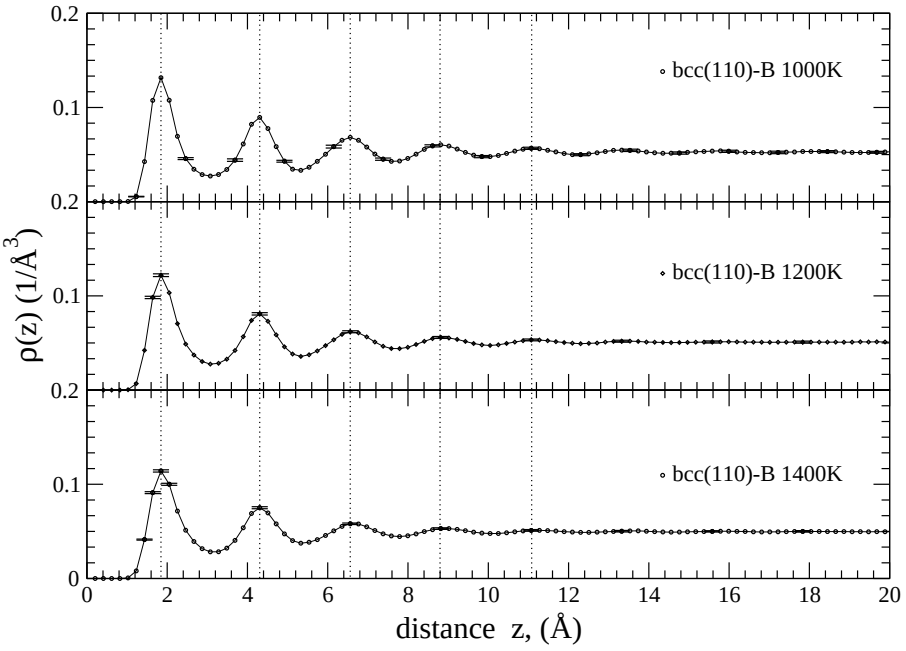


Figure 7.20: Comparing density profiles for the (110) bcc system at different temperatures.

bcc (100) material transforms, via the adsorption of the liquid, into an **fcc** (100) plane. Note that half of the atoms in this plane are rigid and the other half slightly displaced above the ideal positions of the face-center atoms. Hence it is expected that the structure of the liquid at the bcc (100) substrate will be very similar to that found at an actual fcc (100) substrate. The adsorbed layer effectively becomes part of the terminating solid surface.

The interlayer spacing for both bcc systems shown in Figure 7.23 is examined next. The closed symbols indicate the d-spacing found in the ideal structure (the small solid peak in the case of the (110) system is due to rounding off errors in the slicing scheme). In both systems the interlayer distance of the quasi-liquid is *uncorrelated* with the d-spacing of the substrate. This suggests that in both cases the in-plane structure in the liquid is not similar to that of the substrate. Moreover, apart from first layer in each system, the interlayer spacing in both cases is seen to

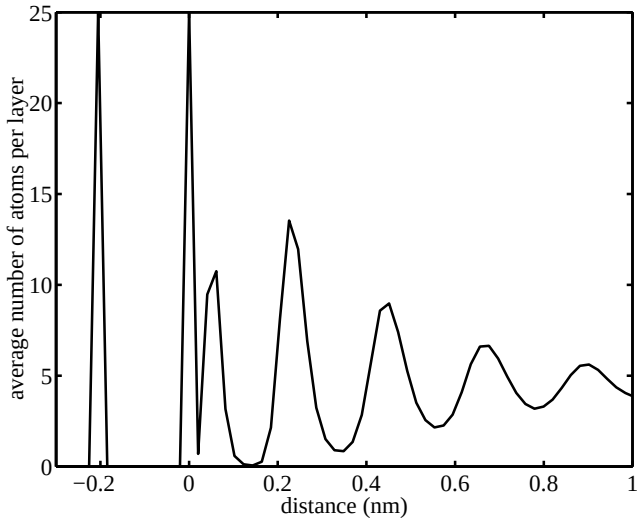


Figure 7.21: The profile of the bcc (100) system showing an extra peak very near to the solid peak, which is due to the adsorbed liquid layer.

fluctuate around a value which is close to the d-spacing of a (111) family of planes in an ideal *fcc* material, similarly to what was obtained for the fcc(100) system. The data for the bcc(100) system shown in the figure include the adsorbed layer, which is responsible for the low d-spacing in the first point. The first "quasi-liquid" layer in both systems has a smaller d-spacing than the other layers.

Figure 7.24 shows the disorder parameter in the bcc systems. The bcc(110) system is consistently more disordered than the bcc(100) system, and at higher temperatures the systems deviate substantially, and the bcc(100) system is seen to preserve more order than the bcc(110) system. Note that the adsorbed liquid layer is part of the terminating plane of the solid, so that when calculating κ it has been excluded from the density envelope.

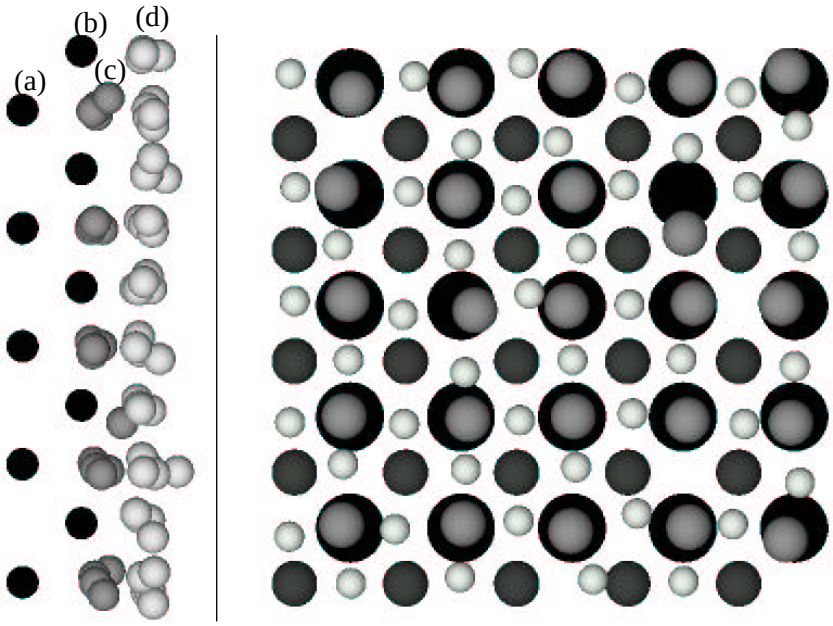


Figure 7.22: The interface region at a bcc (100) substrate at $T = 1000K$. The left panel is a side view and the right panel is a top view. The darkest atoms represent the solid, medium and light gray, the liquid. On the right, different sizes indicate layer position: layer (a) is represented by large spheres, layers (b) and (c) by normal size, and layer (c) by a smaller size. Liquid atoms (dark gray) in layer (c) are adsorbed into the solid, they lie in the *face-centered* sites on the bcc (100) plane, but positioned slightly above the (100) plane as seen from the left panel.

7.2.1 In-plane ordering

The atoms of first liquid layer in the bcc(100) system, shown in light gray in Figure 7.22 and labeled (d), fall into sites directly above holes in the terminating substrate. There exists a large amount of order in this layer, as can easily be seen from the right panel of Figure 7.22. The location of the second and third liquid layers is shown in Figure 7.25(a). The atoms in the second layer fall into well defined sites, above the holes in the first layer, and above atoms in layer (b). In order to see more clearly the actual stacking of the planes in the liquid phase, the sample is cooled down and the atoms are allowed to settle into their equilibrium positions. Figure 7.25(b)

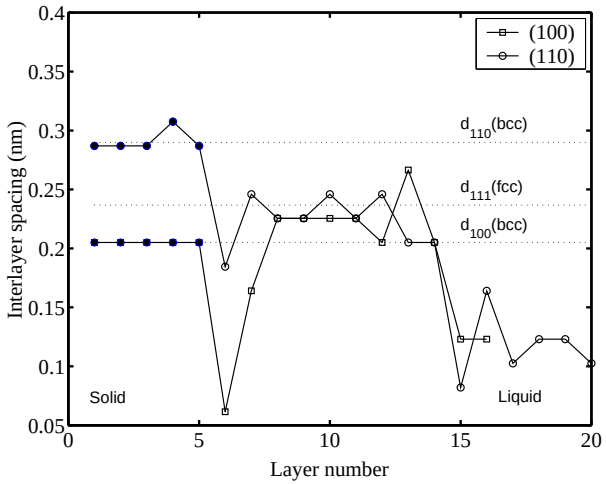


Figure 7.23: The d-spacing for bcc (100) and bcc (110) substrates; in both cases the interlayer spacing inside the liquid is compatible with the periodicity of (111) planes in an fcc aluminium metal.

shows a snapshot of a sample solidified at $T = 600$K, the left panel is a side view, looking down from the [010] direction, and the right panel is a top view looking down the [100] direction. The stacking of the planes shown and the position of the atoms indicates the formation of (100) fcc planes.

A better understanding of the in-plane ordering can be achieved via the two dimensional radial distribution functions calculated in each layer, as shown in the top panel of Figure 7.26. The vertical dotted and dashed lines correspond to the RDFs of the solid bcc(100) and fcc(100) layers, respectively. The RDF of the adsorbed layer, labeled (0) in the figure exhibits a bcc(100) ordering, while the first two "quasi-liquid" layers, labeled (1) and (2) clearly exhibit fcc(100) ordering. The transition to a bulk liquid is seen to occur between layers (3) or (4). This indicates that in-plane ordering is more short lived than the ordering parallel to the interface. This results is consistent with studies of the solid-melt interfaces [12, 36, 37].

On the other hand, the in-plane ordering in the BCC (110) system is much less

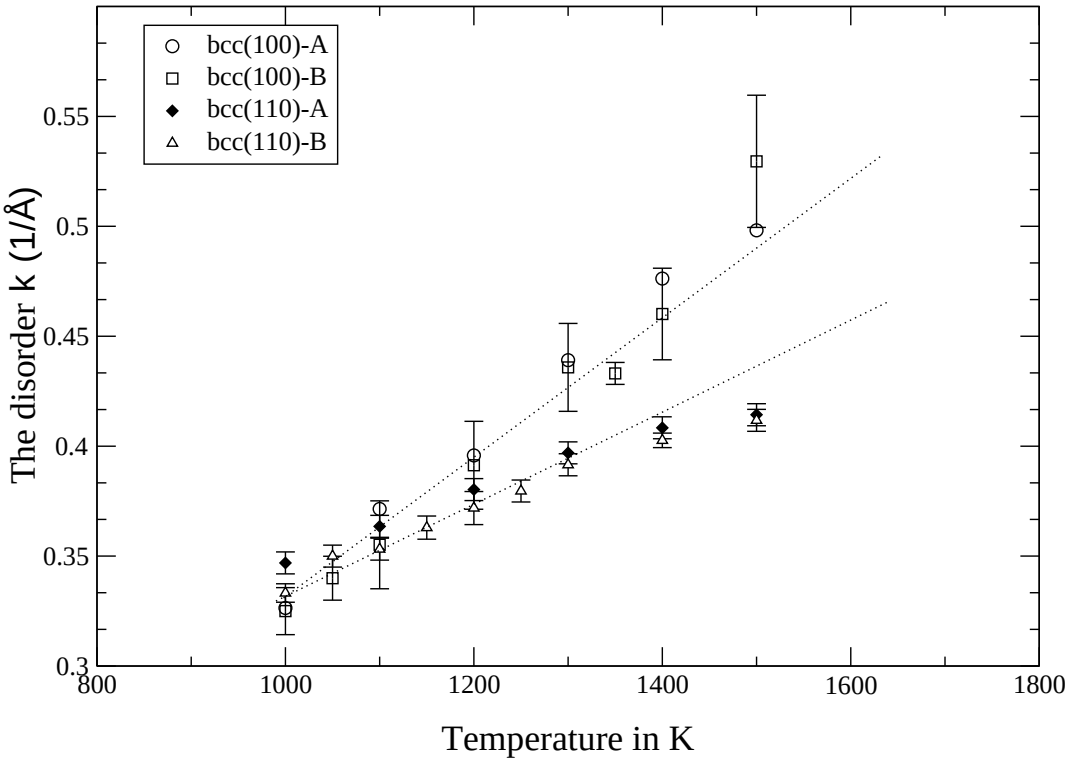


Figure 7.24: The disorder parameter κ for bcc substrates. The results are very similar for both the (110) and (100) substrates at $T \leq 1300K$.

pronounced as can be seen from Figure 7.27, which shows the structure near the interface at 1000K, the last two solid planes (darkest atoms), and the first two liquid layers (medium and light gray). The left panel is a side view, looking down from a $(0\bar{1}1)$ direction, and the right panel is a top view, looking down from the [110] direction. Figure 7.26 shows the RDF for the bcc(110) system at 1000 K. The most remarkable fact is the that RDF of the first "quasi-solid" layer is already liquid-like, in other words there is no preferred in-plane ordering with in these layers, however the ordering normal to the interface is pronounced and extends deep into the liquid, as indicated by the disorder parameter κ (Figure 7.24) and the density profiles (Figure 7.20).

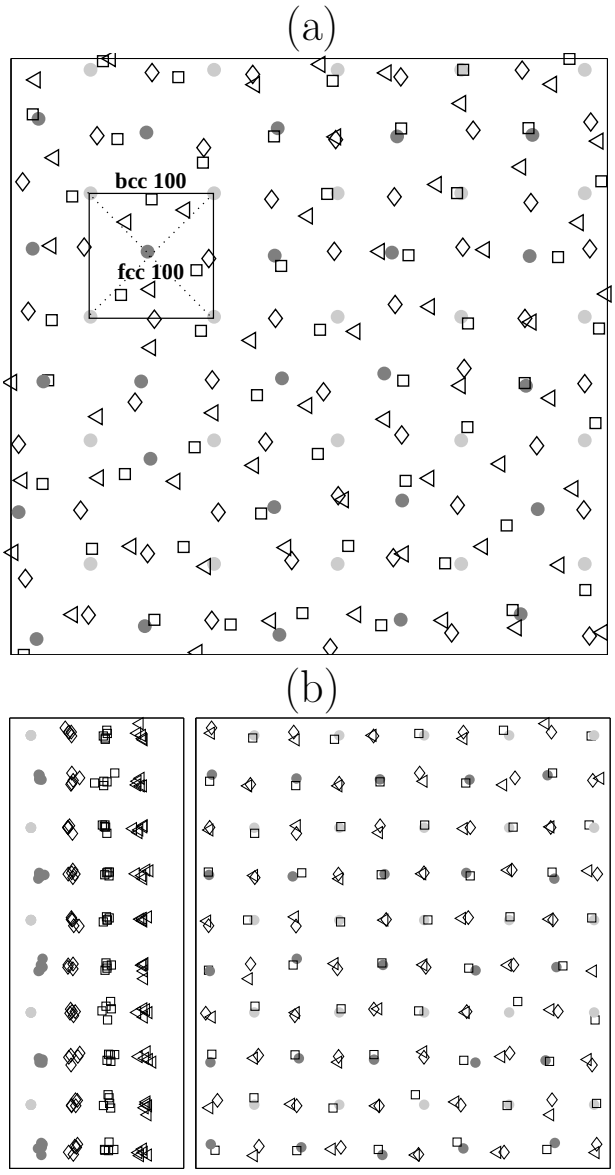


Figure 7.25: (a) Stacking of layers in the bcc (100) system at $T = 1000K$. Light gray circles represent atoms in the terminating bcc (100) plane, dark gray circles the adsorbed layer, diamonds, squares, and triangles represent the first three layers in the liquid. The square connects four atoms of the bcc (100) plane, and the dark circle in the middle indicates the additional site of the fcc (100) plane, occupied by an adsorbed atom. (b) Stacking of layers in a cooled down bcc (100) system at $T = 600K$. Symbols are the same as in (a), but the metallic region has now solidified, forming a bcc (100)/fcc (100) interface.

.

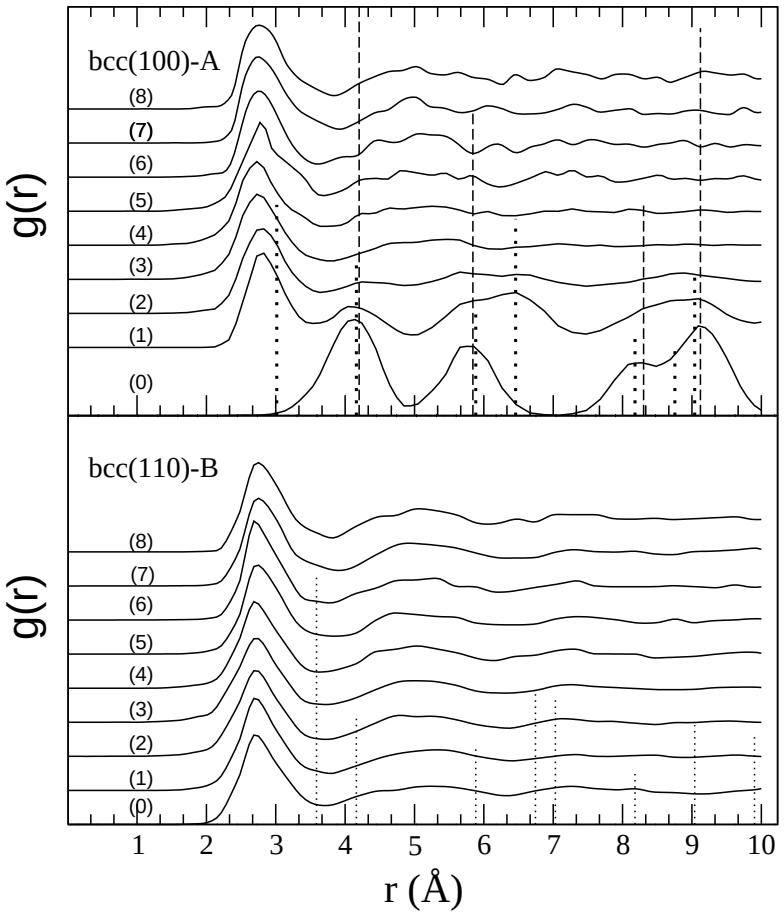


Figure 7.26: The two dimensional radial distribution functions for the bcc(100), top panel, and bcc(110), lower panel, at $T = 1000K$.

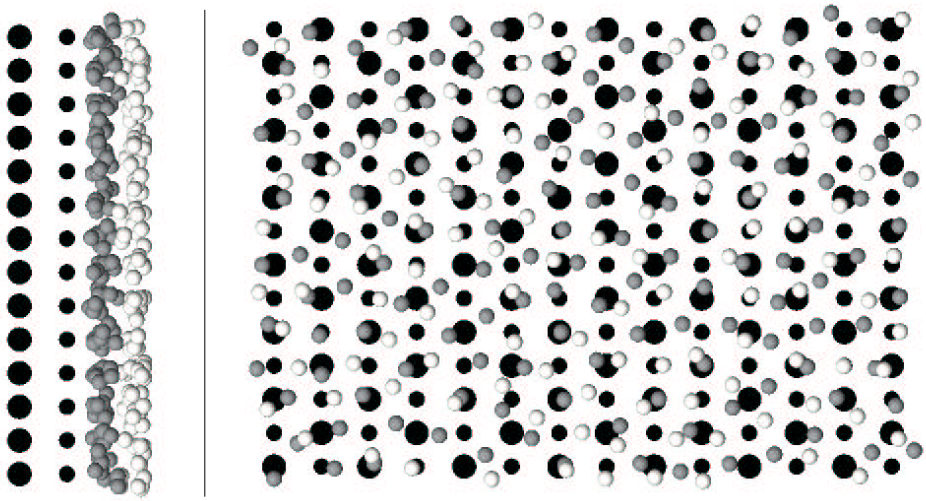


Figure 7.27: The interface region at a bcc (110) substrate at $T = 1000K$. The left panel is a side view and the right panel is a top view. The darkest atoms represent the solid, medium and light gray, the liquid. Different sizes indicate layer position as can be seen from the left panel. In contrast to the (100) system, there is no adsorption of the liquid to the solid substrate.

Chapter 8

Discussion

Two types of interface systems have been investigated. In the first interface system the liquid aluminium is adjacent to an fcc substrate, which although it has a fixed lattice parameter at all temperatures, represents a form which is *compatible* with the interaction potential of the liquid. In the second type of systems, the liquid metal is adjacent to a bcc substrate which is *incompatible* with the interaction potential, both in structure and density. In both systems substantial layering of the liquid was observed, with the extent and amount of the layering strongly dependent on the substrate morphology and on temperature. The large amount of data obtained from the simulations for the different systems, especially those related to the density profiles for the different orientations and different temperatures, necessitated finding a quantitative measure for the ordering exhibited by the density profiles. This led to the introduction of the disorder parameter κ, as described in section 7.1.1.

The disorder parameter κ as a function of temperature for all systems studied is shown in Figure 8.1. This measure indeed proved to be a good quantitative characterization of the amount of ordering in each system, as shown in detail in the preceding chapter. The use of such parameter in the literature in the study of ordering at solid-liquid interfaces is, to the best of our knowledge, non-existent, with one exception to

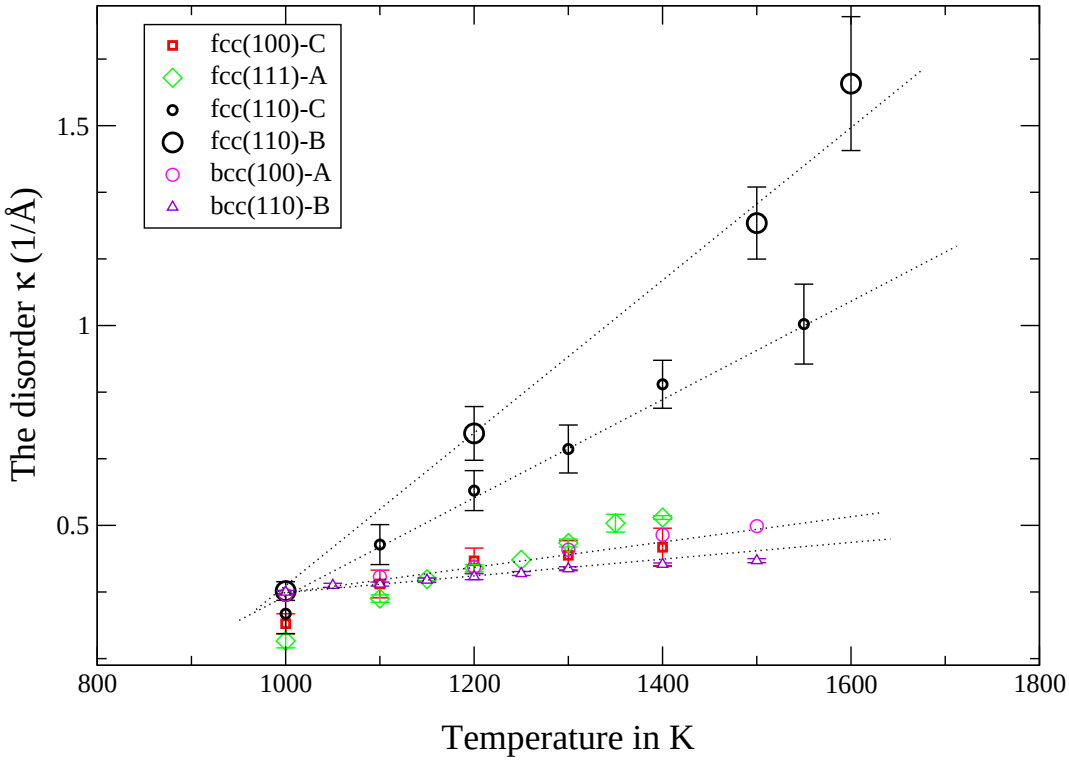


Figure 8.1: The disorder parameter κ for all interfaces studied. The dashed lines are meant to guide the eye, and are not related to the data obtained from the simulations.

be described below, where a parameter similar to κ and related to it was employed.

Figure 8.1 indicates that two different classes of behavior exist for a metal liquid adjacent to a solid substrate. One (the lower curves) is when the substrate is a closed packed solid, such as the fcc(111) and fcc(100) substrates and the bcc(110). Note that in this class the bcc(100) substrate is also included because it is transformed by the adsorption of atoms from the liquid into an fcc(100) like substrate. The second class (the upper curves) is when the metal is adjacent to an open solid substrate, such as the fcc(110) substrate.

In both classes the disorder parameter increases monotonically with temperature, but the increase in the open system is much faster, and almost linear in nature. The

increase in the closed packed systems is noticeably slower. The disorder parameter for the closed packed systems is almost identical within the error bounds of the calculation, while that for the open system deviates substantially, especially at high temperatures.

Another difference is the way the disorder parameter depends on the width of the solid substrate, which as mentioned earlier is a simple way to modify the interaction potential between the substrate and the liquid. In the closed packed systems, the reduction in the width of the substrate did not change the amount of ordering (within the error bounds) while for the open surface a substantial effect was observed, as indicated by the difference in κ for the fcc(110)-B (with a thick substrate) and fcc(110)-C (with a thin substrate) systems, (see also Figure 7.14).

In order to understand these effects, it is important to consider the nature of the interatomic potential. In a metallic system, the potential energy of an atom is dependent on the environment, or more specifically on the density of the surrounding atoms. A metallic bond is created by the sharing of electrons between the atoms. The more atoms this electron is shared with, the lower is the energy of the atom. This is exactly what the embedding part of the EA interatomic potential $U(n)$ represents (see equation 4.6). Indeed, $U(n)$ decreases with increasing atomic density, as shown in Figure 4.1. As a consequence, a liquid atom adjacent to a closed packed surface finds itself in a denser environment than when adjacent to an open surface, and therefore its energy will be lowered. Liquid atoms adjacent to an open surface will have less neighbors from the solid side, this is compensated by increasing the liquid density at the interface, leading to a stronger bound between the liquid and the solid. This explains the larger ordering (smaller κ) for systems fcc(110)-A and C, with the thin substrates, than that for the fcc(110)-B system with the thick substrate. This is also clearly seen in Figure 7.12.

For the closed packed systems, the width of the substrate is screened by the presence of the first layers, in other words, the interaction potential is not affected by the addition of extra layers. This is the reason why the disorder parameter for the bcc(100) systems is very close to that of the fcc(100) system, the transformation of the terminating bcc(100) plane of the substrate to an fcc(100)-like layer is sufficient to induce ordering similar to that obtained for a real fcc(100) system.

The form of the decay of the density profiles with distance from the substrate was studied by Tarazona *et al.* [31] for a spin chain between two smooth walls within a mean field theory. They found that the density decays with a correlation length ξ that depends on temperature in the form:

$$\xi^{-2} = \frac{K_B T}{\rho(1-\rho)\alpha} - 2 \tag{8.1}$$

where ρ is the averaged density in the system, and α is a coupling constant representing the interaction between layers. This means that ξ which is analogous to $1/\kappa$ is proportional to T^2, or that $\kappa \propto \sqrt{T}$ and not linear with temperature as might be deduced from Figure 8.1. However, the two systems studied, that of Tarazona, and that here, are essentially different, so a direct comparison is not entirely justified. In addition the scale of temperature in the mean field treatment is not directly related to the real temperature in the MD system (i.e. the parameter α cannot be related to the current system), so it might be that the disorder parameter is indeed described by Equation 8.1, but on the scale of the figure the feature of the square root function, specifically the decrease in the slop for large arguments is not clearly apparent. Although a closer inspection of κ for the fcc(100)-A and the fcc(111)-B systems in Figure 7.15, and for the bcc(110) systems in Figure 7.24 shows a certain decrease in κ for larger T, this might be attributed to a larger error in the determination of κ at those temperatures, since the number of peaks in the density profiles available for fitting is small (two to three well defined layers). However, both studies predict

that the disorder κ should increase with temperature, but the linear manner is not justified. The complexity of the interaction model used here, prevents a mean field, or density functional method to be applied. Therefore the only result concerning the temperature dependence of κ that can be deduced from this study is that it is an increasing function of temperature.

The phenomena of the layering of the liquid is a well known one, and was observed both in experimental systems [3, 14, 21], as well as theoretical works, especially in the context of crystal-melt systems [17, 35, 36, 38, 42, 46]. Results from these studies indicate that the extent of layering is very similar in the fcc(111), fcc(100), and fcc(110) orientations under the condition of solid-liquid phase coexistence. Nevertheless, no quantitative measure of ordering similar to κ was used in any of these studies. The only reference to a quantity similar to κ was given by Rull *et al.* [43], where the density envelope of a LJ liquid in contact with a fcc(100), fcc(111), and bcc(100) solids at the triple point, was fitted to the form

$$\rho(z) - \rho_l \propto \exp\left[-\alpha z/\Delta z^2\right] \tag{8.2}$$

where ρ_l is the density within the bulk liquid, and Δz is the mean distance between layers, i.e. the mean d-spacing in the quasi-liquid region, and α is a parameter, which Rull *et. al.* argue, based on the simulation data, is universal value for all interfaces. Comparing Equations 8.2 and 7.2, the assumption of universality of α implies that κd^2, where d is the mean d-spacing, should be constant for all systems simulated. Table 8.1 indicates that apart from the fcc(100) and fcc(111) systems at 1000K (the value close to the melting point of aluminium), the values of κd^2 vary substantially with orientation. It should be stressed that in the systems simulated by Rull *et al.* consisted of a solid in equilibrium with the liquid and with an interaction potential that is compatible with both the liquid and solid. The fact that α or κd^2 turns out to be very close for the fcc(111) and fcc(100) systems is not surprising. The planar

densities of the fcc(111) and fcc(100) surfaces is very similar (see Table 5.1). This leads to a similar density in the quasi-liquid layer (as will be explained in more detail in section 8.2), together with the fact that the d-spacing in both systems within the quasi-liquid is similar, it is not surprising that κ for both systems is almost the same at $T = 1000$K, and hence the factor κd^2 turns out to be the same within the error bounds. For the fcc(110) surface, which has a very different planar density and d-spacing, the factor κd^2 does not lead to the same value as for the fcc(111) and fcc(100) systems. Rull *et al.* did not supply data for the fcc(110) systems, and taking into account the small system size they used (840 particles) and the short simulation times (5000 MD steps) which led to large error bars in the calculation of α makes it hard to infer with certainty that α is indeed a universal parameter. The results of the current study suggest that the universality of α, if it exists, does not hold at elevated temperatures and when the system is not in coexisting conditions. Its applicability may be limited to closed packed surfaces with a density comparable to the liquid density, as suggested by the results for the bcc(110) and fcc(110) systems in table 8.1.

The different behavior in κ for the different fcc systems may be related to the ideal formation energies of the surfaces. The close-packed surfaces fcc(100) and fcc(111) in an ideal system have a lower energy than the fcc(110) surface, and therefore are more stable against thermal effects. The formation energies of an *ideal surface*, as calculated with the EA potential, and with LDA [8] are shown in table 8.2. The formation energies of an ideal fcc(111) and fcc(100) surfaces are similar, with the difference between them being only $\sim$ 0.1 eV/atom with the EA potential. The formation energy of the fcc(110) surface is 0.26 eV/atom higher than that of the (100) surface. The similarity of the formation energies of the ideal fcc(111) and fcc(100) systems is compatible with the behavior of κ in Fig. 8.1, where κ for the fcc(111) and fcc(100)

T (K)	k (1/Å)	d (Å)	κd^2
fcc(100)-C			
1000	0.2541	2.149	1.173
1200	0.411	2.225	2.034
1400	0.4452	2.241	2.236
fcc(111)-A			
1000	0.2116	2.336	1.15
1200	0.392	2.34	2.15
1400	0.519	2.33	2.8176
fcc(110)-B			
1000	0.3357	1.39	0.64
1200	0.73	1.38	1.39
bcc(110)-B			
1000	0.3332	2.21	1.627
1200	0.3718	2.21	1.816
1400	0.4026	2.21	1.966

Table 8.1: Calculation of the quantity $\alpha = \kappa d^2$ for various systems used in this study.

systems are almost indistinguishable. This suggest that the energy balance between the different surfaces is not altered by the presence of a liquid phase, although this is not immediately obvious, and explicit calculations have to be made for the different energies of a solid/liquid interface. However, the behavior of κ strongly suggests that the energy balance is not drastically changed.

Fig. 7.11 also shows that the rate of increase of disorder is larger in the fcc [110] direction than for the closed packed surfaces fcc(100) and fcc(111). Again this is

System	EA [eV/Å^2]	[eV/atom]	LDA [eV/Å^2]	[eV/atom]]
(111)	0.054	0.38	0.07	0.48
(100)	0.059	0.48	0.071	0.56
(110)	0.065	0.74	0.08	0.89

Table 8.2: Ideal surface formation energies for aluminium as calculated with the Ercolessi-Adams potential [7] and with LDA [8].

consistent with the lower formation energy of ordered fcc(111) and fcc(100) surfaces. Another striking difference is that as temperature increases the amount of disorder in the closed packed directions seems to saturate while that for the fcc(110) surface seems to increase approximately linearly with temperature. There is even a slight decrease in κ i.e. an increase of ordering, at high temperatures. Note that data for high temperatures ($T > 1400K$) should be analyzed with caution since at these temperatures there are one or two peaks at most and the fitting procedure is no longer accurate.

The disorder parameter κ is shown again for the fcc closed packed systems and the bcc systems in more detail in Figure 8.2. It is clear that the disorder for the fcc(100) and the bcc(100) systems is compatible above 1100 K. At lower temperatures ($T < 1150$ K) the *ordering* in the fcc closed packed systems is larger, at higher temperatures the bcc(110) system shows the greatest ordering. This effect can be understood again based on the nature of the interaction in the liquid metal. The bcc substrates have a lattice parameter of 4.1 Å, the same as for the fcc substrates, however the bulk density of the bcc substrate is half that of the fcc substrate. The liquid atoms adjacent to the bcc substrates are therefore in an environment which is much less dense than what the natural density of the liquid requires. This results in a stronger bound between the liquid and the bcc substrates. This is the same effect described before for the larger ordering in the fcc(110) systems with the thin substrate. At lower temperatures the in-plane ordering in the fcc substrates is larger (this will be discussed more elaborately in section 8.1), this leads to an overall larger ordering perpendicular to the surface, i.e. more order in the liquid planes adjacent to the fcc substrates than in the bcc ones. As the temperature increases this ordering breaks down, and the liquid atoms are less bound to the underlying substrates. For the bcc(110) systems on the other hand, the density of the substrate, being fixed,

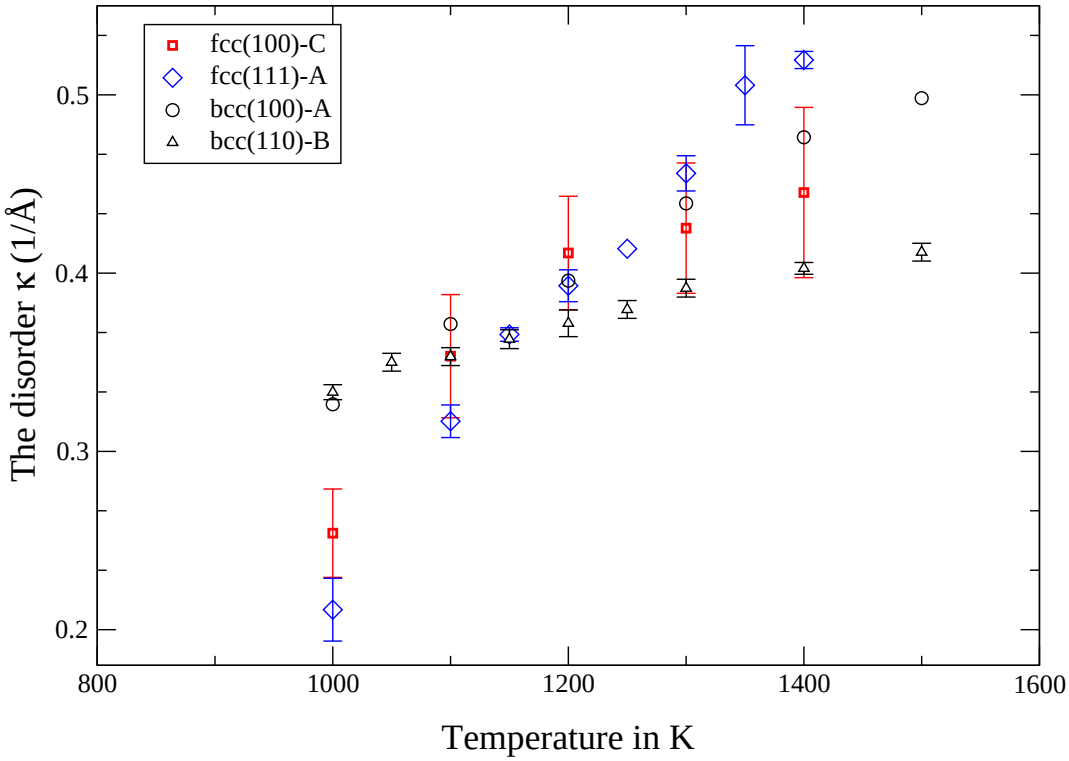


Figure 8.2: The disorder parameter κ for closed packed fcc substrates (fcc(111) and fcc(100)) and for the bcc substrates. The only difference between the current figure and Figure 8.1 is the scale allowing a closer inspection of the differences between the different systems.

will still induce a larger attraction on the liquid, therefore restraining the increase in disorder.

8.1 In-plane ordering

The in-plane ordering within the layers of the quasi-liquid is best demonstrated by the 2D RDF functions (Figures 7.16, 7.17, 7.18, for the fcc substrates, and Figure 7.26 for the bcc substrates). The results from the 2D RDF data reveal that for the fcc systems the structure within the first few layers is compatible with the structure of the adjacent solid substrate. This is not surprising. The field of the solid layer, being

ordered, will produce potential energy minima which are in positions dictated by the lattice structure of the solid. Therefore, in the case where these positions lead to a structure compatible with that of the liquid when it crystallizes, the liquid atoms adjacent to the substrate will assume a structure very close to the ideal lattice. In the case of fcc substrates, this is indeed the case, although the high temperature and the fact that the solid has a fixed lattice parameter leads to more defects in this structure, but on average, as clearly indicated by the 2D RDFs, the liquid atoms in the layers adjacent to the substrate will assume a fcc lattice structure with the proper orientation.

In the case of the bcc(100) substrate, the liquid readily occupies the face-center sites in the terminating bcc(100) plane, changing it to a fcc(100) substrate, and then the consequent layers also assume this type of structure. The extent of in-plane ordering in both the bcc(100) transformed system (not counting the adsorbed atoms layer) is about two layers into the quasi-liquid while that of the fcc(100) is three to four layers. This is due to the fact that in the transformed bcc(100) plane, the atoms sitting in the face-centered positions, i.e. the adsorbed atoms, lie slightly above the terminating bcc(100) plane, and in contrary to the face-center atoms in the fcc(100) substrate, are mobile. This introduces extra disorder into the bcc(100) planes, weakening the field of the substrate, therefore the general order is reduced.

In the bcc(110) case no apparent ordering is present in the liquid layers. There are two factors which might lead to this behavior. First, the bcc(110) close packed structure is highly unfavored by the liquid, that is the locations of minima in the force field of the substrate are not accessible for the liquid atoms which favor to be in an fcc structure. Second, the bcc(110) substrate having a smaller density than the fcc substrate or the liquid density, will produce larger attraction with the liquid atoms adjacent to it, in other words, the liquid is forced to accumulate adjacent to

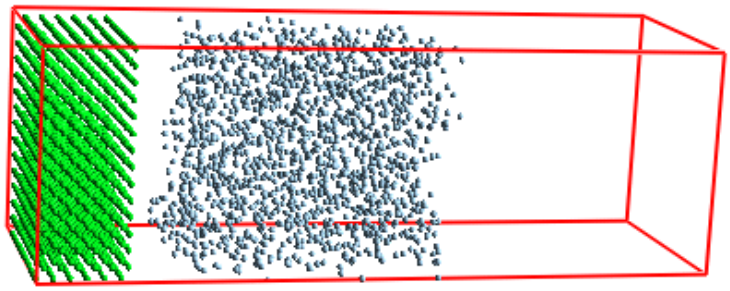


Figure 8.3: The de-wetting of the high density bcc(100) substrate. A snapshot from the simulation on system bcc(100)-C after 20000 MD steps.

the substrate, but without the possibility to create any ordered in-plane structure. This leads to an in-plane structure which is either completely amorphous or liquid like within the planes, as indicated by the 2D RDF, but ordered normal to the plane of the substrate, as indicated by κ.

An interesting effect occurs when the lattice parameter of the bcc substrate is changed so as to eliminate the effect of attraction due to the density. A simple test simulation is performed on system bcc(100)-C, which is designed with a lattice parameter of 2.05 Å, leading to a bulk density of 0.232 atoms/Å^3, which is much larger than the density of the liquid being about 0.051 atoms/Å^3. Figure 8.3 shows that the liquid quickly detaches, or in fact *de-wets* the substrate. In a general context, this indicates that when the substrates density is large, the energy of the solid liquid interface will also be large, and wetting is reduced.

The situation in the bcc(110) system is therefore one in which the liquid wets the substrate but ordering is not induced within the liquid, however this does not mean that ordering perpendicular to the liquid does not necessary exist, as is clearly shown by the layering of the liquid adjacent to these interfaces.

8.2 The interlayer spacing and density variations across the interface

For the closed packed surfaces, fcc(100), fcc(111), and bcc(110), the interlayer spacing switches quickly to a one comparable with the interlayer spacing in the fcc [111] direction. The bcc(100) system, although in principle an open surface, should be included with the closed packed surfaces due to the adsorption effect described above. For the fcc(110) open surface, the d-spacing in the quasi-liquid fluctuates around a value equal to the d-spacing in the direction of fcc [110]. This phenomenon has been observed in a number of solid-liquid interface studies [12, 36, 45], and is is believed to arise from the fact that perturbations within the liquid are more energetically favored if their periodicity is $2\pi/Q_0$, where Q_0 is the wave vector at the first peak of the structure factor $S(Q)$. In a fcc metal, this periodicity is very similar to the d-spacing of planes in the [111] direction.

The results for the fcc systems indicate that the interaction between the ordered solid and the liquid induces layering oscillations within the liquid with a periodicity of the interlayer spacing d. If this periodicity is also commensurate with the natural periodicity of the liquid, as given by $2\pi/Q_0$, then these oscillations are enhanced, and survive a further distance into the liquid. This is the case for the (111) interface. In the case where the d-spacing is very close to $2\pi/Q_0$, such as for the (100) interface, then after some distance (about three layers in Fig. 7.5 for $1000K$, and one layer for the higher T) when the effect of the hard-wall has faded, the d-spacing switches to a value close to $2\pi/Q_0$. When the d-spacing is far from $2\pi/Q_0$, then the oscillations fade away after a short distance.

This suggests that κ for the fcc systems, depends strongly on the d-spacing of the underlying substrate. In fact a reasonable order of magnitude estimate for $\kappa_{\hat{n}}$

as a function of orientation $\hat{n}$, can be obtained by assuming that to first order, the orientation dependence is related only to the d-spacing by:

$$\kappa \sim \frac{1}{d} \tag{8.3}$$

With the value used for the lattice parameter we obtain:

$$\kappa_{111} = 0.433\text{Å}^{-1}, \quad \kappa_{110} = 0.689\text{Å}^{-1}, \quad \text{and} \quad \kappa_{100} = 0.488\text{Å}^{-1}. \tag{8.4}$$

Inspection of Fig. 8.1 shows that this occurs for an intermediate temperature of approximately 1200K. Again we obtain similar κ values for the (111) and (100) surfaces, even when neglecting the jump in d_{100} which would make κ even more comparable.

For the bcc systems, the interlayer spacing in the quasi-liquid and in the substrate are uncorrelated. In the liquid phase the d-spacing is similar to the d-spacing in the fcc closed packed systems, that is in all cases it was very close to the d-spacing of an fcc solid ($d_{111} = 2.367$Å). This effect can be understood by considering the variation of density and number of particles across the interface.

In an ideal lattice, there exists a strict correlation between the inter-planar spacing and the in-plane density in the form of the in-plane density η normalized by the interlayer spacing d:

$$\rho = \eta/d \tag{8.5}$$

where ρ is the bulk density in each layer. Once the density in the layers is fixed, the inter-planar distance has to follow Equation 8.5, with the given bulk density ρ. At the interface region ρ gradually decays across the interface to the density in the bulk liquid.

For a system with an fcc substrate the difference in density across the interface is not large; the density of the fcc substrate being $\rho_{fcc} = 0.058$ atoms/Å^3 (with the lattice parameter of 4.1Å), while that of liquid aluminium within our temperature range is about $\rho_{Al} = 0.052$ atoms/Å^3. Hence there is no substantial change in the

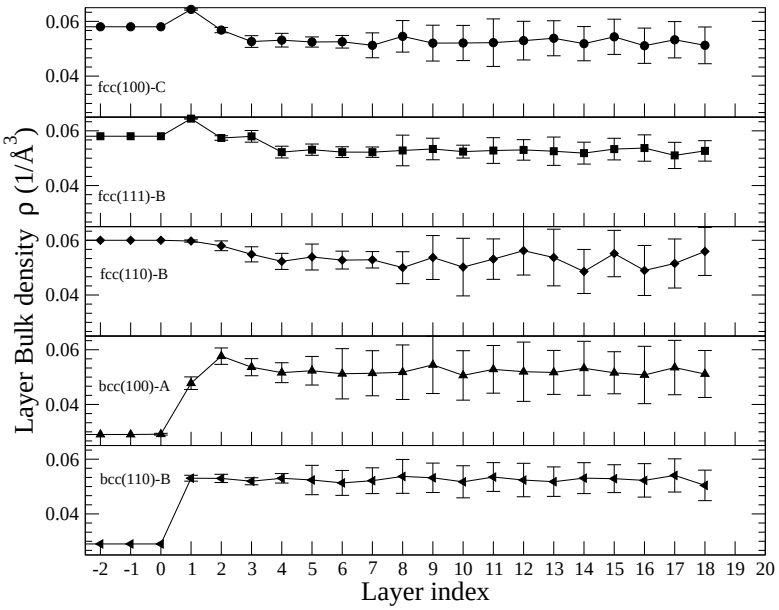


Figure 8.4: The variation of the local layer bulk density at T=1000K. Liquid layers have indices larger than zero.

density of the liquid layers, and consequently there was no major change in the interlayer spacing (see Equation 8.5). Figure 8.4 shows the variation of ρ as calculated from the simulation data for $T = 1000$ K. In contrast to the fcc case, the bcc systems are much less dense than both solid and liquid aluminium (the density of the bcc substrates is $\rho_{bcc} = 0.029$ atoms/Å^3). The liquid near the interface must then accommodate for this change in the density across the interface.

When the liquid metal is adjacent to the solid, there will be an interplay, or competition between the solid and the liquid; the liquid wants to equilibrate to its natural density, while the bcc substrates tend to impose their symmetry on the liquid. In the case of the bcc (100) substrates, the liquid dominates. The terminating plane of the bcc (100) is transformed by the "adsorption of the liquid" into an fcc (100) plane, which is much denser than bcc (100). Indeed this interface behaves similarly to one with a true fcc (100) substrate.

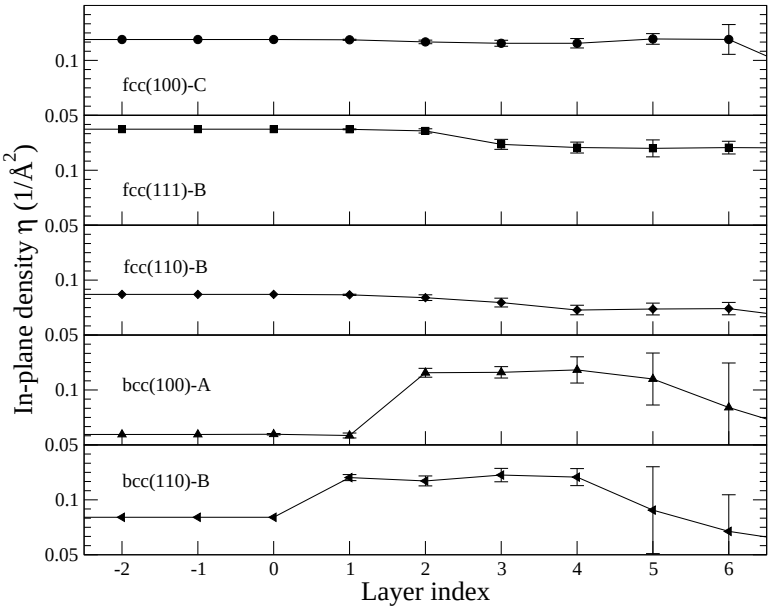


Figure 8.5: The variation of the in-plane density at T=1000K. Liquid layers have indices larger than zero.

In the case of the bcc (110) system the situation is more complicated. Again, the liquid tends to order, but the bcc (110) imposes its in-plane structure, or in effect it induces disorder into the liquid, as described in section 8.1, while the density in the liquid layers is typical for liquid aluminium, see Figure 8.4. The system will reach a certain equilibrium between these two effects.

The average in-plane density in each layer for the all systems is shown in Figure 8.5 at a temperature of $T = 1000$K. The layers with a negative and zero index belong to the perfect lattice, and the rest are the liquid interlayer density. The density of the first layers adjacent to the bcc (110) jumps to a value very close to the density of an ideal fcc (111) or (100) plane. When this density is normalized by the interlayer separation, it is very close to the bulk liquid density, as shown in Figure 8.4. This causes the change in the d-spacing for the (110) system from the bcc (110) value of $d_{110} = 2.8991$Å, to a value close to an fcc (111).

It turns out then that while the liquid is forced to *wet* the bcc (110) substrate by means of the attraction, which is due to the density of the substrate and the interaction nature of the metallic system, it nevertheless imposes its own density. The effect of the substrate induces layering in the liquid, and the combined effect is to produce ordering normal to the interface with the d-spacing governed by the nature of the liquid, and not by the structure of the solid substrate.

An interesting feature of the variation of density across the interface is the mechanism through which this variation occurs. In Figure 8.6 the mean number of atoms in each layer is plotted against the layer index. The mean number of particles in the fcc (100) system is constant within the error bounds, while that of the fcc (111) and fcc (110) changes substantially within the quasi-liquid. On the other hand, the d-spacing in the quasi-liquid region of the fcc (100) system (see Figure 7.5) is shown to expands as the interface is traversed, while the d-spacing of the fcc (111) and fcc(110) systems remains almost constant. This suggests that for the (111) and (110) fcc faces the density decrease associated with the transition from crystal to fluid is achieved by maintaining the spacing between the layers along the direction normal to the interface, while the number of particles in layers decreases. In contrast, for the (100) face, the crystal-liquid transition is achieved by a gradual increase of the interlayer spacing, while the number of particles remains constant. This result is confirmed by a number of studies on crystal melt systems for the fcc (111) and fcc(100) systems [12, 36].

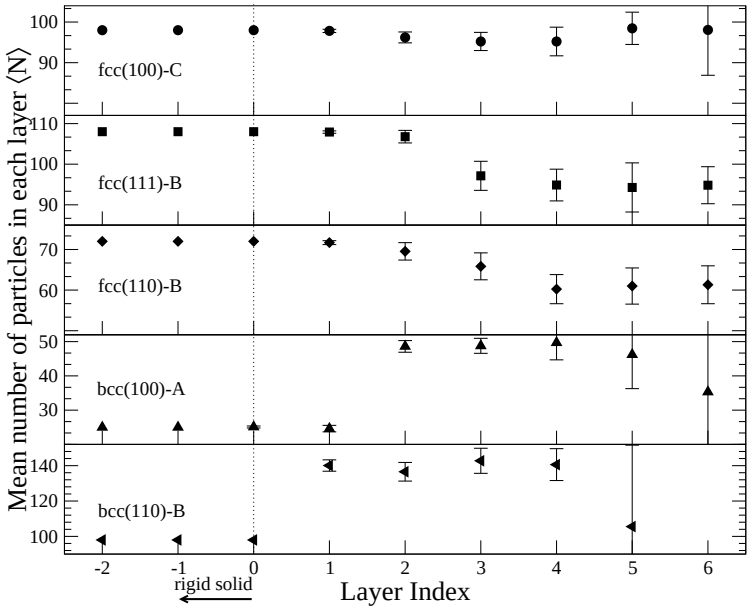


Figure 8.6: The mean number of particles per layer at T=1000K. Liquid layers have indices larger than zero.

Chapter 9

Summary and conclusions

The main purpose of the present research was to examine the effect of the solid substrate on the ordering within the liquid metal in a generic model of a heterophase solid-liquid interface. MD simulations were conducted at temperatures above the melting point of the metal, and with several solid symmetries. The density profile of the liquid phase as a function of temperature and substrate orientation has been studied. The density profiles showed large oscillations corresponding to the layering of the liquid near the interface. The oscillations dampen gradually within the interfacial region according to $\rho(z) \sim \mathrm{e}^{-\kappa z}$, until the density of the liquid phase is reached. The parameter κ was defined as a *disorder* parameter, enabling a quantitative comparison of disorder between the different systems at different temperatures.

Comparing the disorder at interfaces with fcc substrates, indicated that the disorder parameter is largest for the fcc(110) open surface. The amount of disorder is very similar for the closed packed surfaces fcc(100) and fcc(111). The in-plane structure within the layers was studied via the 2D radial distribution function (2D RDF). It is shown that the fcc substrates induce in-plane ordering within the adjacent layers with the same symmetry as the solid substrate. Results for the periodicity of the density oscillations in the liquid indicate a strong correlation with the inter-planar spacing

within the solid. These results suggest that if the solid structure is compatible with the structure of the liquid material upon freezing, the liquid layers at the interface will have substantial in-plane ordering, and the layering periodicity will be that which leads to a conservation of the local bulk density within the layers, which should be equal, or very close to the bulk density of the liquid. The large value of κ for the the (110) fcc system is attributed to the small periodicity of the layering and to the small density of the (110) planes.

For the bcc (100) substrate additional atoms from the liquid phase adsorb into face centered sites in the bcc (100) surface, forming a structure resembling solid fcc (100), but with half the atoms fixed and half slightly displaced above the plane and dynamic. The ordering at this interface was similar to that in the fcc(100) case, with a perturbation due to the displaced atoms in the first plane.

For the bcc (110) surface, the 2D RDF revealed a liquid like structure in all layers at the interface. At the same time the periodicity of the density oscillations was similar to that in the fcc closed packed systems, and in the transformed bcc(100) system. The local layer bulk density is found to be conserved, and equal to the liquid bulk density. This suggests that the liquid plays the dominant part in setting the periodicity in the normal direction, and hence the ordering in that direction.

The analysis of κ indicated that at the lower temperatures ($T < 1200$ K) the interfaces with the fcc closed packed structures are the more ordered, while for higher temperatures the interfaces with the bcc(110) systems are more ordered. This behavior is attributed to the larger attraction which exists between the bcc(110) substrate, being less dense, and the liquid. In other words, the liquid metal wets the bcc(110) substrate better than the others. The turnover in κ between low and high temperatures suggests that the order perpendicular to the interface is affected by the in-plane ordering. At low T, there is more in-plane order at the fcc substrates, than that at the

bcc(110) substrate, which forces κ for the fcc systems to be smaller. As temperature is increased the in-plane order decreases, and the effect of the attraction dominates, so that κ for the bcc(110) becomes larger.

It is concluded also, that when the metal and the solid have incompatible structures, but good wetting exists in the system, the liquid will form layers perpendicular to the interface with a periodicity typical of the natural oscillation period with in the liquid, and with a liquid like in-plane structure. However, when the structure of the solid and the structure of the liquid are compatible, substantial in-plane ordering occurs in the liquid layers.

Appendix A

Density functional method

The density functional theory (DFT), has become in the past few decades an important and powerful tool for handling a many particle system because of its inherent conceptual and computational simplicity and also wide applicability to a large class of problems in the classical and also quantum domains.

In the DFT, the single particle density $\rho(r)$ is used as the basic variable. The grand potential of an inhomogeneous many-particle system is written as a *functional* of $\rho(r)$, which on minimization leads to an Euler-Lagrange equation for obtaining the true equilibrium density distribution. In general, for a single component system under an external potential $V_{ext}(\overrightarrow{r})$, the grand potential functional Ω_v can be written as:

$$\Omega_v[\rho] = F[\rho] + \int dr\rho(r)V_{ext}(r) - \mu \int dr\rho(r) \tag{A.1}$$

where μ is the chemical potential of the liquid. When the density is equal to the equilibrium density $\rho(r) = \rho_0(r)$, then the first two terms in equation(A.1) combine to give the Helmholtz free energy F of the system, while the last term is the Gibbs free energy G. This means that

$$\Omega_v[\rho_0] = F - G = -PV = \Omega \tag{A.2}$$

that is the functional $\Omega_v[\rho]$ when evaluated at the equilibrium density, gives the equilibrium grand potential $-PV$. Here the external potential may be regarded as the effect of an external wall on the fluid. This can be written in functional notation as,

$$\left(\frac{\delta\Omega[\rho]}{\delta\rho(r)}\right)_{\rho_0(r)} = 0 \tag{A.3}$$

$$\Omega_v[\rho_0] = \Omega \tag{A.4}$$

where the equation (A.3) is the fundamental variational principle that permits calculations of equilibrium densities in inhomogeneous systems from approximate functionals $F[\rho]$. Substituting the grand potential functional Ω_v from equation (A.1) into equation (A.3), we obtain the equation

$$\mu_{in}(\rho_0; r) + V_{ext}(r) - \mu = 0 \tag{A.5}$$

where μ_{in} is the *intrinsic chemical potential* defined as a functional derivative of the Helmholtz free energy,

$$\mu_{in} \equiv \frac{\delta F[\rho]}{\delta\rho(r)} \tag{A.6}$$

if $F[\rho]$ is known, then $\rho_0(r)$ can be determined from this equation. For example, consider a system of non-interacting fluid for which the free energy functional is known exactly [16, 17, 30],

$$F_{ideal}[\rho] = \frac{1}{\beta}\int dr\rho(r)[\ln(\lambda^3\rho(r)) - 1] \tag{A.7}$$

where λ is the thermal de Broglie wavelength of the particles, and $\beta = 1/K_BT$. Taking the functional derivative of this equation gives

$$\mu_{in}(\rho_0, r) = \frac{1}{\beta}\ln(\lambda^3\rho(r)) = \mu - V_{ext}(r) \tag{A.8}$$

this equation can be solved to give the equilibrium density of the liquid as

$$\rho_0(r) = z \exp\left[-\beta V_{ext}(r)\right] \tag{A.9}$$

where z is a normalization factor. In other words, the *ordering* in the non-interacting fluid is exponentially dependent on the external potential.

The density functional for a system of interacting particles

In general for the DFT to be useful a knowledge of the functional F in equation A.6 is required. This functional is in most cases unknown, and one has to revert to approximation methods. This first step is to write down the functional as a sum of two parts, one due to the ideal system, and one due to the inter-particle interactions

$$F[\rho] = F_{ideal}[\rho] - \Phi[\rho] \tag{A.10}$$

here the functional $\Phi[\rho]$ includes all the non-ideal contributions to the free energy. The equation for the intrinsic potential then becomes

$$\beta\mu_i[\rho; r] = \ln(\lambda^3\rho(r)) - c(\rho; r) = \beta\mu - \beta V_{ext}(r) \tag{A.11}$$

where $c(\rho; r)$ is the one particle direct correlation function defined by

$$c(\rho; r) \equiv \frac{\delta\Phi}{\delta\rho} \tag{A.12}$$

solving for the equilibrium density gives

$$\rho_0(r) = z \exp(-\beta V_{ext}(r) + c(\rho_0; r)) \tag{A.13}$$

this result indicated that the density of a non homogeneous liquid, for example a liquid near a solid wall, is exponentially dependent on the external potential of the system, and on the inter-particle interactions.

The general problem then transforms to one where Φ is not known in general, however it can be defined through its derivatives with respect to the density, i.e. through the direct correlation function as shown by equation A.12. It also also possible to go to higher derivatives, in fact it is quite common for perturbative DFT studies to use the two particle direct correlation function

$$c(\rho_0; r_1, r_2) = \frac{\delta c(\rho_0, r_1)}{\delta \rho(r_{2)}} = \frac{\delta^2 \Phi}{\delta \rho(r_1) \delta \rho(r_2)} \tag{A.14}$$

The density profile of a non-interacting liquid adjacent to a soft wall

As an example, let us use the previous result for the study of the density profile of a non-interacting fluid adjacent to a soft wall. Here we suppose that the external potential $V_{ext}(r)$ is due to a crystal lattice substrate adjacent to the fluid. There are many ways to represent the interaction of the substrate lattice and the fluid atoms, however for simplicity we shall follow the scheme of Steele [29] and Abraham [56], and assume an effective interaction that depends only on the normal distance of the fluid particle from each crystallographic plane of the substrate, i.e. $V_{ext} \equiv V(z)$, where z is the normal distance. The interaction potential is constructed by replacing the discrete set of centers of interaction in each solid plane of the substrate by a continuous distribution of Lennard-Jones atoms (see section 2.1) with planar density n, the interaction energy between the wall and a fluid atom becomes

$$\phi_w(z) = 2\pi\varepsilon n \sum_{i=1}^{2} [\chi(z_i) - \chi(l_c)] \tag{A.15}$$

where z_i is the perpendicular distance of the fluid atom from the i-th plane of the wall, $l_c = 2.37\sigma$ is the truncation range of the potential, $\varepsilon = 119.8°\text{K}$, and

$$\chi(z_i) = \frac{2}{5} \frac{\sigma^{12}}{z_i^{10}} - \frac{\sigma^6}{z_i^4} \tag{A.16}$$

following [56] we choose $\rho\sigma^3 = 1.01$ for the density of the substrate, leading to a surface density of $n = 0.8/\sigma^2$, and a d-spacing of the substrate $d = 2.69\text{Å}$. In Figure A.1

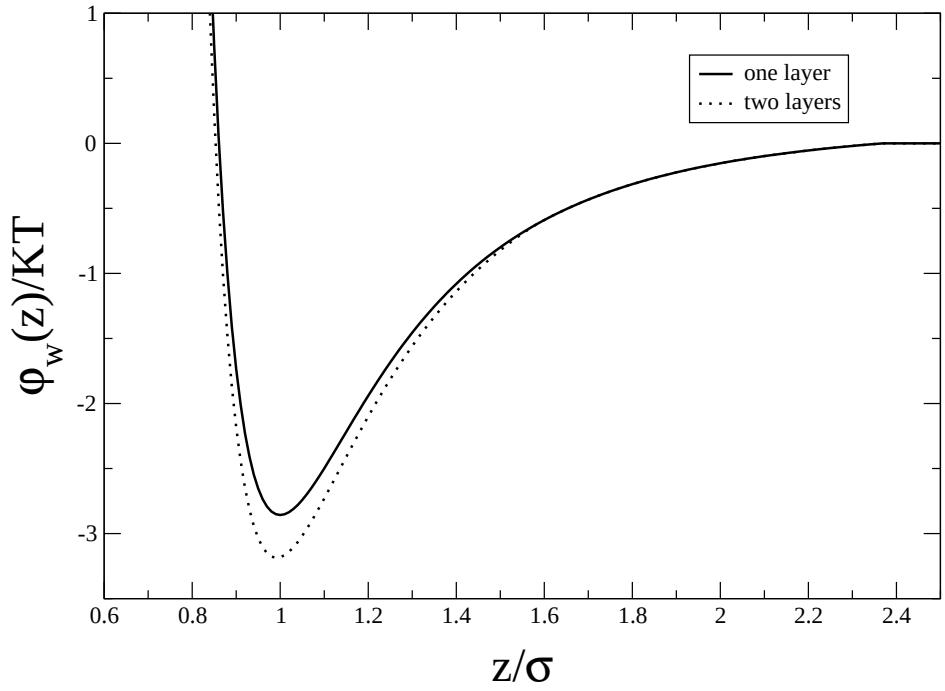


Figure A.1: Atom/solid surface interaction energy according to the Lennard-Jones 10-4 potential, two cases are shown, (1) for one crystallographic plane, and (2) for two crystallographic planes.

the interaction potential is displayed for one and two layers of the substrate. The resulting density profiles, corresponding to one and two layers of the substrate are shown in Figure A.2. It is interesting to note the formation of one liquid layer near the interface. If the interaction between the atoms is turned on then, this liquid layer may be regarded as a new "wall" for the rest of the atoms, leading to a second layer, etc.

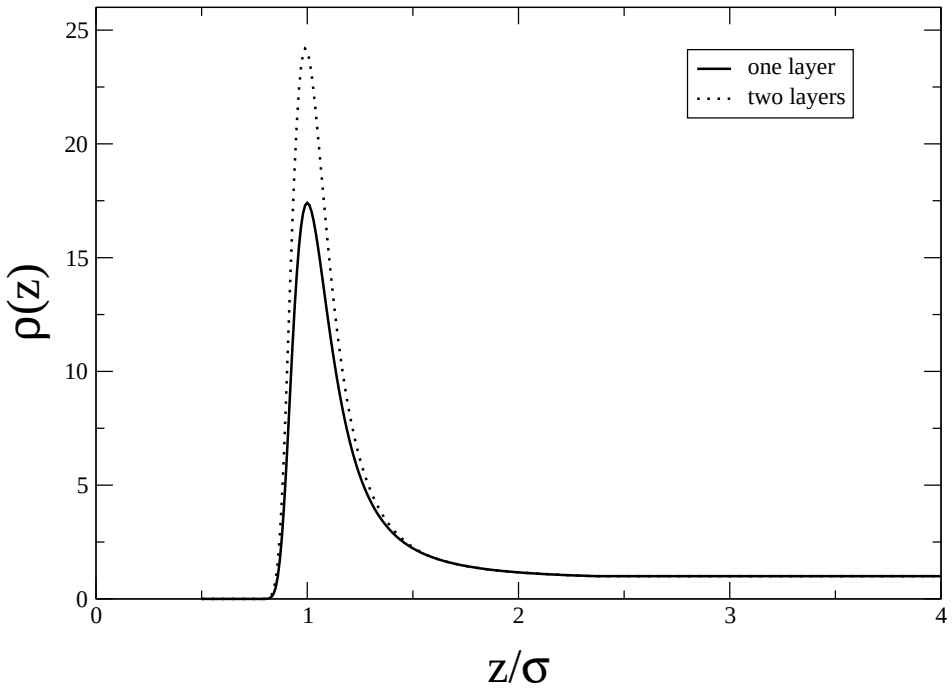


Figure A.2: The equilibrium density profile for a non-interacting fluid adjacent to a soft 10-4 LJ wall. Note that the peaks is higher for a two layered substrate, indicating a stronger interaction with the substrate.

Appendix B

Description of the simulation programs

The source code of the molecular dynamics simulation programs is written using the ANSI C programming language. The complete source code of the main programs are given below. The main code is in two files: `MD.c` which contains most of the code relevant to the MD algorithm, and main input/output functions, and `cva.c` which contains source code of some general low level input/output functions and provides some general utility functions. The potential functions are defined in an external Fortran 77 file `aluminium.f`, which was provided by Dr. Furio Ercolessi (see `http://www.fisica.uniud.it/~ercolessi/`). In addition, use has been made of the `nrutil.c` package from the Numerical recipes book [57], this was mainly to produce arrays in C which have 1 as their starting index instead of 0. However, some arrays in the code were defined in the conventional C standard.

cva2 file	corresponding xyz file
2450 4.920000E+01 2.870000E+01 2.870000E+01 1280	2450
r_1T-N0020000.xyz	comment line
0.000000 0.000000 0.000000	S -5.125000 -13.325000 -11.275000
0.000000 0.000000 0.000000	S -5.125000 -13.325000 -7.175000
...	...
-0.549737 0.188965 -0.045804	L 51.515786 2.941825 1.814855
-0.369673 1.074560 -0.145726	L 48.987962 10.689944 2.793810

Figure B.1: An example of a cva2 and xyz files used for storing coordinates, velocities and accelerations. In this case solid atoms are denoted by 'S', and liquid atoms by 'L'.

Compilation

The major part of this work was done on Linux workstations with the GNU C and Fortran compilers, gcc and g77, respectively. Incorporating Fortran and C code in this system is straightforward, it consists of first compiling the Fortran files using the compile only option of g77, i.e. with out producing an executable: `g77 -o aluminium.f`. This produces an object file named `aluminium.o`. The MD executable is then produced using the following command:

```
gcc -O3 MD.c -o MD nrutil.c aluminium.o cva.c -lm
```

The resulting executable is then MD, the next step is to prepare a control file containing the various variables of the simulation, such as the starting configuration, the temperature, the type of temperature control or thermostat, the number of steps, and so on.

Input of initial sample

The samples consist of the coordinates, and if available, the velocities and accelerations. The data is stored in two files (see Figure B.1 for an example). The first is the `cva2` file, which consists of a one line header containing the number of atoms and the dimensions of the simulation cell in the x, y, and z directions. The second line consists

of the name of the `xyz` file containing the coordinates. If velocities and acceleration are given (usually from a previous run) they are listed starting from the third line, where the velocities of all atoms are listed first followed by the velocities. The `xyz` file in its simplest form contains the number of atoms on the first line, followed by a mandatory comment line. Then the type and position of each atom are listed line by line, see Figure B.1. The `xyz` file format is particularly convenient when visualizing the data, since most visualization packages can read this format. In this work the atomistic visualization package `AViz`, developed in the Computational Physics group at the Technion was used, (see `http://phycomp.technion.ac.il/`).

The input control file

The MD program accepts a control file which determines all the relevant parameters of the simulation. The file should be formatted in two columns, where the first column gives the name of the variable (with a single string without spaces) and the second column is the value of the variable. Each line which starts with a # sign is ignored by the `MD` program and is considered as a comment. An example of a control file is shown in Figure B.2, a description of the various input variables in the control file is given below:

- `TITLE:` title of the project.
- `cva2FileName:` the initial cva2 file (see description in section B)
- `run-id:` and identifier added to all output file names.
- `Trequested:` the requested temperature in K, used for rescaling.
- `Box-Factor-x:` sets the extent of the vacuum region in the x-direction above the free solid surface, in units of the simulation cell size. Set to 1 for periodic

```
#----------------------------
title                    R1
cva2FileName             fcc100.cva2
run-id                   r_1T
Trequested               1300
Box-Factor-x             25
Box-Factor-y             1
Box-Factor-z             1
#----------------------------
# SAMPLING:
#----------------------------
Deltat                   0.02
MDsteps                  20000
roll-interval            5000
write-snap-interval      20000
#----------------------------
# SLICE ANALYSIS CONTROL:
#----------------------------
fss-file                 NULL
slice-width              0.205
sla-roll-interval        20000
begin-stat-sla           0
#----------------------------
#THERMOSTAT = 0, no thermostat (const E), 1 =rescaling
#----------------------------
thermostat               1
scale-interval           100
scale-end                1000000
#----------------------------
# VELOCITY-VERLET AND NEIGHBOUR LIST PARAMETERS
#----------------------------
# the skin is in units of Angstroms.
Skin                     2.0
Iseed                    811197023
#----------------------------> END OF CONTROL FILE.
```

Figure B.2: An example of the input control file of the simulation program.

boundary conditions.

- `Box-Factor-y:` same for the y-direction.
- `Box-Factor-z:` same for the z-direction.
- `Deltat:` the integration time step in units of the real time $t = 5.288 \times 10^{-14}$ seconds.
- `MDsteps:` the total number of integration steps.
- `roll-interval:` the interval in MD steps on which different properties are averaged and then output.
- `write-snap-interval:` the interval in MD steps between outputs of the coordinates, velocities, and accelerations.
- `fss-file:` either `NULL` or a name of a slice data file should be given (usually from a previous run). If `NULL` a new slicing is produced, otherwise a slicing is read from the provided file name.
- `slice-width:` in case `fss-file` is `NULL` the new slices will have a width of `slice-width`, should be in Å.
- `sla-roll-interval:` the interval in MD steps over which slice data is averaged and output.
- `begin-stat-sla:` sets the number of MD steps before slice analysis is initially invoked.
- `thermostat:` either 0 (constant energy run), or 1 for temperature rescaling.
- `scale-interval:` the interval in MD steps between consecutive rescaling of the temperatures.

- `scale-end:` when to stop temperature rescaling, after which constant energy run proceeds.
- `Skin:` the extra padding added to the potential cutoff, used for the neighbor lists.
- `Iseed:` the seed used to initialize the random number generator, used to set initial velocities if not provided in the cva2 data file.

The output:

Several data files are output, all are preceded with the `run-id` identifier, for example with the control file given in Figure B.2, among the output files will be: `r_1T.ET` giving the energy and temperature per particle, `r_1T.Vcm` giving the velocity of the center of mass, `r_1T.fss1` the slice data file containing position, width, and average number of particles in each slice, `r_1T.fss2` another slice data file containing the temperature and energy in each slice, and `r_1T.fss3` the static structure order parameter in each slice. In addition a report file, `r_1T.report`, containing information on input and output parameters and a summary of the actions performed during the run is given. The output of the positions is given in a cva2 and an xyz files, which contain the time stamp in the file name, such as: `r_1T-N0020000.cva2` and `r_1T-N0020000.xyz`.

B.1 Source code

MD.c

```
/*
   Ercolessi-Adams Glue potetnial for Al (see Europhys. Lett. 26(8):583-588, 1994). The potential is loaded in
function "Define_Potential_Tables" using external Fortran routines provided by Furio Ercolessi. (see http://www.
sica.uniud.it/~ercolessi/) Adham Hashibon, phadham@technion.ac.il Last Accessed: Tuesday 23 April 2002

*/
#include "MD.h"
int
```

```
main (int argc, char **argv)
{
  extern char *optarg;
  extern int optind;
  int errflg = 0;
  int i;
  void (*algor) (int);
  char control_filename[80];
  double fac;
  if (argc < 2)
    usage ();
  if (argv[optind] == 0)
    usage ();
  else
    strcpy (control_filename, argv[optind++]);

  Initialize_CVA ();
  Read_Control (control_filename);
  Report_Part_1 (control_filename);
  /* Load Model */
  Init_Model ();
  /* monitor the simulation cell size */
  scell = Get_Box ();
  Define_Aluminium_Potential_Tables ();
  Update_NeighborList (RcutoffPlusSkin);
  Set_Evolution_Control ();
  Stat_Sla (0);
  Compute_Forces ();
  Stat ();

  /*** Start evolution: **/
  for (i = 0; i < MDsteps; i++)
    {
      switch (thermostat)
        {
        case CONST_E:
          VVEA_E (1);
          break;
        case RESCALE_T:
          if (ts_counter % scale_interval == 0)
            VVEA_T (1);
          else
            VVEA_E (1);
          break;
        default:
          report ("NO THERMOSTAT IS GIVEN, ABORTING");
          error ("NO THERMOSTAT IS GIVEN, ABORTING");
          break;
        }

      Stat ();
      Stat_Sla (1);
      if (ts_counter % write_snap_interval == 0)
        WriteSnapShot_CVA2 ();
    } /* MDsteps */

  /* Average over many instances, each averaged over roll-interval samples. */
  Average (&GEppp);
  Average (&GEkpp);
  Average (&GTkpp);
  Average (&GEtot);
  Average (&GEktot);
  /* avoiding the devision */
  fac = 1. / sqrt (GEppp.count);

  fprintf (ET_file,
           "# total avg over %d samples=\n#% 10d % 10f % 10e",
           GEppp.count,
           ts_counter + ts_init, GTkpp.mean / k_B, GTkpp.var / k_B * fac);

  fprintf (ET_file, " % 10e % 10e % 10e % 10e",
           GEppp.mean, GEppp.var * fac, GEkpp.mean, GEkpp.var * fac);

  fprintf (ET_file, " % 10e % 10e % 10e % 10e",
           GEtot.mean, GEtot.var * fac, GEktot.mean, GEktot.var * fac);
```

```
  for (i = 1; i < 4; i++)
    Average (&GVcm[i]);

  fprintf (Vcm_file,
           "# total avg over %d samples=\n#% 10d % 10e % 5e",
           GEppp.count,
           ts_counter + ts_init,
           GVcm[1].mean, GVcm[1].var / sqrt (GEppp.count));

  fprintf (Vcm_file, "% 10e % 5e % 10e % 5e\n",
           GVcm[2].mean,
           GVcm[2].var / sqrt (GEppp.count),
           GVcm[3].mean, GVcm[3].var / sqrt (GEppp.count));

  for (i = 0; i < 3; i++)
    {
      Average (&Gurc[i]);
      Average (&Gllc[i]);
    }
  fprintf (Box_file, "# total avg over %d samples=\n", Gurc[0].count);
  fprintf (Box_file, "% 10d ", ts_counter + ts_init);
  for (i = 0; i < 3; i++)
    fprintf (Box_file, "% .5f % .3e ",
             Gllc[i].mean, Gllc[i].var / sqrt (Gurc[0].count));
  for (i = 0; i < 3; i++)
    fprintf (Box_file, "% .5f % .3e ",
             Gurc[i].mean, Gurc[i].var / sqrt (Gurc[0].count));
  fprintf (Box_file, "\n");

} /* main */

void
WriteSnapShot_CVA2 ()
{
  int i, j;
  FILE *xyzFile, cva2File;
  char xyzFileName[200], cva2FIleName[200];

  char NsnapChar[80], NTchar[80];
  char snap_filename[200];
  char rems[10];

  /* cva stuff */
  struct CVA_BUFFER buff;

  strcpy (snap_filename, run_id);
  sprintf (NsnapChar, "%07d", ts_counter + ts_init);
  sprintf (NTchar, "%03.0f", Trequested / k_B);

  strcat (snap_filename, "-N");
  strcat (snap_filename, NsnapChar);

  Init_CVA_Buffer (&buff);

  buff.H.tstamp = ts_counter + ts_init;
  buff.H.natoms = Natoms;
  buff.H.velac = TRUE;
  strcpy (buff.H.fname, snap_filename);

  buff.pos = (double **) malloc (sizeof (double *) * Natoms);
  for (i = 0; i < Natoms; i++)
    buff.pos[i] = (double *) malloc (sizeof (double) * 3);

  buff.vel = (double **) malloc (sizeof (double *) * Natoms);
  for (i = 0; i < Natoms; i++)
    buff.vel[i] = (double *) malloc (sizeof (double) * 3);

  buff.acc = (double **) malloc (sizeof (double *) * Natoms);
  for (i = 0; i < Natoms; i++)
    buff.acc[i] = (double *) malloc (sizeof (double) * 3);

  buff.atype = (int *) malloc (Natoms * sizeof (int));

  for (i = 0; i < Natoms; i++)
    {
      buff.atype[i] = AtomType (Atype[i + 1]);
      buff.pos[i][0] = pos[i + 1][0 + 1];
```

```
      buff.pos[i][1] = pos[i + 1][1 + 1];
      buff.pos[i][2] = pos[i + 1][2 + 1];

      buff.vel[i][0] = vel[i + 1][0 + 1];
      buff.vel[i][1] = vel[i + 1][1 + 1];
      buff.vel[i][2] = vel[i + 1][2 + 1];

      buff.acc[i][0] = acc[i + 1][0 + 1];
      buff.acc[i][1] = acc[i + 1][1 + 1];
      buff.acc[i][2] = acc[i + 1][2 + 1];
    }

  strcpy (buff.H.vacc, "T");
  buff.H.bbox = Box_CVA_Buffer (&buff);
  /* store the original BoxSize */
  buff.H.box.x = BoxSizeBackup[1];
  buff.H.box.y = BoxSizeBackup[2];
  buff.H.box.z = BoxSizeBackup[3];
  Save_CVA2XYZ_to_File (&buff);
}

void
Null_Vel (int *alist, int n)
{
  int i, k, j;
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        vel[i][k] = 0.0;
    }
  report ("%40s", "Velocities Nulled for solid part");
}

void
Init_velocity (int *alist, int n)
{
  int i, j, k;
  double g[2], v_mean;
  /* center the velocity arround zero */

  v_mean = 0.0;
  iseed = Get_i_Variable ("Iseed");
  fprintf (stderr, "Iseed=%d\n", iseed);
  report ("initializing velocities with seed = %d", iseed);
  srand48 (iseed);
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      gran (g);
      /* get 2 gaussian random numbers */
      vel[i][1] = v_mean + g[0] * sqrt (Trequested);
      vel[i][2] = v_mean + g[1] * sqrt (Trequested);
      gran (g);
      /* get another 2 gaussian random numbers */
      /* choose one of the numbers, this is extra! but .... */
      k = (drand48 () < 0.5) ? 0 : 1;
      vel[i][3] = v_mean + g[k] * sqrt (Trequested);
    }
  TransVel2CM (llist, Nliquid);
}

void
gran (double *g)
{
  /* return an array of two gaussian distributed random numbers in each call.
     The Box and Muller Method is used, see Allen & Tildesley book, app. G */

  double x1, x2;

  x1 = drand48 ();
  x2 = drand48 ();
  (*g) = sqrt (-2.0 * log (x1)) * cos (2 * PI * x2);
  *(g + 1) = sqrt (-2.0 * log (x1)) * sin (2 * PI * x2);
```

```
}

void
TransPos2CM (void)
{
  /* translate Pos to Center of Mass system */

  int i, k;
  double PosCM[4];
  for (k = 1; k <= 3; k++)
    PosCM[k] = 0.0;
  for (i = 1; i <= Natoms; i++)
    {
      PosCM[1] += pos[i][1];
      PosCM[2] += pos[i][2];
      PosCM[3] += pos[i][3];
    }

  for (k = 1; k <= 3; k++)
    PosCM[k] = PosCM[k] / (double) Natoms;
  report ("POS CM: %10g %10g %10g", PosCM[1], PosCM[2], PosCM[3]);
  for (i = 1; i <= Natoms; i++)
    {
      pos[i][1] -= PosCM[1];
      pos[i][2] -= PosCM[2];
      pos[i][3] -= PosCM[3];
    }
}

void
TransVel2CM (int *alist, int n)
{
  double velcm[4];
  auto int i, j, k;

  CalVelCM (alist, n, velcm);
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        {
          vel[i][k] -= velcm[k];
        }
    }
}

void
CalVelCM (int *alist, int n, double *velcm)
{
  int i, j, k;

  for (k = 1; k <= 3; k++)
    velcm[k] = 0.0;
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        velcm[k] += vel[i][k];
    }
  for (k = 1; k <= 3; k++)
    velcm[k] = velcm[k] / (double) (n);
}

void
usage (void)
{
  pr_stderr ("Usage:  MD control-file");
  exit (0);
}

void
pr_stderr (char *fmt, ...)
{
  va_list args;
```

```
   va_start (args, fmt);
   vfprintf (stderr, fmt, args);
   fprintf (stderr, "\n");
   va_end (args);
}

void
report (char *fmt, ...)
{
   va_list args;
   va_start (args, fmt);
   fprintf (report_file, "# ");
   vfprintf (report_file, fmt, args);
   fprintf (report_file, "\n");
   va_end (args);
}

void
Read_vel (FILE * init_conf_file)
{
   int i;
   char srem[10];

   for (i = 1; i <= Natoms; i++)
      fscanf (init_conf_file, "%s %lf %lf %lf",
              srem, &vel[i][1], &vel[i][2], &vel[i][3]);
}

void
Read_acc (FILE * init_conf_file)
{
   int i;
   char srem[10];
   for (i = 1; i <= Natoms; i++)
      fscanf (init_conf_file, "%s %lf %lf %lf",
              srem, &acc[i][1], &acc[i][2], &acc[i][3]);
}

void
Define_Aluminium_Potential_Tables (void)
{
   /* Compute (once for all) tables for the glue potential for Al. */
   auto double Rsq, r, phi, dphi, d2phi, rho, drho,
      d2rho, c, glue, dglue, d2glue;
   int k;

   report (" ");
   report ("%40s", "Invoking Aluminium Define_Potential_Tables");
   Rcutoff = 0.555805441821810e+01;
   RcutoffPlusSkin = Rcutoff + Skin;
   RcutoffSqr = Rcutoff * Rcutoff;
   Rmin = 0.055805441821810e+01; /* ! cutoff-5 A */
   RsqMin = Rmin * Rmin;
   DeltaRsq = (Rcutoff * Rcutoff - RsqMin) / (TableSize - 1);
   InvDeltaRsq = 1.0 / DeltaRsq;
   CoordMax = 1.20;
   DeltaCoord = CoordMax / (TableSize - 1);
   InvDeltaCoord = 1.0 / DeltaCoord;

   PhiTab = dvector (1, TableSize);
   DPhiTab = dvector (1, TableSize);
   RhoTab = dvector (1, TableSize);
   DRhoTab = dvector (1, TableSize);
   UTab = dvector (1, TableSize);
   DUTab = dvector (1, TableSize);

   for (k = 1; k <= TableSize; k++)
      {
         Rsq = RsqMin + (k - 1) * DeltaRsq;
         r = sqrt (Rsq);
         v2_ (&r, &phi, &dphi, &d2phi);
         rh_ (&r, &rho, &drho, &d2rho);
         PhiTab[k] = phi;
         DPhiTab[k] = -dphi / r;
         RhoTab[k] = rho;
```

```
      DRhoTab[k] = -drho / r;
    }

  for (k = 1; k <= TableSize; k++)
    {
      c = (k - 1) * DeltaCoord;
      uu_ (&c, &glue, &dglue, &d2glue);
      UTab[k] = glue;
      DUTab[k] = dglue;
    }
}

void
Update_NeighborList (double Range)
{
  /*
     Update the neighbor list, including all atom pairs that are within a
     distance Range to each other. Range may be equal to the interaction range
     but is usually made somewhat bigger to have the list remain valid for a
     while.

     The Fincham-Ralston method is used (D. Fincham and B.J. Ralston,
     Comp.Phys.Comm. 23, 127 (1981) ), which allows vectorization of the first
     of the two inner loop on vector computers and therefore gives good
     performance. */

  double RangeSq, Sij[4], Rsqij;
  int ListLength;
  int i, j, L, k;
  double posi[4];

  RangeSq = Range * Range;

  /* The following is the Fincham-Ralston loop for list updating */

  /* initialize neighbor index in 'List' */
  L = 1;
  for (i = 1; i <= Natoms; i++)
    {
      for (k = 1; k <= 3; k++)
        posi[k] = pos[i][k];
      for (j = i + 1; j <= Natoms; j++)
        {
          Dist3v (posi, pos[j], Sij);
          Rsqij = Dot_Product_3v (Sij, Sij);
          Advance[j] = (Rsqij < RangeSq) ? 1 : 0;
        } /* j */

      /* pointer to the position in 'List' of the first neighbor of atom 'i' */
      Marker1[i] = L;
      for (j = i + 1; j <= Natoms; j++)
        {
          List[L] = j;
          L += Advance[j];
        } /* j */
      /* pointer to the position in 'list' of the last neighbor of atom 'i',
          note that the neighbors of atom 'i' have Marker1[i] =< L < Marker2[i] */
      Marker2[i] = L - 1;
    }
  ListLength = L - 1;
  /* final used length of List */

  for (i = 1; i <= Natoms; i++)
    {
      /* ! reset displacement vectors */
      DisplaceList[i][1] = 0;
      DisplaceList[i][2] = 0;
      DisplaceList[i][3] = 0;
    }
  updateList = FALSE;
}

void
Refold (double *v)
{
```

```
  int k;
  for (k = 1; k <= 3; k++)
    v[k] -= BoxSize[k] * rint (v[k] * InvBoxSize[k]);
}

double
Sum1v (double *v, int *alist, int n)
{
  int i, k, j;
  double vsum = 0.0;

  for (i = 0; i < n; i++)
    vsum += v[alist[i]];
  return (vsum);
}

void
Dist3v (double *v1, double *v2, double *dist)
{
  int k;
  for (k = 1; k <= 3; k++)
    dist[k] = v1[k] - v2[k];
  Refold (dist);
}

double
Dot_Product_3v (double *v1, double *v2)
{
  double res;

  res = v1[1] * v2[1] + v1[2] * v2[2] + v1[3] * v2[3];
  return (res);
}

void
VVEA_E (int n)
{
  int i, t;
  for (t = 0; t < n; t++)
    {
      for (i = 1; i <= Natoms; i++)
        Refold (pos[i]);
      VV_Const_E_step_1 (llist, Nliquid);
      Compute_Forces ();
      VV_step_2 (llist, Nliquid);
      Update_DisplaceList ();
      updateList = MovedTooMuch ();
      if (updateList == TRUE)
        Update_NeighborList (RcutoffPlusSkin);
      ts_counter++;
    }
}

void
VVEA_T (int n)
{
  int i, t;
  for (t = 0; t < n; t++)
    {
      for (i = 1; i <= Natoms; i++)
        Refold (pos[i]);
      VV_Const_T_step_1 (llist, Nliquid, Trequested);
      Compute_Forces ();
      VV_step_2 (llist, Nliquid);
      Update_DisplaceList ();
      updateList = MovedTooMuch ();
      if (updateList == TRUE)
        Update_NeighborList (RcutoffPlusSkin);
      ts_counter++;
    }

}

void
VV_Const_T_step_1 (int *alist, int n, double T)
{
```

```
  int i, j, k;
  double current_T, chi, Ek;
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        deltar[i][k] = Deltat * vel[i][k] + halfDeltatSqr * acc[i][k];
    }
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        pos[i][k] += deltar[i][k];
    }
  Compute_Ek_T_Vcm (alist, n, &Ek, &current_T);
  chi = sqrt (T / current_T);
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        vel[i][k] = chi * vel[i][k] + halfDeltat * acc[i][k];
    }
}

void
VV_Const_E_step_1 (int *alist, int n)
{
  int j, i, k;
  double chi;
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        deltar[i][k] = Deltat * vel[i][k] + halfDeltatSqr * acc[i][k];
    }
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        pos[i][k] += deltar[i][k];
    }
  for (j = 0; j < n; j++)
    {
      i = alist[j]; /* v(t+dt/2) */
      for (k = 1; k <= 3; k++)
        vel[i][k] += halfDeltat * acc[i][k];
    }
}

void
VV_step_2 (int *alist, int n)
{
  int i, j, k;
  for (j = 0; j < n; j++)
    {
      i = alist[j];
      for (k = 1; k <= 3; k++)
        {
          vel[i][k] += halfDeltat * acc[i][k];
        }
    }
}

void
Compute_Forces (void)
{
  /*
     Compute forces on atoms from the positions, using the potential stored in
     the potential tables. Note how this routine is valid for any two-body
     potential. This version makes use of an existing neighbor list, and runs
```

```
    in O(N) time. */
auto double Rsqij, rho, drho, phi, dphi, rk, weight;
double Sij[4], posi[4];
auto int i, j, k, L;

for (i = 1; i <= Natoms; i++)
  {
    coord[i] = 0.0;
    ene_pot[i] = 0.0;
  }
for (i = 1; i <= Natoms; i++)
  {
    for (k = 1; k <= 3; k++)
      posi[k] = pos[i][k];
    for (L = Marker1[i]; L <= Marker2[i]; L++)
      {
        /* part of List useful for i */
        j = List[L];
        /* take neigh index out of List */
        Dist3v (posi, pos[j], Sij);
        Rsqij = Dot_Product_3v (Sij, Sij);

        if (Rsqij < RcutoffSqr)
          {
            rk = (Rsqij - RsqMin) * InvDeltaRsq + 1.0;
            /* rk is continuous index in table */
            k = (int) rk;
            if (k < 1)
              k = 1;
            weight = rk - k;
            rho = weight * RhoTab[k + 1] + (1.0 - weight) * RhoTab[k];
            coord[i] += rho;
            /* accumulate atomic density */
            coord[j] += rho;
            /* (i and j share it) */
          }
      }
  }
/* Now that we know n_i (coord), calculate glue part of energies U(n_i), and
    derivatives U'(n_i) (deru): */

for (i = 1; i <= Natoms; i++)
  {
    rk = coord[i] * InvDeltaCoord + 1.0;
    k = (int) rk;
    if (k >= TableSize)
      k = TableSize - 1;
    weight = rk - k;
    /* fractional part, in [0,1] */
    ene_pot[i] = weight * UTab[k + 1] + (1.0 - weight) * UTab[k];
    /* linear interpolation */
    deru[i] = weight * DUTab[k + 1] + (1.0 - weight) * DUTab[k];
    /* for U(n_i)/U'(n_i) */
  } /* i */

/* Reset to zero forces and virial term */
for (i = 1; i <= Natoms; i++)
  {
    acc[i][1] = 0.0;
    acc[i][2] = 0.0;
    acc[i][3] = 0.0;
  }
virial = 0.0;

/* Loop over particles in the neighbor list a second time, to compute the
    forces: */

for (i = 1; i <= Natoms; i++)
  {
    for (k = 1; k <= 3; k++)
      posi[k] = pos[i][k];

    for (L = Marker1[i]; L <= Marker2[i]; L++)
```

```
        {
          j = List[L];
          Dist3v (posi, pos[j], Sij);
          Rsqij = Dot_Product_3v (Sij, Sij);
          if (Rsqij < RcutoffSqr)
            {
              rk = (Rsqij - RsqMin) * InvDeltaRsq + 1.0;
              k = (int) rk;
              if (k < 1)
                k = 1;
              weight = rk - k;
              phi = weight * PhiTab[k + 1] + (1.0 - weight) * PhiTab[k];
              ene_pot[i] = ene_pot[i] + 0.50 * phi;
              /* accumulate energy */
              ene_pot[j] = ene_pot[j] + 0.50 * phi;
              /* (i and j share it) */
              dphi = weight * DPhiTab[k + 1] + (1.0 - weight) * DPhiTab[k];
              /* interpolation */
              drho = weight * DRhoTab[k + 1] + (1.0 - weight) * DRhoTab[k];
              dphi = dphi + (deru[i] + deru[j]) * drho;
              virial = virial - dphi * Rsqij;
              for (k = 1; k <= 3; k++)
                acc[i][k] = acc[i][k] + dphi * Sij[k];
              for (k = 1; k <= 3; k++)
                acc[j][k] = acc[j][k] - dphi * Sij[k];
              /* (Fji = -Fij) */
            }
        }
    } /* L */

  virial = -virial / 3;
  /* definition of virial term */
}

void
Update_DisplaceList (void)
{
  int i, k;
  /* Update DisplaceList, which contains the displacements of atoms since the
     last list update. */
  for (i = 1; i <= Natoms; i++)
    {
      for (k = 1; k <= 3; k++)
        DisplaceList[i][k] += deltar[i][k];
    }
}

int
MovedTooMuch ()
{
  /* Scans the DisplaceList vector, containing the displacement vectors of
     particles since the last time the list was updated. It looks for the
     magnitude of the two largest displacements in the system. These are
     summed together. If the sum is larger than Skin, the function returns
     .true., otherwise .false. */

  double Displ, Displ1, Displ2;
  int i;
  Displ1 = 0.0;
  Displ2 = 0.0;

  for (i = 1; i <= Natoms; i++)
    {
      Displ = sqrt (Dot_Product_3v (DisplaceList[i], DisplaceList[i]));
      if (Displ >= Displ1)
        { /* 1st position taken */
          Displ2 = Displ1; /* push current 1st into 2nd place */
          Displ1 = Displ;
        } /* and put this one into current 1st */
      else if (Displ >= Displ2)
        { /* 2nd position taken */
          Displ2 = Displ;
        }
```

```
    } /* i */
  return ((Displ1 + Displ2) > Skin) ? TRUE : FALSE;
}

void
Compute_Sk (int *alist, int n, double *sk)
{
  int i, j, k;
  double sinsum, cossum;

  if (n == 0)
    *sk = 0.0;
  else
    {
      sinsum = 0.0;
      cossum = 0.0;

      for (j = 0; j < n; j++)
        {
          i = alist[j];

          cossum = cossum + cos (Dot_Product_3v (klatt, pos[i]));
          sinsum = sinsum + sin (Dot_Product_3v (klatt, pos[i]));
        }
      cossum = cossum / (double) n;
      sinsum = sinsum / (double) n;
      *sk = sqrt (cossum * cossum + sinsum * sinsum);
    }
}

double
Compute_Etot ()
{
  double E;
  int i;
  E = 0.0;
  for (i = 1; i <= Natoms; i++)
    E += ene_pot[i];
  return (E);
}

void
Compute_Eppp (int *alist, int n, double *E)
{
  if (n == 0)
    *E = 0.0;
  else
    *E = Sum1v (ene_pot, alist, n) / (double) (n);
}

void
Compute_Ek_T_Vcm (int *alist, int n, double *Ek_pp, double *temp)
{
  /* calculate kinetic energy per particle and temperature. */
  int i;
  double two_thirds;
  two_thirds = 2.0 / 3.0;

  if (n == 0)
    *Ek_pp = 0.0;
  else
    *Ek_pp = Compute_Ek_tot (alist, n) / (double) (n);

  *temp = (two_thirds * *Ek_pp);
}

double
Compute_Ek_tot (int *alist, int n)
{
  int i, j, k;
  double esum = 0.0, Ek;
  double v[4];

  CalVelCM (alist, n, V_cm);

  if (n == 0)
    esum = 0.0;
  else
    {
```

```
      for (j = 0; j < n; j++)
        {
          i = alist[j];
          for (k = 1; k < 4; k++)
            v[k] = vel[i][k] - V_cm[k];
          esum += Dot_Product_3v (v, v);
        }
    }
  Ek = 0.5 * esum;
  return (Ek);
}

void
Report_Part_1 (char *control_filename)
{
  long tm;
  char outstr[80];
  int i;

  time (&tm);

  report_file = OpenFile (report_filename, "a");
  fprintf (report_file, "# %s %s", run_id, ctime (&tm));
  report ("%s", "Title:  Liquid Metal-Rigid Ceramic Simulation.");
  report ("%s", title);
  report ("%s", "control file %s read in successfully:", control_filename);
  for (i = 0; i < Name_List_Size; i++)
    report ("%-20s %-20s", Name_List[i], Value_List[i]);
}

void
Read_Control (char *cntrl_file_name)
{
  /* Load the file into memory */
  Load_Control_File (cntrl_file_name);
  /* now assign values to variables */
  run_id = Get_s_Variable ("run-id");
  sprintf (report_filename, "%s.report", run_id);
  title = Get_s_Variable ("title");
  Trequested = Get_d_Variable ("Trequested");

  Potential = Get_i_Variable ("Potential");
  Deltat = Get_d_Variable ("Deltat");
  Norm = Get_i_Variable ("Norm");
  BoxFactor[1] = Get_i_Variable ("Box-Factor-x");
  BoxFactor[2] = Get_i_Variable ("Box-Factor-y");
  BoxFactor[3] = Get_i_Variable ("Box-Factor-z");
  Skin = Get_d_Variable ("Skin");
  Trequested = k_B * Trequested;
  DeltatSqr = Deltat * Deltat;
  halfDeltatSqr = 0.5 * DeltatSqr;
  halfDeltat = 0.5 * Deltat;
}

void
Load_Control_File (char *cntrl_file_name)
{
  FILE *cntrl_file;
  int nl = 0, ncl = 0;
  char cl1[40], cl2[40], lnin[80];

  cntrl_file = OpenFile (cntrl_file_name, "r");
  while ((fgets (lnin, 80, cntrl_file) != NULL))
    {
      if (lnin[0] == '#')
        {
          fprintf (stdout, "%s", lnin);
        }
      else
        {
          if ((ncl = sscanf (lnin, "%s %s\n", cl1, cl2)) == 2)
            {
              strcpy (Name_List[nl], cl1);
              strcpy (Value_List[nl], cl2);
```

```
            }
          else if (ncl == -1)
            fclose (cntrl_file);
          else
            error ("Error reading control file at line %d :  ncl = %d", nl, ncl);
          ++nl;
          if (nl > Max_Num_Vars)
            error
              ("number of variables red in control file %d > Max_Num_Vars ",
               nl);
        }
    }
  Name_List_Size = nl;
} /* Read_Control */

int
Var_Loc (char *var_name)
{
  int i, rcount, iloc;

  iloc = rcount = 0;
  for (i = 0; i < Name_List_Size; i++)
    {
      if (strcmp (var_name, Name_List[i]) == 0)
        {
          iloc = i;
          rcount++;
        }
    }
  if (rcount == 0)
    error ("Variable | %s | not found in list", var_name);
  else if (rcount > 1)
    {
      warning ("More than one location found for variable | %s |");
      warning ("using last location:  %d", iloc);
    }
  return (iloc);
}

double
Get_d_Variable (char *var_name)
{
  int loc;

  loc = Var_Loc (var_name);
  return (atof (Value_List[loc]));
}

int
Get_i_Variable (char *var_name)
{
  int loc;

  loc = Var_Loc (var_name);
  return (atoi (Value_List[loc]));
}

char *
Get_s_Variable (char *var_name)
{
  int loc;

  loc = Var_Loc (var_name);
  return (Value_List[loc]);
}

void
Set_Evolution_Control ()
{

  /***** EVOLUTION PARAMETERS ****/
  report ("Reading Evolution control parameters:");
  /* report("MD time step: Deltat =%G", Deltat); */
  /* Trequested and Deltat are lready loaded in read control */
  MDsteps = Get_i_Variable ("MDsteps");
  report ("Number of MD steps :[ %-ld]", MDsteps);
```

```
  roll_interval = Get_i_Variable ("roll-interval");
  write_snap_interval = Get_i_Variable ("write-snap-interval");
  report ("roll_interval :[ %-ld]", roll_interval);
  report ("Skin :[ %-G]", Skin);
  report ("write-snap-interval :[ %-ld]", write_snap_interval);

  /**** THERMOSTAT *****/
  thermostat = Get_d_Variable ("thermostat");
  switch (thermostat)
    {
    case 0:
      report ("constant E run, No thermostat is applied");
      break;
    case 1:
      report ("Simple temperature rescaling:");
      report ("Requested temperature:  [ %-G]", Trequested / k_B);
      scale_interval = Get_i_Variable ("scale-interval");
      report ("scale-interval:  [ %-d]", scale_interval);
      break;
    default:
      report ("NO THERMOSTAT IS GIVEN, ABORTING");
      error ("NO THERMOSTAT IS GIVEN, ABORTING");
      break;
    }
  if (scale_end > MDsteps)
    {
      report ("Warning:  scale_end =%d > MDsteps =%d", scale_end, MDsteps);
      warning ("scale_end =%d > MDsteps =%d", scale_end, MDsteps);
    }
}

void
Stat ()
{
  int i;
  double Ek_pp, Ep_pp, T, EnTmp;

  /* sample and accumulate averages */
  if (ts_counter == 0)
    {
      Eppp.mean = Eppp.var = 0.0;
      Eppp.count = 0;

      GEkpp = GTkpp = GEppp = Ekpp = Tkpp
        = Etot = Ektot = GEtot = GEktot = Eppp;
      for (i = 1; i < 4; i++)
        {
          Vcm[i] = Ekpp;
          GVcm[i] = Ekpp;
        }
      for (i = 0; i < 3; i++)
        llc[i] = urc[i] = Gllc[i] = Gurc[i] = Eppp;
      sprintf (ET_filename, "%s%s", run_id, ".ET");
      ET_file = OpenFile (ET_filename, "w");

      fprintf (ET_file, "# time | Temp var | Eppp(liquid) var");
      fprintf (ET_file, "| Ekpp(liquid) var | |Eptot var| Ektot var|\n");

      sprintf (Vcm_filename, "%s%s", run_id, ".Vcm");
      Vcm_file = OpenFile (Vcm_filename, "w");
      fprintf (Vcm_file, "# time, Vcm, +-, var\n");

      sprintf (Box_filename, "%s%s", run_id, ".Box");
      Box_file = OpenFile (Box_filename, "w");
      fprintf (Box_file, "#ts_couner llc urc\n");
      fprintf (Box_file, "# x std y std z std x std y std z std\n");
    }

  /* average T and kinetic energy per (liquid) particle: */

  Compute_Ek_T_Vcm (llist, Nliquid, &Ek_pp, &T);
  Accumulate (&Ekpp, Ek_pp);
  Accumulate (&Tkpp, T);
  EnTmp = Compute_Ek_tot (All_atoms_list, Natoms);
  Accumulate (&Ekpp, EnTmp);
```

```
EnTmp = Compute_Etot ();
Accumulate (&Etot, EnTmp);

Compute_Eppp (llist, Nliquid, &Ep_pp);
Accumulate (&Eppp, Ep_pp);
for (i = 1; i < 4; i++)
  Accumulate (&Vcm[i], V_cm[i]);

scell = Get_Box ();
for (i = 0; i < 3; i++)
  {
    Accumulate (&urc[i], scell.urc[i]);
    Accumulate (&llc[i], scell.llc[i]);
  }
if (ts_counter % roll_interval == 0)
  {
    Average (&Eppp);
    Average (&Ekpp);
    Average (&Tkpp);
    Average (&Etot);
    Average (&Ektot);

    fprintf (ET_file, "% 10d", ts_counter + ts_init);
    fprintf (ET_file, " % 10f % 10e % 10e % 10e",
             Tkpp.mean / k_B, Tkpp.var / k_B, Eppp.mean, Eppp.var);

    fprintf (ET_file, " % 10e % 10e % 10e % 10e",
             Ekpp.mean, Ekpp.var, Etot.mean, Etot.var);
    fprintf (ET_file, " % 10e % 10e", Ektot.mean, Ektot.var);

    Accumulate (&GEkpp, Ekpp.mean);
    Accumulate (&GTkpp, Tkpp.mean);
    Accumulate (&GEppp, Eppp.mean);
    Accumulate (&GEtot, Etot.mean);
    Accumulate (&GEktot, Ektot.mean);

    for (i = 1; i < 4; i++)
      Average (&Vcm[i]);

    fprintf (Vcm_file,
             "% 10d % 10e % 5e % 10e % 5e % 10e % 5e\n",
             ts_counter + ts_init, Vcm[1].mean,
             Vcm[1].var, Vcm[2].mean, Vcm[2].var, Vcm[3].mean, Vcm[3].var);

    for (i = 1; i < 4; i++)
      Accumulate (&GVcm[i], Vcm[i].mean);
    /* Box Data */
    for (i = 0; i < 3; i++)
      {
        Average (&urc[i]);
        Average (&llc[i]);
      }

    fprintf (Box_file, "% 10d ", ts_counter + ts_init);
    for (i = 0; i < 3; i++)
      fprintf (Box_file, "% .5f % .3e ", llc[i].mean, llc[i].var);
    for (i = 0; i < 3; i++)
      fprintf (Box_file, "% .5f % .3e ", urc[i].mean, urc[i].var);
    fprintf (Box_file, "\n");

    for (i = 0; i < 3; i++)
      {
        Accumulate (&Gurc[i], urc[i].mean);
        Accumulate (&Gllc[i], llc[i].mean);
      }

    Eppp.mean = Eppp.var = 0.0;
    Eppp.count = 0;
    Ekpp = Tkpp = Etot = Ektot = Eppp;
    for (i = 1; i < 4; i++)
      Vcm[i] = Ekpp;
    for (i = 0; i < 3; i++)
      {
        llc[i] = urc[i] = Eppp;
      }
```

```
    }
}

void
Accumulate (struct dstat *q, double val)
{
  q->mean += val;
  q->var += val * val;
  q->count++;
}

void
Average (struct dstat *q)
{
  if (q->count == 0)
    q->count = 1;
  q->mean = q->mean / (double) q->count;

  q->var = q->var / (double) q->count - q->mean * q->mean;
  if (q->var > 0.0)
    q->var = sqrt (q->var);
  else
    q->var = 0.0;
}

void
Stat_Sla (int action)
{
  int i, j;
  double Ek, Ep, Tk, sk;

  if (action == 1)
    {
      Fill_Slice (&fss);
      for (i = 0; i < fss.SIZE; i++)
        {
          Compute_Eppp (fss.slice[i].iatom, fss.slice[i].count, &Ep);
          Compute_Ek_T_Vcm (fss.slice[i].iatom, fss.slice[i].count, &Ek, &Tk);
          Compute_Sk (fss.slice[i].iatom, fss.slice[i].count, &sk);

          Accumulate (&fss.Ep[i], Ep);
          Accumulate (&fss.Ek[i], Ek);
          Accumulate (&fss.Tk[i], Tk);
          Accumulate (&fss.Sk[i], sk);
        }
      if (ts_counter % sla_roll_interval == 0)
        {
          for (i = 0; i < 3; i++)
            {
              fprintf (fss.file[i], "\n\n");
              fprintf (fss.file[i], "# number-of-bins %-7d\n", fss.SIZE);
              fprintf (fss.file[i], "# ts-counter %-7d\n", ts_counter);
              fprintf (fss.file[i], "# sla-roll-interval %-7d\n",
                       sla_roll_interval);
              fprintf (fss.file[i], "# type % 7d\n", fss.type);
            }

          for (i = 0; i < fss.SIZE; i++)
            {
              if (fss.np[i].mean == 0.0)
                {
                  fss.Ep[i].mean = 0.0;
                  fss.Ep[i].var = 0.0;
                  fss.Ek[i].mean = 0.0;
                  fss.Ek[i].var = 0.0;
                  fss.Tk[i].mean = 0.0;
                  fss.Tk[i].var = 0.0;
                  fss.Sk[i].mean = 0.0;
                  fss.Sk[i].var = 0.0;
                }
              else
                {
                  Average (&fss.Ep[i]);
                  Average (&fss.Ek[i]);
                  Average (&fss.Tk[i]);
```

```
                    Average (&fss.Sk[i]);
                    Average (&fss.np[i]);
                  }
                fprintf (fss.file[0],
                         "% 10d % 5.3f % 8.4f % 8.4f % 8.4f\n",
                         i, fss.pos[i],
                         fss.width[i], fss.np[i].mean, fss.np[i].var);
                fprintf (fss.file[1],
                         "% 10d % 5.3f % 6.2f % 5.3f % 8.4f % 8.5f % 8.4f % 8.5f \n",
                         i, fss.pos[i],
                         fss.Tk[i].mean / k_B,
                         fss.Tk[i].var / k_B,
                         fss.Ep[i].mean,
                         fss.Ep[i].var, fss.Ek[i].mean, fss.Ek[i].var);
                fprintf (fss.file[2],
                         "% 10d % 5.3f % 6.2f % 5.3f\n",
                         i, fss.pos[i], fss.Sk[i].mean, fss.Sk[i].var);
              }
            fss.psum = 0.0;
            for (i = 0; i < fss.SIZE; i++)
              {
                fss.psum += fss.np[i].mean;
                fss.Ep[i] = fss.Ek[i] = fss.Tk[i] = fss.Sk[i] = fss.np[i] = dzero;
              }
            fprintf (fss.file[0], "# particle sum:  psum =%lf\n", fss.psum);
          }
      }
    else if (action == 0)
      {
        report (".........Invoking Stat_Sla for initializing.........");
        sla_roll_interval = Get_i_Variable ("sla-roll-interval");
        report
          ("sla-roll-interval :[ %-d]", sla_roll_interval);
        Set_fine_scale_slice ();
      }
}

void
Set_fine_scale_slice ()
{
  /* flag ==1 --> read file */
  /* This function either reads slice data from previous fss1 output file, or
     defines a new uniform slice */
  /* some work may be done to make it more readable */
  int i, n, ncomment = 0, flag;
  double density, slice_volume;
  char slice_filename[200];
  double w;
  char lnin[80], *fss_filename;
  FILE *fss_file;
  double scratch[4];
  /* scratch, since with old gnu c, its tricky reading from a file directly to
     an array */
  /* look if fss-file is defined (i.e. not NULL) */
  fss_filename = Get_s_Variable ("fss-file");
  if (strcmp (fss_filename, "NULL") != 0)
    {
      fss_file = OpenFile (fss_filename, "r");
      report ("fine scale slice read from file:  %s", fss_filename);
      flag = 1;
    }
  else
    flag = 0;

  report
    ("In Set_fine_scale_slice:  assuming rigid part at the lower-left-corner end.");

  if (flag == 1)
    {
      /* now scan fine-scale-slice file and read in slice data */
```

```
    /* skip and count comment lines */
    while (fgets (lnin, 80, fss_file) != NULL)
      {
        if (lnin[0] != '#' && lnin[0] != '\n')
          break;
        ncomment++;
      }
    /* count the number of slices */
    while (lnin != NULL)
      {
        n = sscanf (lnin, "%d %f %f %f %f",
                    &i, &scratch[0], &scratch[1], &scratch[2], &scratch[3]);
        if (n != 5)
          error ("something wrong in reading fss_file n=%d", n);
        fgets (lnin, 80, fss_file);
        if (lnin[0] == '#' || lnin[0] == '\n')
          break;
      }
    fss.SIZE = i + 1;
    fclose (fss_file);
    /* this is like rewind... */
    fss_file = OpenFile (fss_filename, "r");
    report ("Number of slices in file:  :fss.SIZE");

    fss.pos = (double *) calloc (fss.SIZE, sizeof (double));
    fss.width = (double *) calloc (fss.SIZE, sizeof (double));
    fss.slice =
      (struct SINGLE_SLICE *) calloc (fss.SIZE, sizeof (struct SINGLE_SLICE));
    /* now read it in */
    for (i = 0; i < ncomment; i++)
      fgets (lnin, 80, fss_file);

    report ("Reading fss1 data from file:");
    for (i = 0; i < fss.SIZE; i++)
      {
        fscanf (fss_file, "%ld %lf %lf %lf %lf",
                &n, &scratch[0], &scratch[1], &scratch[2], &scratch[3]);
        fss.pos[i] = scratch[0];
        fss.width[i] = scratch[1];
        report ("%d %f %f %f %f\n",
                i, fss.pos[i], fss.width[i], scratch[2], scratch[3]);
      }
    report ("slice information read successfully");
  }
else
  {
    fss.type = 0; /* uniform spacing */
    report (" :uniform slicing");

    w = Get_d_Variable ("slice-width");

    fss.SIZE = scell.width[Norm - 1] * 2. / w;

    report ("Uniform slice width:  %f", w);
    report ("Total number of uniform slices:  :[ %-d]", fss.SIZE);

    fss.pos = (double *) calloc (fss.SIZE, sizeof (double));
    fss.width = (double *) calloc (fss.SIZE, sizeof (double));
    fss.slice =
      (struct SINGLE_SLICE *) calloc (fss.SIZE, sizeof (struct SINGLE_SLICE));
    fss.width[0] = w;
  }

fss.lms = MAX_NUM_PARTICLES_IN_SLICE; /* defines in .h */
for (i = 0; i < fss.SIZE; i++)
  {
    fss.slice[i].iatom = (int *) calloc (fss.lms, sizeof (int));
  }
report ("maximum particles in each slice :[ %-d]", fss.lms);

sprintf (slice_filename, "%s%s", run_id, ".fss1");
fss.file[0] = OpenFile (slice_filename, "w");
fprintf (fss.file[0], "# slice pos width np +- var\n");
report ("slice data in file :%s", slice_filename);
```

```
  strcpy (slice_filename, " ");
  sprintf (slice_filename, "%s%s", run_id, ".fss2");
  fss.file[1] = OpenFile (slice_filename, "w");
  fprintf (fss.file[1],
           "# % 5s % 8s % 8s % 2s % 5s % 8s % 2s % 5s % 8s % 2s % 5s\n",
           "slice", "pos", "Tk", "+-", "var",
           "Ep", "+-", "var", "Ek", "+-", "var");

  report ("slice data in file :%s", slice_filename);

  strcpy (slice_filename, " ");
  sprintf (slice_filename, "%s%s", run_id, ".fss3");
  fss.file[2] = OpenFile (slice_filename, "w");
  fprintf (fss.file[2],
           "# % 10s % 10s % 10s % 2s % 5s\n",
           "slice", "pos", "sk", "+-", "var");

  report ("slice data in file :%s", slice_filename);

  if (flag != 1)
    { /* i.e. slice not read from file... */
      fss.width[0] = w;
      fss.pos[0] = scell.llc[Norm - 1];

      for (i = 1; i < fss.SIZE; i++)
        {
          fss.width[i] = fss.width[0];
          fss.pos[i] = fss.pos[i - 1] + 0.5 * (fss.width[i - 1] + fss.width[i]);
        }
    }
  fss.Ep = (struct dstat *) calloc (fss.SIZE, sizeof (struct dstat));
  fss.Ek = (struct dstat *) calloc (fss.SIZE, sizeof (struct dstat));
  fss.Tk = (struct dstat *) calloc (fss.SIZE, sizeof (struct dstat));
  fss.Sk = (struct dstat *) calloc (fss.SIZE, sizeof (struct dstat));
  fss.np = (struct dstat *) calloc (fss.SIZE, sizeof (struct dstat));
}

struct BBOX
Get_Box ()
{
  int i, k;
  double urc[4], llc[4];
  struct BBOX bbox;

  for (k = 1; k <= 3; k++)
    {
      urc[k] = pos[1][k]; /* upper right cornet */
      llc[k] = pos[1][k]; /* lower left corner */
    }

  for (i = 1; i <= Natoms; i++)
    {
      for (k = 1; k <= 3; k++)
        {
          urc[k] = max (urc[k], pos[i][k]);
          llc[k] = min (llc[k], pos[i][k]);
        }
    }

  for (k = 0; k < 3; k++)
    bbox.urc[k] = urc[k + 1];
  for (k = 0; k < 3; k++)
    bbox.llc[k] = llc[k + 1];

  for (k = 0; k < 3; k++)
    bbox.width[k] = bbox.urc[k] - bbox.llc[k];

  return (bbox);
}

void
Fill_Slice (struct SLICING *s)
{
  int i, j;
  static double base;
  double dist;
```

```
  /*
     for uniform slicing: */

  /* Check if we need to reset trace of previous slicing */
  if (s→counter > 0)
    {
      for (i = 0; i < s→SIZE; i++)
        {
          for (j = 0; j < s→lms; j++)
            s→slice[i].iatom[j] = 0;
          s→slice[i].count = 0;
        }
    }

  base = scell.llc[Norm - 1] - 0.5 * s→width[0];

  for (i = 1; i <= Natoms; i++)
    {
      dist = pos[i][Norm] - base;
      j = (int) floor (dist / s→width[0]);

      s→slice[j].iatom[s→slice[j].count] = i;
      s→slice[j].count += 1;
      if (s→slice[j].count > s→lms)
        {
          report
            ("ERROR: List size lms not large enough to hold %d",
             s→slice[i].count);
          error
            ("List size lms not large enough to hold %d", s→slice[i].count);
        }
    }
  for (i = 0; i < s→SIZE; i++)
    {
      Accumulate (&s→np[i], s→slice[i].count);
    }
  s→counter++;
}

void
Init_Model ()
{
  FILE *cva2File;
  int velacc = TRUE;
  int i, k;

  report ("Inside....  Init_Model");
  cva2FileName = Get_s_Variable ("cva2FileName");
  cva2File = OpenFile (cva2FileName, "r");
  report ("Initial cva2 FileName:  %s", cva2FileName);

  /* first read the xyz part of the cva2 file */
  report ("Invoking....  Input_cva2_xyz_part");
  Input_cva2_xyz_part (cva2File);

  report ("Invoking...  Define_Atom_Lists");
  Define_Atom_Lists ();

  deltar = dmatrix (1, Natoms, 1, 3);
  vel = dmatrix (1, Natoms, 1, 3);
  acc = dmatrix (1, Natoms, 1, 3);
  ene_pot = dvector (1, Natoms);
  ene_kin = dvector (1, Natoms);
  coord = dvector (1, Natoms);
  deru = dvector (1, Natoms);

  /* arrays related to neighbor lists */
  MaxListLength = MaxPairsPerAtom * Natoms;
  List = ivector (1, MaxListLength);
  Marker1 = ivector (1, Natoms);
  Marker2 = ivector (1, Natoms);
  Advance = ivector (1, Natoms);
  DisplaceList = dmatrix (1, Natoms, 1, 3);

  /* zero variables */
  for (i = 1; i <= Natoms; i++)
```

```
    {
      ene_pot[i] = 0.0;
      vel[i][1] = vel[i][2] = vel[i][3] = 0.0;
      acc[i][1] = acc[i][2] = acc[i][3] = 0.0;
      deltar[i][1] = deltar[i][2] = deltar[i][3] = 0.0;
    }

  report
    ("original simulation cell :[ %-5G][ %-5G][ %-5G]",
     BoxSize[1], BoxSize[2], BoxSize[3]);

  for (i = 1; i <= 3; i++)
    {
      BoxSizeBackup[i] = BoxSize[i];
      BoxSize[i] = BoxFactor[i] * BoxSize[i];
    }

  report ("Box-Factor is:  [%d %d %d]",
          BoxFactor[1], BoxFactor[2], BoxFactor[3]);
  report ("Norm is:  %d", Norm);
  for (i = 1; i <= 3; i++)
    InvBoxSize[i] = 1.0 / BoxSize[i];
  report
    ("actual simulation cell :[ %-5G][ %-5G][ %-5G]",
     BoxSize[1], BoxSize[2], BoxSize[3]);
  velacc = Load_vel (cva2File);
  if (velacc == FALSE)
    {
      Init_velocity (llist, Nliquid);
      Null_Vel (slist, Nsolid);
      for (i = 1; i <= Natoms; i++)
        { /* accellerations are initially set to zero */
          for (k = 1; k <= 3; k++)
            acc[i][k] = 0.0;
        }
      report ("Velocities taken from Gaussian distribution");
      report ("Accelerations set to zero ");
    }
  else
    {
      report ("Reading accelerations");
      Load_acc (cva2File);
      report ("Reading accelerations");
    }
  for (i = 1; i <= Natoms; i++)
    {
      Refold (pos[i]);
    }
}

int
Input_cva2_xyz_part (FILE * cva2File)
{
  /* Input the xyz part only of the cva2 file. */
  char xyzFileName[BUFFSIZE];
  float thisRead[11];
  char *buffer = (char *) malloc (BUFSIZ);
  char *sub_string, *ptr;
  double bbox[3];
  int i, k, ts, velacc = TRUE;

  fgets (buffer, BUFFSIZE, cva2File);
  sub_string = strtok (buffer, " ");
  Natoms = atoi (sub_string);

  for (i = 0; i < 3; i++)
    {
      sub_string = strtok (NULL, " ");
      BoxSize[i + 1] = atof (sub_string);
    }
  sub_string = strtok (NULL, " ");
  ts_init = atoi (sub_string);
  report ("Inside Input cva2:");
```

```
  report ("Read Header:  %d %f %f %f %d\n",
          Natoms, BoxSize[1], BoxSize[2], BoxSize[3], ts_init);

  pos = dmatrix (1, Natoms, 1, 3);
  Atype = (char **) malloc (sizeof (char *) * (Natoms + 1));
  for (i = 0; i <= Natoms; i++)
    Atype[i] = (char *) malloc (sizeof (char) * 3);

  fgets (buffer, BUFFSIZE, cva2File);
  sub_string = strtok (buffer, "\n");
  strcpy (xyzFileName, sub_string);
  report ("xyz FileName:  %s", xyzFileName);

  /* read xyz data file */
  if (Load_xyz_data (xyzFileName) == -1)
    error ("No data found in xyzFile:  %s", xyzFileName);
  report ("Read xyz data ...");
} /* main */

int
Load_xyz_data (char xyzFileName[BUFFSIZE])
{
  /* returns Natoms, otherwise a -1 if a prblem */
  /* store atom types into the array: Atype store positions into the array :
     pos */

  FILE *xyzFile;
  int natoms;
  char type[3];
  float thisRead[11];
  char *buffer = (char *) malloc (BUFFSIZE);
  char *sub_string, *ptr;
  int i, k;

  report ("Reading xyz data ...");
  xyzFile = OpenFile (xyzFileName, "r");
  fgets (buffer, BUFFSIZE, xyzFile);

  natoms = atoi (buffer);

  /* read and discarn the id string */
  fgets (buffer, BUFFSIZE, xyzFile);

  for (i = 1; i <= natoms; i++)
    {
      fgets (buffer, BUFFSIZE, xyzFile);

      k = 0;
      while (buffer[k] == ' ')
        k++;
      type[0] = buffer[0 + k];
      buffer[0 + k] = ' ';
      type[1] = buffer[1 + k];
      buffer[1 + k] = ' ';
      type[2] = 0;

      /* read the positions */
      thisRead[0] = atof (strtok (buffer + 2, " "));
      for (k = 1; k <= 3; k++)
        {
          ptr = strtok (NULL, " ");
          if (ptr)
            {
              thisRead[k] = atof (ptr);
            }
          else
            {
              thisRead[k] = 0.0;
            }
        }

      strcpy (Atype[i], type);
      pos[i][1] = thisRead[0];
      pos[i][2] = thisRead[1];
      pos[i][3] = thisRead[2];
    }
```

```
  return (natoms);
}

int
Load_vel (FILE * cva2File)
{
  float thisRead[11];
  char *buffer = (char *) malloc (BUFSIZ);
  char *sub_string, *ptr;
  int i, k;
  int velacc = TRUE;
  int particle_count = 0;

  for (i = 1; i <= Natoms; i++)
    {
      if (fgets (buffer, BUFFSIZE, cva2File) != NULL)
        {
          /* read the positions */
          thisRead[0] = atof (strtok (buffer, " "));
          for (k = 1; k < 3; k++)
            {
              ptr = strtok (NULL, " ");
              if (ptr)
                thisRead[k] = atof (ptr);
              else
                thisRead[k] = 0.0;
            }
          vel[i][1] = thisRead[0];
          vel[i][2] = thisRead[1];
          vel[i][3] = thisRead[2];
          /* fprintf(stdout, " %f %f %f\n", vel[i][1], vel[i][2], vel[i][3]); */
          particle_count++;
        }
    }
  report ("Read %d velocities", particle_count);
  if (particle_count == 0)
    velacc = FALSE;
  return (velacc);
}

int
Load_acc (FILE * cva2File)
{
  float thisRead[11];
  char *buffer = (char *) malloc (BUFSIZ);
  char *sub_string, *ptr;
  int i, k;
  int particle_count = 0;

  for (i = 1; i <= Natoms; i++)
    {
      if (fgets (buffer, BUFFSIZE, cva2File) != NULL)
        {
          /* read the positions */
          thisRead[0] = atof (strtok (buffer, " "));
          for (k = 1; k < 3; k++)
            {
              ptr = strtok (NULL, " ");
              if (ptr)
                thisRead[k] = atof (ptr);
              else
                thisRead[k] = 0.0;
            }
          acc[i][1] = thisRead[0];
          acc[i][2] = thisRead[1];
          acc[i][3] = thisRead[2];
          /* fprintf(stdout, " %f %f %f\n", acc[i][1], acc[i][2], acc[i][3]); */
          particle_count++;
        }
    }
  report ("Read %d accelerations", particle_count);
  if (particle_count < Natoms)
    error ("not enough accelerations in file");
```

```
}

void
Define_Atom_Lists ()
{
  int i, k, sc, lc;
  Atom_Code tag;

  Nsolid = Nliquid = 0;

  /* first loop to calculate the number of solid and liquid atoms in file */
  for (i = 1; i <= Natoms; i++)
    {
      tag = AtomType (Atype[i]);
      switch (tag)
        {
        case solid:
          Nsolid++;
          break;
        case liquid:
          Nliquid++;
          break;
        default:
          error
            ("Neither Solid nor Liquid atom!  Atype[%d] is %s!", i, Atype[i]);
          break;
        } /* switch */
    } /* i */
  llist = (int *) calloc (Nliquid, sizeof (int));
  slist = (int *) calloc (Nsolid, sizeof (int));
  All_atoms_list = (int *) calloc (Natoms, sizeof (int));
  for (i = 0; i < Natoms; i++)
    All_atoms_list[i] = i + 1; /* this is for the calculation of Ek_tot... */

  sc = lc = 0;
  for (i = 1; i <= Natoms; i++)
    {
      tag = AtomType (Atype[i]);
      switch (tag)
        {
        case solid:
          slist[sc] = i;
          sc++;
          break;
        case liquid:
          llist[lc] = i;
          lc++;
          break;
        default:
          error ("Neither Solid not Liquid atom!  Atype[%d] is %s!", Atype[i]);
          break;
        } /* switch */
    } /* i */
  report ("Inside Define_Atom_Listst");
  report ("found %d liquid atoms", Nliquid);
  report ("fount %d solid atoms", Nsolid);
  report ("Exiting Define_Atom_Listst");
} /* DEfine.. */

int
AtomType (char atype[ ])
{
  static char Atom_type_list[7][4] =
    { "S ", "L ", "Al", "O ", "B ", "C ", "Cu" };
  int list_size = 7;
  int i, rcount, iloc;
  iloc = rcount = 0;
  for (i = 0; i < list_size; i++)
    {
      if (strncmp (atype, Atom_type_list[i], 2) == 0)
        {
          iloc = i;
          rcount++;
        }
    }
  if (rcount == 0)
```

```
    error ("@@ Atom |%s| not found in list", atype);
  else if (rcount > 1)
    {
      warning ("More than one location found for Atom type | %s |", atype);
      warning ("using last location:  %d", iloc);
    }
  return (iloc);
}
```

MD.h

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <stdarg.h>
#include "nrutil.h"
#include "mytypes.h"
#include <string.h>
#include "cva.h"

#define FALSE 0
#define TRUE 1
#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))

#define PI 3.14159265358979323846
#define k_B (8.617385E-5) /* Boltzmann constant */
#define TimeUint 5.288E-14 /* time unit in secs */

#define MAX_MODEL_NUMBER 4
#define BUFFSIZE 240

#define MAX_NUM_PARTICLES_IN_SLICE 5000

#define CONST_E 0
#define RESCALE_T 1
#define HENDERSON 2

typedef enum {solid=0, liquid, other} Atom_Code;

long int iseed=123;

int BoxFactor[4] = {1,1,1};
int BoxSizeBackup[4];


/* stat variables */
struct dstat Eppp, Ekpp, Tkpp, Vcm[4], Etot, Ektot;
double V_cm[4]; /* poten. kin. and T per particle. */
struct dstat GEppp, GEkpp, GTkpp, GVcm[4], GEtot, GEktot;
struct dstat urc[3], llc[3], Gurc[3], Gllc[3];


struct BBOX scell; /* simulation cell */

struct SLICING fss, caus, canus;


char *cva2FileName;
char *run_id;
char report_filename[200];
FILE *report_file;
char ET_filename[200];
FILE *ET_file;
char Vcm_filename[200];
FILE *Vcm_file;
char Box_filename[200];
FILE *Box_file;

char *title;

int ModelNumber;
int Natoms, laf, lal;
int Nliquid, Nsolid;
int *llist, *slist, *All_atoms_list;
int Np[3], Npr[3], Napp[3];
```

```
double Deltat, a_latt, dp[3];
int Norm;
char snorm[80];
double klatt[4];
double BoxSize[4], InvBoxSize[4];

double Trequested;
int scale_interval=1000000, scale_end = 0, thermostat=0, Potential=0;
long MDsteps, roll_interval=10, sla_roll_interval=10, begin_stat_sla;
long write_snap_interval =1000000;
long ts = 0, ts_counter = 0, ts_init=0;

double **deltar; /* displacement */
double **pos; /* positions */
double **vel; /* velocities */
double **acc; /* accelerations */
char **Atype; /* atom types */


double *ene_pot;
double *ene_kin;
double *coord; /* coordinations */
double *deru; /* U'(n_i) */
double virial; /* the virial to compute the pressure term */

#define MaxPairsPerAtom 100
int MaxListLength;
int *Advance, *Marker1, *Marker2, *List;
double **DisplaceList;
double Skin = 3.0; /* Angstroms */
int updateList = TRUE;


double DeltatSqr, halfDeltatSqr, halfDeltat;



/*
    Parameters and tables of the ab-initio derived glue potential for
    Aluminum [F. Ercolessi and James B. Adams, Europhysics Letters 26,
    583 (1994) ]. The tables are filled by calling externally defined
    subroutines v2, rh, uu, returning the actual potential.
*/

/* If running on AIX, SGI or similar OS need to define underscore...*/
#if underscore
#define v2 v2_
#define rh rh_
#define uu uu_
#endif


double Rcutoff = 0.555805441821810e+01, RcutoffSqr;
double RcutoffPlusSkin = 0.555805441821810e+01+3.0;
double Rmin;
double RsqMin;
double DeltaRsq;
double InvDeltaRsq;
double CoordMax;
double DeltaCoord;
double InvDeltaCoord;
#define TableSize 20001

double *PhiTab, *DPhiTab; /* phi(r),1/r dphi/dr */
double *RhoTab, *DRhoTab; /* rho(r),1/r drho/dr */
double *UTab, *DUTab; /* U(n), du/dn */

void Define_Aluminium_Potential_Tables(void);

extern void v2(double *, double *, double *, double *);
extern void rh(double *, double *, double *, double *);

#define Max_Num_Vars 100 /* Maximunm number of variables allowed in control file */
char Name_List [ Max_Num_Vars ][40];
char Value_List [ Max_Num_Vars ][40];
char Type_List [ Max_Num_Vars ][5];
int Name_List_Size;
char *Get_s_Variable ( char *);
```

double Get_d_Variable (**char** ∗);
int Get_i_Variable (**char** ∗);

void Init_fcc (**void**);
void Read_Initial_Configuration(**void**);
void WriteSnapShot(**void**);
FILE ∗OpenFile(**char** ∗, **char** ∗);
void warning(**char** ∗, ...);
void error(**char** ∗, ...);
void report(**char** ∗, ...);
void pr_stderr(**char** ∗, ...);
void usage(**void**);
void Read_vel (FILE ∗);
void Read_acc (FILE ∗);
void Allocate_Mem (**void**);
void Update_NeighborList(**double**);

double Dot_Product_3v(**double** ∗, **double** ∗);
void Dist3v(**double** ∗, **double** ∗, **double** ∗);
void Refold(**double** ∗);
void VVEA_T (**int**);
void VVEA_E (**int**);
void VV_Const_T_step_1(**int** ∗, **int** , **double**);
void VV_Const_E_step_1(**int** ∗, **int**);
void VV_step_2(**int** ∗, **int**);
void Compute_Forces(**void**);
void Update_DisplaceList(**void**);
int MovedTooMuch();
void Compute_Eppp (**int** ∗, **int**, **double** ∗);
double Compute_Ek_tot(**int** ∗, **int**);
double Compute_Etot();
void Compute_Ek_T_Vcm (**int** ∗, **int**, **double** ∗, **double** ∗);
double Compute_T(**int** ∗, **int**);
void Compute_Sk(**int** ∗, **int**, **double** ∗);
void Report_Part_1 ( );
void Report_Part_2 ( );
void Read_Control (**char** ∗);
void Load_Control_File (**char** ∗);

double Sum1v(**double** ∗, **int** ∗, **int**);
void Accumulate(**struct** dstat ∗, **double**);
void Set_Evolution_Control ();
void Stat (**void**);
void Average (**struct** dstat ∗);
void Stat_Sla (**int**);
void Init_velocity(**int** ∗ , **int**);
void Null_Vel(**int** ∗, **int**);
void CalVelCM (**int** ∗, **int** , **double** ∗);
void TransVel2CM (**int** ∗, **int**);
void TransPos2CM (**void**);
void gran(**double** ∗);
struct BBOX Get_Box( );
void Set_fine_scale_slice ( );
void Fill_Slice (**struct** SLICING ∗s);
int Load_xyz_data (**char** xyzFileName[BUFFSIZE]);
void Init_Model ( );
int Input_cva2_xyz_part(FILE ∗ cva2File);
void Define_Atom_Lists();
int Load_acc (FILE ∗cva2File);
int Load_vel (FILE ∗cva2File);
int AtomType (**char** ∗type);
void WriteSnapShot_CVA2();

cva.c

```
#include "CVAGLOBAL.h"
#include "cva.h"
```

```
/* error: print an error message and die */
void
error (char *fmt, ...)
{
  va_list args;

  va_start (args, fmt);
  fprintf (stderr, "# error:  ");
  vfprintf (stderr, fmt, args);
  fprintf (stderr, "\n");
  va_end (args);
  exit (1);
}

/* warning: print a warning message and continue */
void
warning (char *fmt, ...)
{
  va_list args;

  va_start (args, fmt);
  if (Quiet == FALSE)
    {
      fprintf (stderr, "# Warning:  ");
      vfprintf (stderr, fmt, args);
      fprintf (stderr, "\n");
    }
  va_end (args);

}

void
Allocate_pos_CVA_Buffer (struct CVA_BUFFER *buff, int Natoms)
{
  int i;
  buff->H.natoms = Natoms;

  buff->pos = (double **) malloc (sizeof (double *) * Natoms);
  for (i = 0; i < Natoms; i++)
    buff->pos[i] = (double *) malloc (sizeof (double) * 3);

  buff->atype = (int *) malloc (sizeof (int) * Natoms);

}

void
describe_CVA_Buffer (struct CVA_BUFFER *buff, char *blabla)
{

  strcpy (buff->H.description, blabla);
}

void
Init_CVA_Buffer (struct CVA_BUFFER *buff)
{
  /*
     Initialize buffer: give default header values, a tag, no memory
     allocations are performed for pos and vel, etc ...

   */

  int k;
  int Is_EXISTING = 0;

  if (buff->H.tag > 0 && buff->H.tag <= BUFFERS_HANDLE && buff->H.status == 1)
    {

      warning
        ("in Init_CVA_Buffer:  <re-initializing an existing Buffer %d",
         buff->H.tag);
      /* maybe later can this can be added.. */
      Is_EXISTING = 1;
    }
  if (Is_EXISTING == 0)
    {
      BUFFERS_HANDLE++;
      buff->H.tag = BUFFERS_HANDLE;
      buff->H.status = 1; /* to indicate it is initialized at least
```

```
                                        once... */
    }
  strcpy (buff→H.fname, "\0");

  buff→H.fhandle = NULL;
  strcpy (buff→H.description, "\0");
  strcpy (buff→H.rems, SPERCENT);

  strcpy (buff→H.vacc, "F");
  buff→H.velac = FALSE;
  buff→H.natoms = 0;
  buff→H.box.x = 0.0;
  buff→H.box.y = 0.0;
  buff→H.box.z = 0.0;

  /* now initialize BBox: */
  for (k = 0; k < 3; k++)
    {
      buff→H.bbox.llc[k] = buff→H.bbox.urc[k] = 0.0;
    }

  for (k = 0; k < 3; k++)
    {
      buff→H.bbox.width[k] = 0.0;
    }

  buff→H.bbox.volume = 0.0;
}

void
Wipe_CVA_Buffer (struct CVA_BUFFER *buff)
{
  /*
     Wipes the buffer in memory, i.e. it leaves some residues around, but
     marks its tag as -1;

     Must add the part to wipe vel and acc if they are present, and must check
     if pos is present too, i.e i have must put some more status variables into
     Header, example H.status.pos, vel. etc... */

  if (buff→H.fhandle != NULL)
    {
      warning ("in Wipe_CVA_Buffer:  <file handle not NULL>");
      Close_CVA_Buffer_File (buff);
    }
  else if (buff→H.tag < 0)
    {
      error
        ("in Wipe_CVA_Buffer:  <buffer is already closed tag is %d>",
         buff→H.tag);
    }
  else
    {
      int i;
      for (i = 0; i < buff→H.natoms; i++)
        free (buff→pos[i]);
      free (buff→pos);

      if (strcmp (buff→H.vacc, "T"))
        free (buff→vel);

      buff→H.tag = -1;
    }
}

void
Print_CVA_Header (struct CVA_BUFFER *buff)
{

  printf ("#");
  {
    int dash;
    for (dash = 0; dash < 77; dash++)
      printf ("-");
    printf ("\n");
  }
```

```
  fprintf (stdout, "<<< Print_CVA_Header >>>\n");

  fprintf (stdout, "# Buffer Header:  %d @ %s\n", buff→H.tag, buff→H.fname);

  fprintf (stdout, "# %s %s %d %E %E %E\n",
           buff→H.rems, buff→H.vacc, buff→H.natoms,
           buff→H.box.x, buff→H.box.y, buff→H.box.z);
  fprintf (stdout,
           "# BBox:\n# volume = %f\n# llc= %f %f %f\n# urc= %f %f %f\n# width = %f %f %f\n",
           buff→H.bbox.volume, buff→H.bbox.llc[0],
           buff→H.bbox.llc[1], buff→H.bbox.llc[2],
           buff→H.bbox.urc[0], buff→H.bbox.urc[1],
           buff→H.bbox.urc[2], buff→H.bbox.width[0],
           buff→H.bbox.width[1], buff→H.bbox.width[2]);
  fprintf (stdout, "# %-10s %-10s %-10s %-10s", "status", "fhandle",
           "rems", "velacc");

  fprintf (stdout, "\n");
  fprintf (stdout, "# ");
  fprintf (stdout, " % -10d", buff→H.status);
  if (buff→H.fhandle == NULL)
    fprintf (stdout, "%-10s ", "NULL");
  else
    fprintf (stdout, "%-10s ", "++");
  fprintf (stdout, "%-10s ", buff→H.rems);
  fprintf (stdout, "%-10s ", buff→H.vacc);
  fprintf (stdout, "\n");

  fprintf (stdout, ">>> Leaving:  Print_CVA_Header <<<\n");
}

char *
Get_CVA_inFile (char name_array[ ][MAX_STRING_LENGTH], int ifile)
{

  if (ifile < 0 || ifile >= MAX_NUM_IN_FILES)
    {
      error ("in Get_inFile:  <Address %d out of bounds>", ifile);
    }
  else
    {
      return (name_array[ifile]);
    }

}

void
Open_CVA_Buffer_File (struct CVA_BUFFER *buff, char *filename)
{
  /*
     open buffer file, read in the header, and return it. the file remains
     open, and the handle is returned too as 'Hbuff.fhandle'

   */

  if (buff→H.tag <= 0 || buff→H.tag > BUFFERS_HANDLE)
    {
      /* warning("in Open_Buffer_FILE: <buffer does not exist, invoking
         Init_CVA_Buffer>"); */

      Init_CVA_Buffer (buff);
    }

  strcpy (buff→H.fname, filename);

  buff→H.fhandle = OpenFile (buff→H.fname, "r");

  fscanf (buff→H.fhandle,
          "%s %s %d %lE %lE %lE %ld", buff→H.rems, buff→H.vacc,
          &buff→H.natoms, &buff→H.box.x, &buff→H.box.y,
          &buff→H.box.z, &buff→H.tstamp);

}

void
Close_CVA_Buffer_File (struct CVA_BUFFER *buff)
{

  /*
```

```
    Close the buffer file, but leave the buffer alive in memory. */

  if (buff→H.fhandle == NULL)
    error ("in Close_CVA_Buffer_File:  <file handle not available>");

  fclose (buff→H.fhandle);
  buff→H.fhandle = NULL;
}

void
Save_CVA_Buffer_to_File (struct CVA_BUFFER *buff)
{

  int i, Natoms;
  char cva_filename[200];
  Natoms = buff→H.natoms;

  if (buff→H.fname == NULL)
    error ("cant save buffer, no header specified");

  strcpy (cva_filename, buff→H.fname);
  strcat (cva_filename, ".cva");
  buff→H.fhandle = OpenFile (buff→H.fname, "w");

  fprintf (buff→H.fhandle,
           "%s %s %d %E %E %E %ld\n", buff→H.rems, buff→H.vacc,
           buff→H.natoms, buff→H.box.x, buff→H.box.y,
           buff→H.box.z, buff→H.tstamp);

  for (i = 0; i < Natoms; i++)
    {

      fprintf (buff→H.fhandle, "%f %f %f\n",
               buff→pos[i][0], buff→pos[i][1], buff→pos[i][2]);
    }

}

void
Save_CVA2_Buffer_to_File (struct CVA_BUFFER *buff, char *xyz_filename)
{
  int i, Natoms;
  FILE *cva2_file;
  char cva2_filename[200];

  Natoms = buff→H.natoms;

  if (buff→H.fname == NULL)
    error ("cant save buffer, no header specified");

  strcpy (cva2_filename, buff→H.fname);
  strcat (cva2_filename, ".cva2");
  cva2_file = OpenFile (cva2_filename, "w");

  fprintf (cva2_file,
           "%d %E %E %E %d\n",
           buff→H.natoms, buff→H.box.x, buff→H.box.y,
           buff→H.box.z, buff→H.tstamp);

  fprintf (cva2_file, "%s\n", xyz_filename);

  if (buff→H.velac == FALSE)
    {
      fclose (cva2_file);
      return;
    }
  else
    {

      for (i = 0; i < Natoms; i++)
        {

          fprintf (cva2_file, "%f %f %f\n", buff→vel[i][0],
                   buff→vel[i][1], buff→vel[i][2]);

        }

      for (i = 0; i < Natoms; i++)
        {
          fprintf (cva2_file, "%f %f %f\n",
```

```
                        buff→acc[i][0], buff→acc[i][1], buff→acc[i][2]);
          }
        fclose (cva2_file);
        return;
      }

  return;

}

void
Save_CVA2XYZ_to_File (struct CVA_BUFFER *buff)
{

  /* the xyz part is first saved, later I could add a flag to say whether cv2
     needs to be saved too */

  int i, Natoms;
  FILE *xyz_file;
  char xyz_filename[200];

  Natoms = buff→H.natoms;

  if (buff→H.fname == NULL)
    error ("cant save buffer, no header specified");

  strcpy (xyz_filename, buff→H.fname);
  strcat (xyz_filename, ".xyz");
  xyz_file = OpenFile (xyz_filename, "w");

  fprintf (xyz_file, "%ld\n##\n", buff→H.natoms);

  for (i = 0; i < Natoms; i++)
    {

      fprintf (xyz_file, "%s %f %f %f\n", Atom_types[buff→atype[i]],
               buff→pos[i][0], buff→pos[i][1], buff→pos[i][2]);
    }

  fclose (xyz_file);

  Save_CVA2_Buffer_to_File (buff, xyz_filename);

}

void
Fill_CVA_Buffer (struct CVA_BUFFER *buff,
                 struct CVA_BUFFER *outbuff, int inormal, double s1, double s2)
{
/* slicing...*/
  int i, na = 0, j;
  double pos[3];

  /* first loop to count number of atoms in new buffer */
  for (i = 0; i < buff→H.natoms; i++)
    {
      pos[0] = buff→pos[i][0];
      pos[1] = buff→pos[i][1];
      pos[2] = buff→pos[i][2];

      if (pos[inormal] >= s1 && pos[inormal] <= s2)
        na++;
    }

  outbuff→pos = (double **) malloc (sizeof (double *) * na);
  for (i = 0; i < na; i++)
    outbuff→pos[i] = (double *) malloc (sizeof (double) * 3);

  j = 0;

  for (i = 0; i < buff→H.natoms; i++)
    {
      pos[0] = buff→pos[i][0];
      pos[1] = buff→pos[i][1];
      pos[2] = buff→pos[i][2];

      if (pos[inormal] >= s1 && pos[inormal] <= s2)
        {
          outbuff→pos[j][0] = pos[0];
```

```
            outbuff→pos[j][1] = pos[1];
            outbuff→pos[j][2] = pos[2];
            j++;
          }
      }

   outbuff→H.natoms = na;
   outbuff→H.bbox = Box_CVA_Buffer (outbuff);
   outbuff→H.box.x = outbuff→H.bbox.width[0];
   outbuff→H.box.y = outbuff→H.bbox.width[1];
   outbuff→H.box.z = outbuff→H.bbox.width[2];

}

void
Fill_CVA_Buffer_old (struct CVA_BUFFER *buff,
                     struct CVA_BUFFER *outbuff,
                     struct SLICE_BUFFER *sbuff, int s1, int s2)
{
   /*
      Reads in the data from an open buffer, called after 'Open_Buffer'

    */

   int i, is, j, k, na;
   na = 0;
   for (is = s1; is <= s2; is++)
      na += sbuff→slice[is].count;

   outbuff→pos = (double **) malloc (sizeof (double *) * na);

   for (i = 0; i < na; i++)
      outbuff→pos[i] = (double *) malloc (sizeof (double) * 3);

   j = 0;
   for (is = s1; is <= s2; is++)
      {

         for (i = 0; i < sbuff→slice[is].count; i++)
            {

               k = sbuff→slice[is].iatom[i];

               outbuff→pos[j][0] = buff→pos[k][0];
               outbuff→pos[j][1] = buff→pos[k][1];
               outbuff→pos[j][2] = buff→pos[k][2];
               j++;

            }
      }
   outbuff→H.natoms = na;
   outbuff→H.bbox = Box_CVA_Buffer (outbuff);
   outbuff→H.box.x = outbuff→H.bbox.width[0];
   outbuff→H.box.y = outbuff→H.bbox.width[1];
   outbuff→H.box.z = outbuff→H.bbox.width[2];

}

void
Read_CVA_Buffer_File (struct CVA_BUFFER *buff)
{
   /*
      Reads in the data from an open buffer, called after 'Open_Buffer' */

   int i, Natoms;

   if (buff→H.fhandle == NULL)
      error ("in Read_CVA_Buffer_File:  <file handle not available>");

   Natoms = buff→H.natoms;
   buff→pos = (double **) malloc (sizeof (double *) * Natoms);
   for (i = 0; i < Natoms; i++)
      buff→pos[i] = (double *) malloc (sizeof (double) * 3);

   for (i = 0; i < Natoms; i++)
      {

         fscanf (buff→H.fhandle, "%lf %lf %lf",
                 &buff→pos[i][0], &buff→pos[i][1], &buff→pos[i][2]);
```

```
    }
  buff→H.bbox = Box_CVA_Buffer (buff);
}
void
Load_CVA_Buffer (struct CVA_BUFFER *buff, char *filename)
{
  Open_CVA_Buffer_File (buff, filename);
  Read_CVA_Buffer_File (buff);
  Close_CVA_Buffer_File (buff);
}
FILE *
OpenFile (char *filename, char *op)
{
  FILE *handle;
  handle = fopen (filename, op);
  if (handle == NULL)
    error ("An error has occured with file %s", filename);
  else
    return (handle);
}
double
Dist3v0n (double *v1, double *v2)
{
  /* adopted for vectors starting with index 0 */
  int k;
  double dist[3];
  for (k = 0; k < 3; k++)
    {
      dist[k] = v1[k] - v2[k];
    }
  return (sqrt (Dot_Product3v0 (dist, dist)));
}
void
Dist3v0 (double *v1, double *v2, double *dist) /* This on;y sutracts!! */
{
  /* adopted for vectors starting with index 0 */
  int k;
  for (k = 0; k < 3; k++)
    {
      dist[k] = v1[k] - v2[k];
    }
}
double
Dot_Product3v0 (double *v1, double *v2)
{
  /* adopted for vectors starting with index 0 */
  double res;
  res = v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2];
  return (res);
}
double
Delta_r_sqr (struct CVA_BUFFER *buff1, struct CVA_BUFFER *buff2)
{
  /* calculate \Sum_i (r(t)^2 -r(o)^2)/Natoms., where r(t) is given by buffer
     1, and r(0) by buffer 2.
     note: the result is returned normalized by the number of atoms Na. */
  int i, Natoms;
  double sum = 0.0, dr[3];
```

```
  if (buff1→H.natoms != buff2→H.natoms)
    error ("in Delta_r_sqr:  <Buffers don't match for operation>");

  Natoms = buff1→H.natoms;

  for (i = 0; i < Natoms; i++)
    {

      Dist3v0 (buff1→pos[i], buff2→pos[i], dr); /* dr = buff1 - buff2 */
      sum += Dot_Product3v0 (dr, dr);

    }
  return (sum / (double) Natoms);
}

double
Delta_r_sqr_region (struct CVA_BUFFER *buff1,
                    struct CVA_BUFFER *buff2, int n1, int n2)
{
  /* calculate differences, and accumulate result and take square */

  int i;
  double sum = 0.0, dr[3];
  /* Must have another error test in here! */
  if (buff1→H.natoms != buff2→H.natoms)
    error ("in Delta_r_sqr:  <Buffers don't match for operation>");

  for (i = n1; i < n2; i++)
    {

      Dist3v0 (buff1→pos[i], buff2→pos[i], dr); /* dr = buff1 - buff2 */
      sum += Dot_Product3v0 (dr, dr);

    }
  return (sum / (double) (n2 - n1 + 1));
}

double
Delta_r_sqr_slice (struct CVA_BUFFER *buff1, struct CVA_BUFFER *buff2,
                   struct SLICE_BUFFER *bslice, int islice)
{
  /* calculate differences, and accumulate result and take square */

  int i, iatom;
  double sum = 0.0, dr[3];
  /* Must have another error test in here! */
  if (buff1→H.natoms != buff2→H.natoms)
    error ("in Delta_r_sqr:  <Buffers don't match for operation>");

  if (islice >= bslice→number)
    error ("in Delta_r_sqr:  <islice too large, not available slice %d", islice);

  for (i = 0; i < bslice→slice[islice].count; i++)
    {

      iatom = bslice→slice[islice].iatom[i];
      Dist3v0 (buff1→pos[iatom], buff2→pos[iatom], dr); /* dr = buff1
                                                                - buff2 */
      sum += Dot_Product3v0 (dr, dr);

    }
  return (sum / (double) (bslice→slice[islice].count));
}

double
Delta_r_sqr_layer1 (struct CVA_BUFFER *buff1,
                    struct CVA_BUFFER *buff2, int n1, int n2)
{
  /* calculate differences, and accumulate result and take square */

  int i;
  double sum = 0.0, dr[3];
  /* Must have another error test in here! */
  if (buff1→H.natoms != buff2→H.natoms)
    error ("in Delta_r_sqr:  <Buffers don't match for operation>");

  for (i = n1; i < n2; i = i + 2)
    {
```

```
      Dist3v0 (buff1→pos[i], buff2→pos[i], dr); /* dr = buff1 - buff2 */
      sum += Dot_Product3v0 (dr, dr);

    }
  return (sum / (double) (n2 - n1 + 1));
}

struct BBOX
Box_CVA_Buffer (struct CVA_BUFFER *buff)
{
  /* find the 'actual' bounding box of the sample, not to confuse of the
     (fixed) initial Box, defined with the sample. */
  int i, k;
  struct BBOX bbox;

  for (k = 0; k < 3; k++)
    {
      bbox.llc[k] = bbox.urc[k] = buff→pos[0][k];

    }

  for (i = 0; i < buff→H.natoms; i++)
    {
      for (k = 0; k < 3; k++)
        {
          bbox.llc[k] = min (bbox.llc[k], buff→pos[i][k]);
          bbox.urc[k] = max (bbox.urc[k], buff→pos[i][k]);
        }
    }

  for (k = 0; k < 3; k++)
    {
      bbox.width[k] = (bbox.urc[k] - bbox.llc[k]);
    }

  bbox.volume = bbox.width[0] * bbox.width[1] * bbox.width[2];

  return (bbox);

} /* Box */

void
Slice_CVA_Buffer (struct CVA_BUFFER *buff, struct SLICE_BUFFER *bslice)
{
  /* Slices the buffer into slicenum equal slices, and returns an array of
     slicenum SLICE structures,each containing the atoms that belong to each
     aslice. */

  int i, islice, count = 0;

  /* check if buffer is ok for sliceing... */
  bslice→tag = buff→H.tag;

  bslice→width = buff→H.bbox.width[bslice→inorm] / bslice→number;
  fprintf (stdout,
           "in Slice_CVA_Buffer:  inorm =%d H.bbox.width = %f, number = %d\n",
           bslice→inorm, buff→H.bbox.width[bslice→inorm], bslice→number);

  fprintf (stdout, " width = %f \n", bslice→width);
  /* allocate the slice aray */

  bslice→slice =
    (struct ONE_SLICE *) calloc (bslice→number, sizeof (struct ONE_SLICE));

  for (i = 0; i < buff→H.natoms; i++)
    {

      /* to which slice the i'th atom goes to */
      islice =
        (int) ((buff→pos[i][bslice→inorm] -
                buff→H.bbox.llc[bslice→inorm]) / bslice→width);
      /* some (paranoia) checking! */
      if (islice >= bslice→number)
        {
          warning ("# in Slice_CVA_Buffer:  <islice on edge %d> \n", islice);
          islice = bslice→number - 1;
        }
      if (islice < 0)
```

```
            {
              warning ("# in Slice_CVA_Buffer:  <islice <0 %d> \n", islice);
              islice = 0;
            }

          count = bslice→slice[islice].count; /* short hand variable :- */

          /* now fill up list pointer to pos */
          if (count == 0)
            {
              bslice→slice[islice].iatom = (int *) calloc (1, sizeof (int *));
              bslice→slice[islice].iatom[0] = i;
            }
          else
            {

              bslice→slice[islice].iatom
                = (int *)
                realloc (bslice→slice[islice].iatom, sizeof (int *) * (count + 1));
              bslice→slice[islice].iatom[count] = i;
            }

          bslice→slice[islice].count++;
        }

} /* slice */

void
Check_Slice_CVA_Buffer (struct CVA_BUFFER *buff, struct SLICE_BUFFER *slbuff)
{
  int isl;

  fprintf (stdout, "# %-10s\n", "WIDTH");
  fprintf (stdout, "# %-10f\n", slbuff→width);
  fprintf (stdout, "# %-10s %-10s\n", "ISLICE", "COUNT");

  for (isl = 0; isl < slbuff→number; isl++)
    {
      fprintf (stdout, "# %-12d %-12d\n", isl, slbuff→slice[isl].count);
    }

  for (isl = 0; isl < slbuff→number; isl++)
    {
      fprintf (stdout, "# %-10s %-10d %-10s %-10d \n", "ISLICE ", isl,
               "positions", slbuff→slice[isl].count);

      /* for ( i=0; i < slbuff->slice[isl].count; i++) { for ( k=0; k<3; k++) {
         fprintf(stdout, " % 22.15E", buff->pos[ slbuff->slice[isl].iatom[i]
         ][k]);} fprintf(stdout,"\n"); } */
    }
}

/* Vis3D stuff goes here */

/* translate particles arround the origion */

void
Print_CVA_BBOX (struct BBOX *bbox)
{
  fprintf (stdout, "# Bounding Box:\n");
  fprintf (stdout,
           "# volume = %f\n# llc= %f %f %f\n# urc= %f %f %f\n# width = %f %f %f\n",
           bbox→volume,
           bbox→llc[0], bbox→llc[1], bbox→llc[2],
           bbox→urc[0], bbox→urc[1], bbox→urc[2],
           bbox→width[0], bbox→width[1], bbox→width[2]);

}

void
yzRefold_vect (float *v, struct CVA_HEADER *H)
{
  int k;
  float box_size[3];

  box_size[0] = H→box.x;
  box_size[1] = H→box.y;
  box_size[2] = H→box.z;
```

```
  for (k = 1; k < 3; k++)
    v[k] -= box_size[k] * rint (v[k] / box_size[k]);
}
void
Initialize_CVA ()
{
  BUFFERS_HANDLE = -1;
}
```

cva.h

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <stdarg.h>
#include "nrutil.h"
#include "mytypes.h"
#include <string.h>
#include "cva.h"

#define FALSE 0
#define TRUE 1
#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))

#define PI 3.14159265358979323846
#define k_B (8.617385E-5) /* Boltzmann constant */
#define TimeUint 5.288E-14 /* time unit in secs */

#define MAX_MODEL_NUMBER 4
#define BUFFSIZE 240

#define MAX_NUM_PARTICLES_IN_SLICE 5000

#define CONST_E 0
#define RESCALE_T 1
#define HENDERSON 2

typedef enum {solid=0, liquid, other} Atom_Code;

long int iseed=123;

int BoxFactor[4] = {1,1,1};
int BoxSizeBackup[4];


/* stat variables */
struct dstat Eppp, Ekpp, Tkpp, Vcm[4], Etot, Ektot;
double V_cm[4]; /* poten. kin. and T per particle. */
struct dstat GEppp, GEkpp, GTkpp, GVcm[4], GEtot, GEktot;
struct dstat urc[3], llc[3], Gurc[3], Gllc[3];


struct BBOX scell; /* simulation cell */

struct SLICING fss, caus, canus;


char *cva2FileName;
char *run_id;
char report_filename[200];
FILE *report_file;
char ET_filename[200];
FILE *ET_file;
char Vcm_filename[200];
FILE *Vcm_file;
char Box_filename[200];
FILE *Box_file;

char *title;

int ModelNumber;
int Natoms, laf, lal;
int Nliquid, Nsolid;
```

```
int *llist, *slist, *All_atoms_list;
int Np[3], Npr[3], Napp[3];
double Deltat, a_latt, dp[3];
int Norm;
char snorm[80];
double klatt[4];
double BoxSize[4], InvBoxSize[4];

double Trequested;
int scale_interval=1000000, scale_end = 0, thermostat=0, Potential=0;
long MDsteps, roll_interval=10, sla_roll_interval=10, begin_stat_sla;
long write_snap_interval =1000000;
long ts = 0, ts_counter = 0, ts_init=0;

double **deltar; /* displacement */
double **pos; /* positions */
double **vel; /* velocities */
double **acc; /* accelerations */
char **Atype; /* atom types */


double *ene_pot;
double *ene_kin;
double *coord; /* coordinations */
double *deru; /* U'(n_i) */
double virial; /* the virial to compute the pressure term */

#define MaxPairsPerAtom 100
int MaxListLength;
int *Advance, *Marker1, *Marker2, *List;
double **DisplaceList;
double Skin = 3.0; /* Angstroms */
int updateList = TRUE;


double DeltatSqr, halfDeltatSqr, halfDeltat;



/*
    Parameters and tables of the ab-initio derived glue potential for
    Aluminum [F. Ercolessi and James B. Adams, Europhysics Letters 26,
    583 (1994) ]. The tables are filled by calling externally defined
    subroutines v2, rh, uu, returning the actual potential.
*/

/* If running on AIX, SGI or similar OS need to define underscore...*/
#if underscore
#define v2 v2_
#define rh rh_
#define uu uu_
#endif


double Rcutoff = 0.555805441821810e+01, RcutoffSqr;
double RcutoffPlusSkin = 0.555805441821810e+01+3.0;
double Rmin;
double RsqMin;
double DeltaRsq;
double InvDeltaRsq;
double CoordMax;
double DeltaCoord;
double InvDeltaCoord;
#define TableSize 20001

double *PhiTab, *DPhiTab; /* phi(r),1/r dphi/dr */
double *RhoTab, *DRhoTab; /* rho(r),1/r drho/dr */
double *UTab, *DUTab; /* U(n), du/dn */

void Define_Aluminium_Potential_Tables(void);

extern void v2(double *, double *, double *, double *);
extern void rh(double *, double *, double *, double *);

#define Max_Num_Vars 100 /* Maximunm number of variables allowed in control file */
char Name_List [ Max_Num_Vars ][40];
char Value_List [ Max_Num_Vars ][40];
char Type_List [ Max_Num_Vars ][5];
```

```
int Name_List_Size;
char *Get_s_Variable ( char *);
double Get_d_Variable ( char *);
int Get_i_Variable ( char *);

void Init_fcc ( void);
void Read_Initial_Configuration( void );
void WriteSnapShot(void);
FILE *OpenFile( char *, char *);
void warning(char *, ...);
void error(char *, ...);
void report(char *, ...);
void pr_stderr(char *, ...);
void usage(void);
void Read_vel ( FILE * );
void Read_acc ( FILE *);
void Allocate_Mem ( void );
void Update_NeighborList(double );

double Dot_Product_3v( double *, double *);
void Dist3v( double *, double *, double *);
void Refold( double * );
void VVEA_T ( int);
void VVEA_E ( int);
void VV_Const_T_step_1( int *, int , double );
void VV_Const_E_step_1( int *, int );
void VV_step_2(int *, int);
void Compute_Forces(void);
void Update_DisplaceList( void);
int MovedTooMuch();
void Compute_Eppp (int *, int, double *);
double Compute_Ek_tot( int *, int);
double Compute_Etot();
void Compute_Ek_T_Vcm ( int *, int, double *, double *);
double Compute_T(int *, int);
void Compute_Sk(int *, int, double *);
void Report_Part_1 ( );
void Report_Part_2 ( );
void Read_Control ( char *);
void Load_Control_File ( char *);

double Sum1v(double *, int *, int);
void Accumulate(struct dstat *, double );
void Set_Evolution_Control ();
void Stat ( void );
void Average ( struct dstat * );
void Stat_Sla ( int );
void Init_velocity(int * , int);
void Null_Vel(int *, int );
void CalVelCM ( int *, int , double *);
void TransVel2CM ( int *, int );
void TransPos2CM ( void );
void gran(double *);
struct BBOX Get_Box( );
void Set_fine_scale_slice ( );
void Fill_Slice ( struct SLICING *s);
int Load_xyz_data ( char xyzFileName[BUFFSIZE] );
void Init_Model ( );
int Input_cva2_xyz_part( FILE * cva2File );
void Define_Atom_Lists();
int Load_acc ( FILE *cva2File);
int Load_vel ( FILE *cva2File);
int AtomType ( char *type);
void WriteSnapShot_CVA2();
```

CVAGLOBAL.h

```
/* buffers external variable */
int BUFFERS_HANDLE; /* -1 indicate no buffers currently and for a
                       buffer, a negative tag indicate it was
                       deleted should always be updated before the
                       assignment*/
int Natoms,const_E_atoms, const_T_atoms;
unsigned int Quiet=0;
/*********************************************/
```

aluminum.f The following Fortran routine was kindly provided by Dr. Furio Ercolessi.

```
      SUBROUTINE v2(arg,func,dfunc,d2func)
* Aluminum : pair potential and its first two derivatives.
* Generated automatically by PoCo, version 04-may-93
* Hamiltonian type #. 2 run on 93/06/09 at 15.04.43
* Uses subroutine seval from netlib@ornl.gov [to get it,
* use 'send seval from sfmm'], trivially modified to
* compute also dfunc and d2func and use double precision.
      implicit double precision (a-h,o-z)
      parameter (nv2= 17)
      parameter (argmax= .555805441821810D+01)
      double precision xv2(nv2),yv2(nv2),bv2(nv2),cv2(nv2),dv2(nv2)
      save xv2,yv2,bv2,cv2,dv2
      data xv2( 1) / .202111069753385D+01 /
      data xv2( 2) / .227374953472558D+01 /
      data xv2( 3) / .252638837191732D+01 /
      data xv2( 4) / .277902720910905D+01 /
      data xv2( 5) / .303166604630078D+01 /
      data xv2( 6) / .328430488349251D+01 /
      data xv2( 7) / .353694372068424D+01 /
      data xv2( 8) / .378958255787597D+01 /
      data xv2( 9) / .404222139506771D+01 /
      data xv2( 10) / .429486023225944D+01 /
      data xv2( 11) / .454749906945117D+01 /
      data xv2( 12) / .480013790664290D+01 /
      data xv2( 13) / .505277674383463D+01 /
      data xv2( 14) / .530541558102636D+01 /
      data xv2( 15) / .555805441821810D+01 /
      data xv2( 16) / .555807968210182D+01 /
      data xv2( 17) / .555810494598553D+01 /
      data yv2( 1) / .196016472197158D+01 /
      data yv2( 2) / .682724240745344D+00 /
      data yv2( 3) / .147370824539188D+00 /
      data yv2( 4) / -.188188235860390D-01 /
      data yv2( 5) / -.576011902692490D-01 /
      data yv2( 6) / -.519846499644276D-01 /
      data yv2( 7) / -.376352484845919D-01 /
      data yv2( 8) / -.373737879689433D-01 /
      data yv2( 9) / -.531351030124350D-01 /
      data yv2( 10) / -.632864983555742D-01 /
      data yv2( 11) / -.548103623840369D-01 /
      data yv2( 12) / -.372889232343935D-01 /
      data yv2( 13) / -.188876517630154D-01 /
      data yv2( 14) / -.585239362533525D-02 /
      data yv2( 15) / .000000000000000D+00 /
      data yv2( 16) / .000000000000000D+00 /
      data yv2( 17) / .000000000000000D+00 /
      data bv2( 1) / -.702739315585347D+01 /
      data bv2( 2) / -.333140549270729D+01 /
      data bv2( 3) / -.117329394261502D+01 /
      data bv2( 4) / -.306003283486901D+00 /
      data bv2( 5) / -.366656699104026D-01 /
      data bv2( 6) / .588330899204400D-01 /
      data bv2( 7) / .384220572312032D-01 /
      data bv2( 8) / -.390223173707191D-01 /
      data bv2( 9) / -.663882722510521D-01 /
      data bv2( 10) / -.312918894386669D-02 /
      data bv2( 11) / .590118945294245D-01 /
```

```
      data bv2( 12) / .757939459148246D-01 /
      data bv2( 13) / .643822548468606D-01 /
      data bv2( 14) / .399750987463792D-01 /
      data bv2( 15) / .177103852679117D-05 /
      data bv2( 16) / -.590423369301474D-06 /
      data bv2( 17) / .590654950414731D-06 /
      data cv2( 1) / .877545959718548D+01 /
      data cv2( 2) / .585407125495837D+01 /
      data cv2( 3) / .268820820643116D+01 /
      data cv2( 4) / .744718689404422D+00 /
      data cv2( 5) / .321378734769888D+00 /
      data cv2( 6) / .566263292669091D-01 /
      data cv2( 7) / -.137417679148505D+00 /
      data cv2( 8) / -.169124163201523D+00 /
      data cv2( 9) / .608037039066423D-01 /
      data cv2( 10) / .189589640245655D+00 /
      data cv2( 11) / .563784150384640D-01 /
      data cv2( 12) / .100486298765028D-01 /
      data cv2( 13) / -.552186092621482D-01 /
      data cv2( 14) / -.413902746758285D-01 /
      data cv2( 15) / -.116832934994489D+00 /
      data cv2( 16) / .233610871054729D-01 /
      data cv2( 17) / .233885865725971D-01 /
      data dv2( 1) / -.385449887634130D+01 /
      data dv2( 2) / -.417706040200591D+01 /
      data dv2( 3) / -.256425277368288D+01 /
      data dv2( 4) / -.558557503589276D+00 /
      data dv2( 5) / -.349316054551627D+00 /
      data dv2( 6) / -.256022933201611D+00 /
      data dv2( 7) / -.418337423301704D-01 /
      data dv2( 8) / .303368330939646D+00 /
      data dv2( 9) / .169921006301015D+00 /
      data dv2( 10) / -.175759761362548D+00 /
      data dv2( 11) / -.611278214082881D-01 /
      data dv2( 12) / -.861140219824535D-01 /
      data dv2( 13) / .182451950513387D-01 /
      data dv2( 14) / -.995395392057973D-01 /
      data dv2( 15) / .184972909229936D+04 /
      data dv2( 16) / .362829766922787D+00 /
      data dv2( 17) / .362829766922787D+00 /
      if (arg.ge.argmax) then
          func = .000000000000000D+00
          dfunc = 0.d0
          d2func = 0.d0
      else
          call seval(nv2,arg,xv2,yv2,bv2,cv2,dv2,func,dfunc,d2func)
      endif
      end

      SUBROUTINE rh(arg,func,dfunc,d2func)
* Aluminum : atomic density and its first two derivatives.
* Generated automatically by PoCo, version 04-may-93
* Hamiltonian type # 2, run on 93/06/09 at 15.04.43
* Uses subroutine seval from netlib@ornl.gov [to get it,
* use 'send seval from sfmm'], trivially modified to
* compute also dfunc and d2func and use double precision.
      implicit double precision (a-h,o-z)
      parameter (nrh= 17)
      parameter (argmax= .555805441821810D+01)
      double precision xrh(nrh),yrh(nrh),brh(nrh),crh(nrh),drh(nrh)
      save xrh,yrh,brh,crh,drh
      data xrh( 1) / .202111069753385D+01 /
      data xrh( 2) / .227374953472558D+01 /
      data xrh( 3) / .252638837191732D+01 /
      data xrh( 4) / .277902720910905D+01 /
      data xrh( 5) / .303166604630078D+01 /
      data xrh( 6) / .328430488349251D+01 /
      data xrh( 7) / .353694372068424D+01 /
      data xrh( 8) / .378958255787597D+01 /
      data xrh( 9) / .404222139506771D+01 /
      data xrh( 10) / .429486023225944D+01 /
```

```
data xrh( 11) / .454749906945117D+01 /
data xrh( 12) / .480013790664290D+01 /
data xrh( 13) / .505277674383463D+01 /
data xrh( 14) / .530541558102636D+01 /
data xrh( 15) / .555805441821810D+01 /
data xrh( 16) / .555807968210182D+01 /
data xrh( 17) / .555810494598553D+01 /
data yrh( 1) / .865674623712589D-01 /
data yrh( 2) / .925214702944478D-01 /
data yrh( 3) / .862003123832002D-01 /
data yrh( 4) / .762736292751052D-01 /
data yrh( 5) / .606481841271735D-01 /
data yrh( 6) / .466030959588197D-01 /
data yrh( 7) / .338740138848363D-01 /
data yrh( 8) / .232572661705343D-01 /
data yrh( 9) / .109046405489829D-01 /
data yrh( 10) / .524910605677597D-02 /
data yrh( 11) / .391702419142291D-02 /
data yrh( 12) / .308277776293383D-02 /
data yrh( 13) / .250214745349505D-02 /
data yrh( 14) / .147220513798186D-02 /
data yrh( 15) / .000000000000000D+00 /
data yrh( 16) / .000000000000000D+00 /
data yrh( 17) / .000000000000000D+00 /
data brh( 1) / .608555214104682D-01 /
data brh( 2) / -.800158928716306D-02 /
data brh( 3) / -.332089451111092D-01 /
data brh( 4) / -.521001991705069D-01 /
data brh( 5) / -.618130637429111D-01 /
data brh( 6) / -.529750064268036D-01 /
data brh( 7) / -.442210477548108D-01 /
data brh( 8) / -.473645664984640D-01 /
data brh( 9) / -.390741582571631D-01 /
data brh( 10) / -.101795580610560D-01 /
data brh( 11) / -.318316981110289D-02 /
data brh( 12) / -.281217210746153D-02 /
data brh( 13) / -.236932031483360D-02 /
data brh( 14) / -.683554708271547D-02 /
data brh( 15) / -.638718204858808D-06 /
data brh( 16) / .212925486831149D-06 /
data brh( 17) / -.212983742465787D-06 /
data crh( 1) / -.170233687052940D+00 /
data crh( 2) / -.102317878901959D+00 /
data crh( 3) / .254162872544396D-02 /
data crh( 4) / -.773173610292656D-01 /
data crh( 5) / .388717099948882D-01 /
data crh( 6) / -.388873819867093D-02 /
data crh( 7) / .385388290924526D-01 /
data crh( 8) / -.509815666327127D-01 /
data crh( 9) / .837968231208082D-01 /
data crh( 10) / .305743500420042D-01 /
data crh( 11) / -.288110886134041D-02 /
data crh( 12) / .434959924771674D-02 /
data crh( 13) / -.259669459714693D-02 /
data crh( 14) / -.150816117849093D-01 /
data crh( 15) / .421356801161513D-01 /
data crh( 16) / -.842575249165724D-02 /
data crh( 17) / -.843267014952237D-02 /
data drh( 1) / .896085612514625D-01 /
data drh( 2) / .138352319847830D+00 /
data drh( 3) / -.105366473134009D+00 /
data drh( 4) / .153300619856764D+00 /
data drh( 5) / -.564184148788224D-01 /
data drh( 6) / .559792096400504D-01 /
data drh( 7) / -.118113795329664D+00 /
data drh( 8) / .177827488509794D+00 /
data drh( 9) / -.702220789044304D-01 /
data drh( 10) / -.441413511810337D-01 /
data drh( 11) / .954024354744484D-02 /
```

```
      data drh( 12) / -.916498550800407D-02 /
      data drh( 13) / -.164726813535368D-01 /
      data drh( 14) / .754928689733184D-01 /
      data drh( 15) / -.667110847110954D+03 /
      data drh( 16) / -.912720300911022D-01 /
      data drh( 17) / -.912720300911022D-01 /
      if (arg.ge.argmax) then
          func = .000000000000000D+00
          dfunc = 0.d0
          d2func = 0.d0
      else
          call seval(nrh,arg,xrh,yrh,brh,crh,drh,func,dfunc,d2func)
      endif
      end

      SUBROUTINE uu(arg,func,dfunc,d2func)
* Aluminum : glue function and its first two derivatives.
* Generated automatically by PoCo, version 04-may-93
* Hamiltonian type # 2, run on 93/06/09 at 15.04.43
* Uses subroutine seval from netlib@ornl.gov [to get it,
* use 'send seval from sfmm'], trivially modified to
* compute also dfunc and d2func and use double precision.
      implicit double precision (a-h,o-z)
      parameter (nuu= 13)
      parameter (argmin= .000000000000000D+00)
      double precision xuu(nuu),yuu(nuu),buu(nuu),cuu(nuu),duu(nuu)
      save xuu,yuu,buu,cuu,duu
      data xuu( 1) / .000000000000000D+00 /
      data xuu( 2) / .100000000000000D+00 /
      data xuu( 3) / .200000000000000D+00 /
      data xuu( 4) / .300000000000000D+00 /
      data xuu( 5) / .400000000000000D+00 /
      data xuu( 6) / .500000000000000D+00 /
      data xuu( 7) / .600000000000000D+00 /
      data xuu( 8) / .700000000000000D+00 /
      data xuu( 9) / .800000000000000D+00 /
      data xuu( 10) / .900000000000000D+00 /
      data xuu( 11) / .100000000000000D+01 /
      data xuu( 12) / .110000000000000D+01 /
      data xuu( 13) / .120000000000000D+01 /
      data yuu( 1) / .000000000000000D+00 /
      data yuu( 2) / -.113953324143752D+01 /
      data yuu( 3) / -.145709859805864D+01 /
      data yuu( 4) / -.174913308002738D+01 /
      data yuu( 5) / -.202960321360630D+01 /
      data yuu( 6) / -.225202324967546D+01 /
      data yuu( 7) / -.242723053979436D+01 /
      data yuu( 8) / -.255171976467357D+01 /
      data yuu( 9) / -.260521638832322D+01 /
      data yuu( 10) / -.264397894381693D+01 /
      data yuu( 11) / -.265707884842034D+01 /
      data yuu( 12) / -.264564149400021D+01 /
      data yuu( 13) / -.260870604452106D+01 /
      data buu( 1) / -.183757286015853D+02 /
      data buu( 2) / -.574233124410516D+01 /
      data buu( 3) / -.236790436375322D+01 /
      data buu( 4) / -.307404645857774D+01 /
      data buu( 5) / -.251104850116555D+01 /
      data buu( 6) / -.196846462620234D+01 /
      data buu( 7) / -.154391254686695D+01 /
      data buu( 8) / -.846780636273251D+00 /
      data buu( 9) / -.408540363905760D+00 /
      data buu( 10) / -.286833282404628D+00 /
      data buu( 11) / -.309389414590161D-06 /
      data buu( 12) / .236958014464143D+00 /
      data buu( 13) / .503352368511243D+00 /
      data cuu( 1) / .830779120415016D+02 /
      data cuu( 2) / .432560615333001D+02 /
      data cuu( 3) / -.951179272978074D+01 /
      data cuu( 4) / .245037178153561D+01 /
      data cuu( 5) / .317960779258630D+01 /
```

```
      data cuu( 6) / .224623095704576D+01 /
      data cuu( 7) / .199928983630817D+01 /
      data cuu( 8) / .497202926962879D+01 /
      data cuu( 9) / -.589626545953876D+00 /
      data cuu( 10) / .180669736096520D+01 /
      data cuu( 11) / .106163236918694D+01 /
      data cuu( 12) / .130795086934864D+01 /
      data cuu( 13) / .135599267112235D+01 /
      data duu( 1) / -.132739501694005D+03 /
      data duu( 2) / -.175892847543603D+03 /
      data duu( 3) / .398738817043878D+02 /
      data duu( 4) / .243078670350231D+01 /
      data duu( 5) / -.311125611846847D+01 /
      data duu( 6) / -.823137069125319D+00 /
      data duu( 7) / .990913144440207D+01 /
      data duu( 8) / -.185388527186089D+02 /
      data duu( 9) / .798774635639692D+01 /
      data duu( 10) / -.248354997259420D+01 /
      data duu( 11) / .821061667205675D+00 /
      data duu( 12) / .160139339245701D+00 /
      data duu( 13) / .160139339245701D+00 /
      if (arg.le.argmin) then
          func = .000000000000000D+00
          dfunc = 0.d0
          d2func = 0.d0
      else
          call seval(nuu,arg,xuu,yuu,buu,cuu,duu,func,dfunc,d2func)
      endif
      end

      subroutine seval(n, u, x, y, b, c, d, f, df, d2f)
      implicit none
      integer n
      double precision u, x(n), y(n), b(n), c(n), d(n)
      double precision f, df, d2f
c
c this subroutine evaluates the cubic spline function
c
c seval = y(i) + b(i)*(u-x(i)) + c(i)*(u-x(i))**2 + d(i)*(u-x(i))**3
c
c where x(i) .lt. u .lt. x(i+1), using horner's rule
c
c if u .lt. x(1) then i = 1 is used.
c if u .ge. x(n) then i = n is used.
c
c input..
c
c n = the number of data points
c u = the abscissa at which the spline is to be evaluated
c x,y = the arrays of data abscissas and ordinates
c b,c,d = arrays of spline coefficients computed by spline
c
c if u is not in the same interval as the previous call, then a
c binary search is performed to determine the proper interval.
c
      integer i, j, k
      double precision dx
      data i/1/
      if ( i .ge. n ) i = 1
      if ( u .lt. x(i) ) go to 10
      if ( u .le. x(i+1) ) go to 30
c
c binary search
c
   10 i = 1
      j = n+1
   20 k = (i+j)/2
      if ( u .lt. x(k) ) j = k
      if ( u .ge. x(k) ) i = k
      if ( j .gt. i+1 ) go to 20
c
c evaluate spline
c
   30 dx = u - x(i)
      f = y(i) + dx*(b(i) + dx*(c(i) + dx*d(i)))
      df = b(i) + dx*(2.d0*c(i) + 3.d0*dx*d(i))
      d2f = 2.d0*c(i) + 6.d0*dx*d(i)
      return
```

```
end
```

References

[1] Bonissent A. and Abraham F. F. Application of perturbation theory to the crystal-melt interface. *J. Chem. Phys.*, 74(2):1306–1309, 1981.

[2] Oxtoby D. W. and Haymet A. D. J. A molecular theory of the solid-liquid interface. II. study of bcc crystal-melt interface. *J. Chem. Phys.*, 76(12):6262, 1982.

[3] Huisman W. J., Peters J. F., Zwanenburg M. J., de Vries S. A., Derry T. E., Aberanathy D., and Van der Veen J. F. Layering of a liquid metal in contact with a hard wall. *Nature*, 390:397–381, 1997.

[4] Huisman W. J. and van der Veen J. F. Modelling the atomic density across a solid-liquid interface. *Surface Science*, 402-404:866–870, 1998.

[5] Ercolessi F. and Adams J. B. Interatomic potentials from first-principles calculations: The force-matching method. *Europhys. Lett.*, 26(8):583–588, 1994.

[6] De Tolla F. Ercolessi F. and Tosatti E. Maximum overheating and partial wetting of non-melting crystal surfaces. *Phys. Rev. Lett.*, 74:3201, 1995.

[7] Hansen U., Vogl P., and Fiorentini V. Quasiharmonic versus exact surface free energies of Al: A systematic study employing a classical interatomic potential. *Phys. Rev. B.*, 60:5055, 1999.

[8] Stumpf R. and Scheffler M. Direct observation of a surface charge density wave. *Phys. Rev. B.*, 53:4958, 1996.

[9] Bernal J. D. A geometrical approach to the structure of liquids. *Nature*, 183:141, 1959.

[10] Abraham F. F. and Singh Y. The structure of a hard-sphere fluid in contact with a soft repulsive wall. *J. Chem. Phys.*, 67(5):2384–2385, 1977.

[11] Broughton J. Q. and Gilmer G. H. Molecular dynamics investigation of the crystal fluid interface: VI. excess surface free energies of crystal-liquid systems. *J. Chem. Phys.*, 84(10):5759, 1986.

[12] Chen E. T., Barnett R. N., and Landman U. Crystal-melt and melt-vapor interfaces of nickel. *Phys. Rev. B*, 40(2):924, 1989.

[13] Davidchack R. L. and Laird B. B. Direct calculation of the hard-sphere crystal/melt interfacial energy. *Phys. Rev. E.*, 85(22):4751, 2000.

[14] Reichert H., Klein O., Dosch H., Denk M., Honklmaki V., Lippmann T., and Reiter G. Observation of five-fold local symmetry in liquid lead. *Nature*, 408:839, 2000.

[15] Curtin W. A. Density-functional theory of the solid liquid interface. *Phys. Rev. Lett.*, 59(11):1228, 1987.

[16] Choudhury N. and Ghosh S. K. Adsorption of Lennard-Jones fluid mixture in a planar slit: A perturbative density functional approach. *Phys. Rev. E*, 64:art. no. 021206, 2001.

[17] Haymet A. D. J. and Oxtoby W. A molecular theory for freezing: Comparison of theories, and results for hard spheres. *J. Chem. Phys.*, 84(3):1769, 1986.

[18] Swiler T. P. The role of liquid-substrate interactions on wetting in metallic embedded atom method systems. *Acta mater.*, 48:4775, 2000.

[19] Swiler T. P. and Loehman R. E. Molecular dynamics simulations of reactive wetting in metal-ceramic systems. *Acta mater.*, 48:4419, 2000.

[20] Geysermans P. and Gorse D. Molecular dynamics study of the solid-liquid interface. *J. Chem. Phys.*, 113(15):6382, 2000.

[21] Howe J. M. Direct observationof order in the liquid at a solid-liquid interface by high-resolution transmission electron microscopy. *Philosophical Magazine*, 74(3):761, 1996.

[22] Jackson K.A and Chalmers B. *Can. J. Phys.*, 34:473, 1956.

[23] Skapski A. N. *Acta. Met.*, 4:576, 1956.

[24] Turnbull D. In Prins J. A., editor, *Physics of non-crystalline solids*, page 46, 1964.

[25] Spaepen F. A structural model for the solid-liquid interface in monoatomic systems. *Acta Metallurgica*, 23:729, 1975.

[26] Bonissent A. and Mutaftschhiev B. A computer built random model simulation of the crystal-melt interface. *Philos. Mag.*, 35:65, 1977.

[27] Henderson D., Abraham F. F., and Barker J. The Ornstein-Zernike equation for a fluid in contact with a surface. *Molecular Physics*, 31(4):1291, 1976.

[28] Abraham F. F. and Singh Y. Comment on "The structure of hard sphere fluid in contact with a soft repulsive wall". *J. Chem. Phys.*, 68(10):4767–4768, 1978.

[29] Steele W. A. *The interfaces of gases with solid surfaces*, chapter 2. Pergamon, New York, 1974.

[30] Haymet A. D. J. and Oxtoby W. A molecular theory for the solid liquid interface. *J. Chem. Phys.*, 74(4):2559, 1981.

[31] Tarazona P. A model for density oscillations in liquids between solid walls. *Molecular Physics*, 56(3):557, 1985.

[32] Allen M. P. and Tildesley F. J. *Computer Simulation of Liquids.* Oxford University Press, 1987.

[33] Frenkel D. and Smit B. *Understanding Molecular Simulation, from Algorithms to Applications.* Academic Press, 1986.

[34] Finnis M. W. The theory of metal-ceramic interfaces. *J. Phys.: Condens. Matter*, 8:5811, 1996.

[35] Kyrlidis A. and Brown R. A. Density-functional theory and atomistic simulation of the hard-sphere melt-solid interface. *Phys. Rev. E*, 52(6):5832, 1995.

[36] Broughton J. Q., Bonissent A., and Abraham F. F. The fcc (111) and (100) crystal-melt interfaces, comparison by molecular dynamics simulation. *J. Chem. Phys.*, 74(7):4029, 1981.

[37] Broughton J. Q. and Gilmer G. H. Molecular dynamics investigation of the crystal fluid interface: I. bulk properties. *J. Chem. Phys.*, 79(10):5095, 1983.

[38] Broughton J. Q. and Gilmer G. H. Molecular dynamics investigation of the crystal fluid interface: II. structure of the fcc (111), (100), and (110) crystal-vapor systems. *J. Chem. Phys.*, 79(10):5105, 1983.

[39] Broughton J. Q. and Gilmer G. H. Molecular dynamics investigation of the crystal fluid interface: III. dynamical properties of fcc crystal-vapor systems. *J. Chem. Phys.*, 79(10):5119, 1983.

[40] Broughton J. Q. and Gilmer G. H. Molecular dynamics investigation of the crystal fluid interface: IV. free energies of crystal-vapor systems. *J. Chem. Phys.*, 84(10):5741, 1986.

[41] Broughton J. Q. and Gilmer G. H. Molecular dynamics investigation of the crystal fluid interface: V. structure and dynamics of crystal-melt systems. *J. Chem. Phys.*, 84(10):5749, 1986.

[42] Huitema H.E.A., Vlot M. J., and van der Erden J. P. Simulation of crystal growth from Lennard-Jones melt: detailed measurement of the crystal structure. *J. Chem. Phys.*, 111(10):4714, 1999.

[43] Rull L. F. and Toxvaerd S. The structure and thermodynamics of the solid-fluid interface. II. *J. Chem. Phys.*, 78(6):3273, 1983.

[44] Davidchack R. L. and Laird B. B. Simulation of the binary hard-sphere crystal/melt interface. *Phs. Rev. E.*, 54(6):R5905, 1996.

[45] Davidchack R. L. and Laird B. B. Simulation of the hard-sphere crystal/melt interface. *J. Chem. Phys*, 108(22):9452, 1998.

[46] Mori A., Manabe R., and Nishioka K. Construction and investigation of a hard-sphere crystal-melt interface by a molecular dynamics simulation. *Phs. Rev. E.*, 51(5):R3831, 1995.

[47] Spaepen F. Five-fold symmetry in liquids. *Nature*, 408:781, 2000.

[48] Rapaport D. C. *The Art of Molecular Dynamics Simulations*. Cambridge University Press, 1995.

[49] Andersen H. C. Molecular dynamics at constant pressure and/or temperature. *J. Chem. Phys.*, 72:2384, 1980.

[50] Hoover W. G. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A*, 31:1695–1697, 1985.

[51] Foiles S. M., Baskes M. I., and Daw M. S. Embedded-atom-method functions for the FCC metals Cu, Ag, Au, Ni, Pd, Pt, and their alloys. *Phys. Rev. B*, 33:7983, 1986.

[52] Finnis M. W. and Sinclair J. E. A simple empirical n-body potential for transition metals. *Philos. Mag. A*, 50:45, 1984.

[53] Jacobsen K. W., Norskov J. K., and Puska M. J. Interatomic interactions in the effective-medium theory. *Phys. Rev. B*, 35:7423, 1987.

[54] Lutsko J. F., Wolf D., Phillpot S. R., and S. Yip. Molecular-dynamics study of lattice-defect-nucleated melting in metals using an embedded-atom-method potential. *Phys. Rev. B.*, 40, 1989.

[55] Tomagnini O., Ercolessi F., Iarlori S., Di Tolla F. D., and Tosatti E. Role of layering oscillations at liquid metal surfaces in bulk recrystallization and surface melting. *Phys. Rev. Lett.*, 76(7):1118, 1996.

[56] Abraham F. F. The interfacial density profile of a Lennard-Jones fluid in contact with a (100) Lennard-Jones wall and its relationship to idealized fluid/wall systems: A Monte Carlo simulation. *J. Chem. Phys.*, 68(8):3713–3716, 1978.

[57] Press W. H. *Numerical recipes: The art of scientific computing*. Cambridge University Press, 1986.

קורולציות מבנה במשטח הביניים

נוזל-מוצק: מחקר אטומיסטי

אדהם חשיבון

קורולציות מבנה במשטח הביניים
נוזל-מוצק: מחקר אטומיסטי

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת תואר
דוקטור לפילוסופיה

אדהם חשיבון

הוגש לסנט הטכניון — מכון טכנולוגי לישראל

אב תשס"ב חיפה אוגוסט 2002

חיבור על מחקר נעשה בהדרכת ד"ר יוחנה אדלר, ד"ר ווין קפלן ופרופ'
סטיב ליפסון
בפקולטה לפיסיקה

הכרת תודה

ברצוני להודות לד"ר יוחנה אדלר ופרופ' סטיב ליפסון מהפקולטה לפיסי-
קה, ולד"ר ווין קפלן מהפקולטה להנדסת חומרים על הנחייתם המעולה,
התמיכה והעידוד במהלך המחקר.

הנני אסיר תודה לפרופ' Michael W. Finnis על שיתוף הפעולה הפורה.

הנני אסיר תודה לד"ר Furio Ercolessi על תרומת פוטנציאל האינטאקציה
של האלומיניום.

תודה גם לחבריי ועמיתיי הרבים בטכניון, במיוחד בקבוצה לפיסיקה
חישובית, ובקבוצה לחומרים קירמיים, על הסיוע והתמיכה.

תודה למשפחתי, אשר עמדה תמיד לצידי ותמכה בי.

אני מודה לקרן ליליאן ומילטון אדוארדס, משרד המדע ולקרן גרמניה
ישראל (GIF) על התמיכה הכספית הנדיבה בהשתלמותי

העבודה מוקדשת לאימי נאדיה.

תוכן עניינים

תקציר באנגלית 1

1 מבוא 4

2 המבנה במשטח הביניים נוזל-מוצק 7

2.1 מודלים אנליטיים 9

2.2 מחקרים חישוביים 16

2.3 מחקרים ניסיונאיים 22

3 מטרת המחקר 25

4 שיטות העבודה 27

4.1 שיטת הדינמיקה המולקולרית 27

4.2 האינטראקציות הבין אטומיות 29

4.3 תנאי השפה 34

5 תיאור מערכת הסימולציה 35

5.1 המבנה של הדגמים 37

5.2 העובי של המצע המוצק 39

5.3 הקונפיגורציה ההתחלתית 43

5.4 סימון הדגמים 44

5.5 מערכות עם מבנה קובי ממורכז פיאה 44

5.6 מערכות עם מבנה קובי ממורכז גוף 46

6 תיאור הסימולציות וחישובי גדלים 49

6.1 פרופיל הצפיפות 50

6.2 האינרגיה הפוטנציאלית 51

6.3 פרמטר המבנה הסטטי 51

6.4 פונקציית ההתפלגות הראדיאלית הדו-מימדית 52

6.5 תהליך ההרפיה 53

6.6 דגימת ערכים 55

7 תוצאות 57

7.1 תוצאות עבור דגמים ממורכזי פיאה קובית 57

7.1.1 איפיון כמותי של הסדר 66

7.1.2 השפעת גודל המערכת 70

7.1.3 סדר תוך משטחי 73

7.2 תוצאות עבור דגמים ממורכזי גוף קובים 75

7.2.1 סדר תוך משטחי 79

8 דיון בתוצאות 86

8.1 סדר תוך משטחי 94

8.2 המרחק בין השכבות ושינויי הצפיפות באיזור הביניים 97

9 סיכום ומסקנות 103

א שיטת הצפיפות הפונקציונאלית 106

ב תאור תוכנית המחשב 112

ב.1 117

רשימת מקורות 163

תקציר

מ

רשימת איורים

2.1 הקירובים השונים לאינטראקציה בין המוצק לנוזל כפי ששימשו את Bonissent ו- Abraham [1] . איור מתוך מקור [1] 11

2.2 השוואה בין פרופילי הצפיפות מהקירובים השונים של איטראקציית נוזל-מוצק כפי שהתקבלו על ידי Bonissent ו- Abraham [1] . איור מתוך מקור [1] 14

2.3 פרופילי הצפיפות הממוצעים עבור משטחי נוזל-מוצק במערכת bcc (100) ב- (a) ומערכת bcc (111) ב- (b) כפי שהתקבלו על ידי Haymat ו- Oxtoby. איור מתוך מקור [2] 15

2.4 עדות ניסיונית להיווצרום של שכבות נוזליות באיזור הביניים נוזל-מוצק כפי שהתקבלה על ידי Huisman ושות' [3, 4] . בתמונה העליונה מובא פרופיל הצפיפות האלקטרונית הממוצע בתוך הגאליום הנוזלי, ובתמונה התחתונה מובא תיאור סכימטי של המבנה האטומי המוצע. איור מתוך מקור [3] 24

4.1 שלושת הפונקציות המתארות את פוטנציאל האינטראקציה האופטימלי עבור אלומיניום של Ercolessi ו- Adams. איור מתוך מקור [5] 31

4.2 התנהגות בזמן של טיפה נוזלית מעל מצע מוצק מאותו החומר. הטיפה שמעל למשטח (110) מתפשטת באופן מוחלט אל תוך המצע, ואילו זו שמעל למשטח (111) אינה מתפשטת לגמרי. איור מתוך מקור [6] 33

5.1 תאור סכימטי של מערכת הסימולציה. העובי של המוצק צריך להיות יותר גדול מטווח האינטראקציה של הפוטנציאל $W_s > R_c$, בנוסף, מימדי החלק הנוזל צריכים להיות גדולים מספיק בכוון הניצב למשטח המפריד על מנת שהשפעת פני השטח החופשי של הנוזל לא תורגש באיזור הביניים נוזל-מוצק. 36

5.2 מבנה תאי יחידה סימטריים עבור סריג fcc ב- (a) וסריג bcc ב- (b). וקטורי היחידה בתרשים מוגדרים בנוסחאות 5.2 ו- 5.3 38

5.3 מבט מלמעלה על המשטח (100) של מצע fcc. שני המישורים האחרונים נראים בתרשים, המישור העליון בירוק, וזה שתחתיו באדום. 40

5.4 מבט מלמעלה על המשטח (111) של מצע fcc. שלושת המישורים האחרונים נראים בתרשים, המישור העליון בכחול, זה שאחריו באדום והשלישי בירוק. 40

5.5 מבט מלמעלה על המשטח (110) של מצע fcc. שני המישורים האחרונים נראים בתרשים, המישור העליון בירוק, וזה שתחתיו באדום. 41

5.6 מבט מלמעלה על המשטח (100) של מצע bcc. שני המישורים האחרונים נראים בתרשים, המישור העליון בירוק, וזה שתחתיו באדום. 42

5.7 מבט מלמעלה על המשטח (110) של מצע bcc. שני המישורים האחרונים נראים בתרשים, המישור העליון בירוק, וזה שתחתיו באדום. 42

6.1 תיאור החלוקה של הדגם לשכבות בניצב למשטח הביניים. כל שכבה הינה בעובי Δz. הגדלים השונים מחושבים בכל שכבה לחוד. 50

6.2 דוגמה לתהליך ההרפיה עבור המערכת fcc(100)-A (ראה טבלה 5.3). חוזרים על תהליך ההרפיה מספר פעמים עד שהמצב ההתחלתי והסופי זהים מבחינה סטטיסטית. 53

6.3 המספר הממוצע של האטומים $\langle N \rangle$ (בתרשים העליון), האינרגיה הפוטנציאלית הממוצעת לחלקיק u_p, האינרגיה הקינטית הממוצעת לחלקיק E_k, ופרמטר המבנה הסטטי $S(\mathbf{K})$ כפונקציה של המרחק מפני השטח של המוצק. 54

7.1 פרופיל הצפיפות הממוצעת $\rho(z)$ כפונקציה של המרחק מפני המוצק במערכת fcc(100)-C עבור שלוש טמפרטורות שונות: 1000K, 1200K, ו- 1400K. 58

7.2 כמו באיור 7.1 עבור משטח ביניים במערכת fcc (111)-B. 59

7.3 כמו באיור 7.1 עבור משטח ביניים במערכת fcc (110)-C. 60

7.4 השוואה בין פרופילי הצפיפות בכוונים השונים במערכות fcc בטמפרטורה של 1000K. הקווים האנכיים מתארים את מיקום השיאים במערכת עם משטח ביניים בכוון (100). d_p הינו המרחק בין השכבות כפי שמחושב מהשיאים, ואילו d_v כפי שמחושב מנקודות המינימה. 61

7.5 המרחק בין השכבות d_p, עבור מערכת fcc (100) ב- 1000K, 1200K ו- 1400K 62

7.6 כמו איור 7.5 עבור מערכת fcc (111). 63

7.7 כמו איור 7.6 עבור מערכת fcc (110). 64

7.8 המרחקים בין השכבות בכוונים השונים של מערכות עם מוצק בעל סימטרית fcc ב- 1000K. 64

7.9 המרחקים בין השכבות בכוונים השונים של מערכות עם מוצק בעל סימטרית fcc ב- 1000K כפי שמחושבים מהמינימה בפרופיל הצפיפות. 65

7.10 (a) המעטפת של פרופיל הצפיפות בדגם fcc עם משטח (111) ב- 1250K, ו- (b) בדיקה של ההתאמה האספוננציאלית. 68

7.11 פרמטר האי-סדר κ כפונקציה של הטמפרטורה עבור שלושת המערכות בעלי סימטרית fcc. 69

7.12 השוואה בין פרופילי הצפיפות עבור המערכות השונות עם משטח ביניים fcc (100) ב- 1100K. 70

7.13 השוואה בין פרופילי הצפיפות עבור המערכות השונות עם משטח ביניים fcc (111) ב- 1300K. 71

7.14 השוואה בין פרופילי הצפיפות עבור המערכות השונות עם משטח ביניים fcc (110) ב- 1000K. 72

7.15 פרמטר האי-סדר κ כפונקציה של הטמפרטורה עבור כל הדגמים עם מוצק בעל סימטריית fcc. 72
7.16 פונקציית ההתפלגות הרדיאלית הדו-מימדית עבור מערכת fcc (100) ב- $T = 1000K$. ההתפלגויות עבור השכבות השונות מסודרות אחת מעל השניה. 73
7.17 כמו איור 7.16 עבור מערכת fcc (111). 74
7.18 כמו איור 7.16 עבור מערכת fcc (110). 75
7.19 פרופיל הצפיפות הממוצעת $\rho(z)$ כפונקציה של המרחק מפני המוצק במערכת bcc (100) עבור טמפרטורות שונות. 76
7.20 כמו איור 7.19 עבור מערכת bcc (110). 77
7.21 פרופיל הצפיפות במערכת bcc (100), קרוב למוצק ישנה שכבה נוספת כתוצאה מהאטומים הנספחים. 78
7.22 איזור משטח הביניים במערכת bcc(100). בצד שמאל מובא מבט צדדי ובצד ימין מבט עילי על הסידור האטומי. השכבות השונות מסומנים בתרשים. . 79
7.23 המרחק בין השכבות בשתי המערכות bcc (100) ו- bcc (110). 80
7.24 פרמטר אי-הסדר κ עבור המערכות עם מצע bcc. 81
7.25 סידור השכבות במערכת bcc (100) באיזור הביניים. (a) מבט מלמעלה על מערכת ב- T=1000K, ו- (b) מבט מלמעלה (צד ימין) ומבט צדדי (צד שמאל) במערכת מקוררת ל- T=600K 83
7.26 פונקציית ההתפלגות הרדיאלית הדו-מימדית עבור מערכת bcc (100) (למעלה) ומערכת bcc (110) (למטה) ב-$T = 1000K$. ההתפלגויות עבור השכבות השונות מסודרות אחת מעל השניה. 84
7.27 איזור משטח הביניים במערכת bcc (110) ב- $T = 1000$K, משמאל מבט צדדי ומימין מבט עילי. במערכת זו אין ספיחה של אטומים מהנוזל אל תוך המוצק. 85

8.1 פרמטר האי-סדר κ כפונקציה של הטמפרטורה עבור כל המערכות. 87

8.2 פרמטר האי-סדר κ עבור הדגמים עם מוצק בעלי סימטריית fcc (111) ו-
(100) ואלה עם מוצק בעלי סימטריית bcc. 94
8.3 אי-הרטבה של המצע המוצק הצפוף על במערכת bcc(100)-C על ידי הנוזל. 96
8.4 הצפיפות הנפחית בכל שכבה ב- $T = 1000\mathrm{K}$. 99
8.5 הצפיפות המשטחית בכל שכבה ב- $T = 1000\mathrm{K}$. 100
8.6 המספר הממוצע של אטומים בכל שכבה ב- $T = 1000\mathrm{K}$. 102

א.1 האינטראקציה בין הנוזל לקיר בעל אינטראקציה רכה מסוג LJ 10-4 עבור
שני מקרים: (1) קיר מורכב משכבה אחת -ו- (2) משתי שכבות. 110
א.2 פרופיל הצפיפות של נוזל חסר אינטראקציה ליד קיר רך. 111

ב.1 דוגמה לקבצי הקלט xyz ו- cva2. 113
ב.2 דוגמה לקובץ הקלט של תוכנית הסימולציה 115

רשימת טבלאות

4.1 השוואה בין ערכים מחושבים לפי פוטנציאל Ercolessi-Adams וערכים מדוד-
ים ניסיונית, נתונים מתוך מקור [5] 32

5.1 רשימה של המצעים המוצקים וכיווניהם. המרחק בין המישורים d וה-
צפיפות המישורית $\eta = \rho d$, כאשר ρ הצפיפות הנפחית, עבור פרמטר סריג
$a_0 = 4.1$Å. 41

5.2 המספר הנדרש של מישורים במוצק בכל דגם, מחושב לפי פרמטר סריג
$a = 4.1$Å. 43

5.3 פרמטרי הדגמים בעלי משטח ביניים במקביל למישור fcc (100). תיאור
המערכות מובא בפרק 5.4. 45

5.4 כמו טבלה 5.3 עבור משטח ביניים במקביל למישור fcc (111). 45

5.5 כמו טבלה 5.3 עבור משטח ביניים במקביל למישור fcc (110). 46

5.6 פרמטרי הדגמים בעלי משטח ביניים במקביל למישור bcc (100). פרמטר
הסריג עבור המערכות A ו- B הוא 4.1Å $^\circ$ ואילו עבור מערכת C 2.05Å. . 48

5.7 כמו טבלה 5.6 עבור דגמים בעלי משטח ביניים מקביל למישורי bcc(110). 48

8.1 חישוב הגודל $\alpha = \kappa d^2$ עבור המערכות השונות. 92

8.2 אינרגית פני השטח האידיאלית עבור אלומיניום כפי שחושבו עם פוטנציאל
Ercolessi-Adams [7] ומ- LDA [8] 92

תקציר

לאיזור הביניים בין מוצק לנוזל יש משמעות רבה בתהליכים בעלי חשיבות טכנולוגית כגון התמצקות, גידול גבישים, תהליכי חיבור בנוכחות פאזה נוזלית, וסיכה. בשל כך, חקר משטחי הביניים בין נוזל למוצק זכה להתעניינות רבה. אולם, מבחינה ניסיונית, מכיוון שמשטח הביניים כלוא בין שתי פאזות צפופות, האינפורמציה לגבי המשטח אינה נגישה באמצעות רוב השיטות הניסיוניות הידועות. על כן, המידע הניסיוני אודות משטח הביני-ים מתקבל בדרך כלל ממדידות עקיפות. יש מספר מצומצם של מחקרים ניסיוניים אשר הצליחו לחדור אל המשטח המפריד בין נוזל למוצק באופן ישיר[3, 14, 21] . ממחקרים אלה עולה כי באיזור של משטח הביניים מופיעות שכבות מסודרות של הנוזל, כאשר הצפיפות האטומית בשכבות אלה מתאפיינת בתנודות מחזוריות עם משרעת היורדת ככל שמתרח-קים מפני המוצק. כלומר, הנוכחות של הפאזה המסודרת של הגביש משרה סדר בתוך הנוזל בכוון ניצב למשטח. בנוסף, נמצא שבתוך השכבה הנוזלית הראשונה ישנה תערובת של סדר גבישי, וסדר נוזלי בעל סימטריה מחומשת [14] , אולם באף אחד מהניסויים הנ"ל לא ניתן היה לקבוע את מידת הסדר בשכבות הבאות, ולא את תלות הסדר במבנה הגביש.

רוב המודלים התיאורטיים על משטח הביניים בין נוזל למוצק [1, 15–17] , מתיחסים למקרה ששתי הפאזות שייכות לאותו החומר ובשווי משקל תרמודינמי. מערכות כאלה מופיעות בעיקר בבעיות הקשורות לגידול גבישים או התכה. הבעיה העיקרית כאן היא לכלול באותו המודל התיאורטי את שינויי המבנה במעבר מנוזל למוצק. ממחקרים אלה עולה כי המעבר ממוצק לנוזל קורה בהדרגתיות תוך כדי היווצרותן של שכבות מסודרות באיזור הביניים, אשר צפיפותן יורדת באופן הדרגתי במעבר מהמוצק לנוזל. הסדר האטומי בתוך השכבות גם הוא דועך ככל שמתקרבים לנוזל, אולם עם אורך קורלציה קטן יותר.

משמעות הדבר היא היווצרותם של שכבות נוזליות באיזור הביניים נוזל-מוצק.

לעומת זאת, מספר המחקרים התיאורטיים המטפלים במשטח נוזל מוצק בין שני חומרים שונים הוא קטן מאוד [19, 20] . גם כאן התקבל שמופיעות תנודות מחזוריות בצפיפות האטומית הממוצעת בנוזל בניצב למשטח. הקושי העיקרי במודלים אלה הוא שבנוסף לשינויי המבנה בין שני צידי המשטח, על המודל לתאר גם את שינויי ההרכב הכימי, ואת השינוי באופי הקשרים בין החומרים, באיזור הביניים.

במחקרים אלה לא נבדק הקשר בין התנודות המחזוריות בצפיפות בתוך הנוזל לבין המבנה של המוצק הצמוד. כמו כן לא נבדקה הקורלציה בין מבנה פני המשטח של הנוזל והמבנה הנוצר בתוך השכבות הנוזליות. שאלה מענינת נוספת, היא כיצד ישתנה הסדר בתוך הנוזל כפונקציה של הטמפרטורה. לשאלות אלה השפעה על ההבנה הבסיסית של תהליכי הרטבה.

במחקר זה, נבנה מודל מופשט של משטח ביניים נוזל מוצק בין שני חומרים שונים. הנוזל מורכב מאלומיניום, ואילו המוצק מתואר על ידי אוסף של אטומים המקובעים במבנה סטטי קבוע בעל סימטרייה מסוימת. זהו מודל כללי של משטח נוזל מוצק במערכת בה החומר המוצק הוא הרבה יותר קשיח מהחומר הנוזל, בדומה למצב במערכות מתכת-קרמיקה למשל. האינטראקציה בין אטומי האלומיניום מתוארת על ידי פוטנציאל Ercolessi-Adams (EA) [5] , אשר נמצא כי הוא מתאר את התכונות של המתכת בצורה טובה מאוד, גם בפאזה הנוזלית וגם קרוב לפני שטח. בנוסף, האינטראקציה בין הנוזל למוצק מתוארת גם כן באמצעות פוטנציאל (EA) , כלומר האינטראקציה בין הנוזל למוצק היא בעלת אופי מתכתי, אלא שאטומי המוצק אינם זזים בהשפעת פוטנציאל זה.

שיטת הדינמיקה המולקולרית שימשה ככלי לסימולציה של משטחי הביניים בטמפרטורות שונות. דגם הסימולציה הורכב ממצע מוצק בעל מבנה מוגדר, כאשר לידו הונח נוזל בטמפרטורות שונות, מעל ל- 940K שהיא טמפרטורת ההתכה של האלומיניום, לפי פוטנציאל (EA) . במהלך הסימולציה נמדד פרופיל הצפיפות בתוך הנוזל כפונקציה של המרחק מפני המוצק, כמו כן נמדד הסדר בתוך כל שכבה באמצעות פונקציית ההתפלגות הרדיאלית הדו-מימדית (RDF).

שני מבנים בסיסיים שימשו בבניית המוצק. מבנה קובי ממורכז פיאה (face-centered-cubic (fcc)) בשלושה כיוונים שונים, (100), (111), ו- (110). ומבנה קובי ממורכז גוף (body-centered cubic (bcc)) בשני כיוונים שונים: (100), ו- (110). קבוע הסריג היה $a = 4.1$ Å שהוא ערך המתאים לצפיפות קרוב לשיווי משקל בין נוזל למוצק של אלומיניום. בנוסף, נבדקה ההשפעה של צפיפות המוצק בדגם (100) (bcc) בעל פרמטר סריג של $a = 2.05$ Å. כמו כן, נבדקה ההשפעה של שינוי עובי המוצק על חוזק האינטראקציה בין נוזל למוצק.

נמצא כי פרופיל הצפיפות בנוזל בניצב למשטח המוצק, מאופיין בתנודות מחזוריות הדועכות אל הצפיפות האחידה של הנוזל, כאשר קצב הדעיכה עולה עם הטמפרטורה, ומספר התנודות, כלומר השכבות, יורד. בטמפרטורה של $T = 1000$ K ישנם בין 10 ל- 9 שכבות בתוך איזור הביניים. בטמפרטורה של $T = 1400$ K ישנם בין 4 ל- 5 שכבות באיזור הביניים. עבור הדגמים בעלי מוצק עם סימטריית (fcc) המחזוריות של התנודות בנוזל זהה לזו שבתוך המוצק עבור הכוונים (111) ו- (110). ואילו עבור מוצק בכוון (100), המחזוריות זהה לזו שבמוצק עבור השכבות הקרובות (מספר השכבות תלוי בטמפרטורה), אולם ככל שמתרחקים מפני המוצק המחזוריות שואפת לזו שבנוזל עם מוצק בכוון (111). סיבה זו נעוצה בכך שאורך המחזור של ההפרעות המועדפות בתוך נוזל מתכתי, קרוב למרחק בין מישורים בכוון (111). מעבר דומה לא ניצפה בכוון fcc (110) מכיוון שהמרחק בין המישורים קטן מדי מכדי שמעבר כזה יהיה כדאי מבחינה אנרגטית.

הסידור האטומי בתוך השכבות במשטח הביניים עם המוצקים בעלי סימטריית fcc הוא בהתאמה לסידור של המצע המוצק. הסדר בשכבות קטן ככל שמתרחקים מהמוצק, עד שמתקיימות שכבות נוזליות לחלוטין. עבור דגם fcc (110) המעבר משכבות בעלי סדר לשכבות ללא סדר קורה לאחר 7-8 שכבות. עבור דגם עם מוצק fcc (111) ו- (100) המעבר קורה לאחר 4-5 שכבות.

עבור הדגמים בעלי מוצק עם סימטריית (bcc) (100) ו- (110), המחזוריות של התנודות בנוזל אינה קשורה לזו שבתוך המוצק. בשני המקרים המחזוריות עוברת מהר מאוד (לאחר שכבה אחת) למחזוריות קרובה מאוד לזו שבמוצק מסודר fcc (111). בנוסף, במערכת עם מצע bcc (100) מתרחשת תופעה של ספיחה של אטומים מהנוזל אל האתרים ממורכזי

הפיאה במישור bcc (100) . על ידי כך הופך פני השטח של המוצק לדמוי משטח fcc (100). הסדר התוך שכבתי שמתקבל בדגמים אלה דומה לסדר שבנוזל במערכת עם מוצק fcc (100). ואילו במערכת עם מוצק bcc (110) , לא נצפה שום סדר בתוך השכבות קרוב לפני המשטח.

איפיון כמותי של הסדר בניצב למשטח הביניים מתקבל מהאופי של הדעיכה של פרופיל הצפיפות. בהסתמך על תוצאות תיאורטיות [31] ומהצורה שמתקבלת עבור המעטפת של פרופיל הצפיפות בניסוי [3] ובמחקר זה, בוצעה התאמה של המעטפת לפי דעיכה אקספוננציאלית מהצורה:

$$\rho(z) = a\mathrm{e}^{-\kappa \mathrm{z}} + \mathrm{b}$$

כאשר $\rho(z)$ הינה הצפיפות הממוצעת במרחק z מהמשטח המוצק, κ, a ו- b הם פרמטרים שמתקבלים מההתאמה. הפרמטר κ הוא למעשה פרמטר אי-סדר, ככל שהפרופיל דועך לאט יותר, כלומר κ קטן יותר, הסדר נמשך מרחק גדול יותר אל תוך הנוזל.

מתקבל כי מקדם האי-סדר κ גדל מונוטונית עם הטמפרטורה, כאשר κ הגדול ביותר מתקבל עבור מערכת fcc (110) , בהתאמה עם אנרגית היצירה הגבוהה יחסית של משטח זה, וגם עם המחזוריות הקטנה בפרופיל הצפיפות. ההפרש בפרמטר האי-סדר κ עבור הדגמים עם מבנה משטח צפוף (closed packed) , קרי דגמים עם סימטריית fcc (111) ו- (100), וגם עבור המערכת bcc (100) עם המשטח הנספח, הוא קטן ובתחום השגיאה הסטטיסטית שמתקבלת מהמדידות. דבר זה קונסיסטנטי עם ההתנהגות של פרופילי הצפיפות, שכן המחזוריות בדגמים אלה זהה.

עבור הדגמים עם מוצק בעל משטח bcc (110) מתקבל כי בטמפרטורות נמוכות, $T < 1200$ K , האי-סדר הוא יותר גדול מבמשטחים הצפופים. ואילו בטמפרטורות גבוהות יותר האי-סדר במערכת זו קטן יותר. תופעה זו קשורה לאופי האינטראקציה בין הנוזל והמוצק. ככל שצפיפות המוצק קטנה יותר חוזק האינטראקציה גדול יותר, שכן הקואורדינציה של החלקיקים קרוב למוצק כזה נמוכה יותר מכפי שמתקיים בתוך הנוזל, מה שגורם לה-צטברות גדולה של הנוזל קרוב למשטח. השינוי ב- κ נובע מכך שהסדר בניצב למשטח קשור לסדר בתוך השכבות. עבור טמפרטורה נמוכה, השכבות יותר מסודרות ליד מוצק

בעל סימטריית fcc מאשר ליד מוצק בעל סימטריית bcc, מה שגורם ל- κ להיות יותר קטן עבור דגמים אלה. כאשר הטמפרטורה עולה הסדר התוך שכבתי קטן, ואילו האפקט של האינטראקציה נהיה יותר דומיננטי, מה שגורם ל- κ קטן יותר עבור הדגמים עם סימטריית bcc (110) . לסיכום, כאשר המבנה של המתכת והמוצק מתאימים, כלומר מבנה הנוזל במצב ההתמצקות קרוב למבנה המוצק, המבנה של הנוזל במשטח הביניים יהיה קרוב לזה במצב התמצקות. לעומת זאת כאשר ישנה אי התאמה במבנה, אולם קיימת הרטבה טובה בין הנוזל למוצק, יווצרו שכבות בתוך הנוזל ללא סדר בתוך השכבות.