POINT DEFECTS, LATTICE STRUCTURE AND MELTING

SLAVA SORKIN

POINT DEFECTS, LATTICE STRUCTURE AND MELTING

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN PHYSICS

SLAVA SORKIN

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

ELUL, 5765 HAIFA SEPTEMBER, 2005

THIS RESEARCH THESIS WAS SUPERVISED BY MY LOVELY ADVISORS DR. JOAN ADLER AND PROF. EMIL POLTURAK UNDER THE AUSPICES OF THE PHYSICS DEPARTMENT

ACKNOWLEDGMENT

I wish to express my gratitude to Dr. J. Adler and Prof. E. Polturak for the excellent guidance and support during this research.
I am grateful to Dr. A. Hashibon, Dr. G. Wagner and Dr. Z. Salman, and to my fellow students A. Kanigel, N. Schreiber and to my wife A. Sorkin for their help during the research period.
I would like to express my gratitude to J. Tall and M. Goldberg for the help on the IUCC machines and parallel computing. I also thank the IUCC for allowing me CPU time on their linux-cluster, the Cray T90 and Cray SV1 supercomputers of the Ben Gurion University, and the Silicon Graphics supercomputer of the Technion.

THE GENEROUS FINANCIAL HELP OF THE TECHNION AND THE GERMAN ISRAELI FOUNDATION (GIF) IS GRATEFULLY ACKNOWLEDGED

Contents

List of Figures

List of Tables

Abstract

Melting is a fundamental process in which a crystal undergoes a phase transition from a solid to a melt. Despite its common occurrence, understanding this process still a challenge.

A number of theories, which consider melting as a process occurring homogeneously throughout the crystal have been proposed during the past century. For example, according to Lindemann, melting is triggered by a mechanical instability of the solid, which caused by enhanced vibration of the atoms. Solids liquefy when the amplitude of atomic thermal vibrations exceeds some fraction of interatomic spacing. According to Born, melting arises from the onset of a mechanical instability of the crystal lattice, which manifests itself in an imaginary phonon frequency and the vanishing shear elastic moduli, accompanied by the collapse of the crystal lattice. Other models are based on spontaneous thermal production of the intrinsic lattice defects (vacancies, interstitials and dislocations) near the melting point and this leads to break-down of the long-range crystalline order and melting transition. However, the extrinsic defects (free surfaces and grain boundaries) were not considered as an important ingredient of a melting scenario. Those models are based on the concept of one-phase melting or continuous melting, i.e. they imply that the phase change might be continuous or nearly so; given sufficient resolution it should be possible to track the breakdown of the solid throughout its transition to the liquid state. These models are capable of calculate the melting temperature T_m, but in the most it is overestimated.

The existing theories of melting are still far from being complete and raise new questions. Hence, the purpose of the present research is to gain a better understanding of the mechanism of melting transition, and especially to investigate the rôle of point defects and the surface of the solid in the melting transition. Despite the fact that we learned very much recently about the melting of fcc metals, it is not clear if those results were specific to fcc structure of these metals, so we decided to study melting of a bcc metal, vanadium, by means of computer simulations.

An interatomic potential proposed by Finnis and Sinclair [42], was chosen for our simulations. The potential was tested by calculation of various properties of a perfect crystal of vanadium. The results are in good agreement with available experimental data. Afterwards point defects were introduced into the bulk either by the removal an atom (vacancy) or by the addition one (self-interstitial). The most stable configuration of defects at low temperatures was found to be a dumb-bell, the $< 110 >$ split-interstitial. Point defects change the physical properties of the solid. Interstitials expand the sample, while vacancies decrease its volume. The change of the volume is less noticeable for vanadium than for copper which is attributed to the less close-packed structure of its bcc lattice. We found also that the shear moduli are softened as a result of the volume expansion of the solid which is associated either with an increase in temperature or interstitial concentration. This softening of the moduli is less pronounced for vanadium in comparison with copper.

There is a strong evidence that Born instability is the trigger for bulk melting. The instability is set in by interstitials which expand the solid up to a critical volume, at which the lattice of the crystal becomes mechanically unstable and collapses. This defect-mediated mechanical melting occurs at the temperatures below the melting temperature of the perfect crystal. We verified that the critical volume at which the crystal melts is independent of the path thru the phase space by which it reached, i.e. either by heating the perfect crystal or by adding defects at a constant temperature. We performed simulations with various concentrations of point defects and found that bulk melting temperature T_b is lowered by interstitials, but this effect is less pronounced in comparison with the same effect for copper.

The mechanical melting can not be observed directly in the laboratory, because a real crystal will eventually melt at T_m which is lower than T_b via thermodynamic melting process that nucleates at its surface. The process can be suppressed experimentally if the surface is eliminated, for example, by coating one material with another one, with larger T_m. In this way silver coated by gold was superheated by 25K above T_m [93]. In computer experiments we are able to eliminate the surface using periodic boundary conditions in all directions and thus can investigate bulk melting.

In order to study surface melting we use periodic boundary conditions only in two directions and create a free surface in the third one. The thermodynamic melting temperature was found to be $T_m = 2220 \pm 10K$ by means of the method proposed by Lutchko et al. [77] (the bulk melting temperature of a perfect sample is $T_b = 2550 \pm 5$). Melting of crystals begins at the surface, because the activation energy for formation of a liquid phase is lower at the surface, than in the bulk. The liquid layer at the surface eliminates the barrier for nucleation of the liquid phase, and thus no metastability effects (superheating) exist.

Most of the theoretical models of surface melting phenomenological in nature, and therefore neglect the atomistic details of the phenomenon. Only recently the first microscopic theory has been proposed [36], which is capable of describing static properties of the rare-gas crystals. The microscopic description of surface melting phenomena emerged mainly from computer simulations.

We studied surface premelting of vanadium using molecular dynamics. The structural, transport and energetic properties of the various low-index surfaces of vanadium, namely Va(001), Va(011) and Va(111), at different temperatures were investigated. We found that upon increasing the temperature the vibrations of atoms at the surface region becomes so large, that they "disturb" each other. As a result, point defects are generated which begin to migrate between the surface layers and an adlayer appears on the top of the first surface layer. The disorder begins to spread from the topmost layer to a deeper ones. At higher temperatures a thin quasiliquid film appears in the surface region. The observed premelting phenomena are most pronounced in the surface region of the least packed Va(111) face, and is less noticeable for the closest packed Va(011) face. Similar results were obtained in simulations of fcc metals, where the least packed face (011) exhibits premelting, while the closest packed face (111) remains ordered almost up to the melting point.

In order to understand the relation between the bulk and surface melting, we applied the Born criterion of melting to the surface region and found a linear relation between the activation energy of surface defects and the melting temperature. This relation was confirmed by results of experiments and computer simulations of metals with fcc structure [92]. In order to test the model for metals with bcc structure, we calculated the activation energy of the surface defects for the least packed Va(111) and compared it with theoretical prediction. The agreement between the theory and simulations was found to be reasonable. A general conclusion was made that the

Born criterion correctly describes both surface and bulk melting, and may provide the “missing link” which will finally tie together these two scenarios for melting transitions.

List of symbols

T	temperature
P	pressure
V	volume
N	number of atoms
S	entropy
E	eenrgy
T_m	melting temperature
k_B	Boltzmann constant
ν_E	Einstein frequency
a_0	lattice constant
c_L	Lindemann constants
C_{44}	$< 100 >$ shear modulus
C'	$< 110 >$ shear modulus
H	Hamiltonian
Z_N	partition function
F	Helmholtz free energy
G_s	Gibbs free energy of a solid
G_l	Gibbs free energy of a liquid

x, y, z	the three Cartesian coordinates
$\alpha, \beta, ..$	Cartesian indeces
$\vec{r}_i$	position of atom i
$\vec{v}_i$	velocity of atom i
$\vec{f}_i$	force on atom i
n_i	electronic density around atom i
$s_{i,\alpha}$	scaled position of atom i in α direction
$\dot{s}_{i,\alpha}$	scaled velocity of atom i in α direction
$U,\ E_{pot}$	potential function
$K,\ E_{kin}$	kinetic energy
$\sigma_{\alpha\beta}$	stress tensor
$\epsilon_{\alpha\beta}$	strain tensor
$H_{\alpha\beta}$	matrix that spans the sample unit
$B_{\alpha\beta\gamma\delta}$	Born term
$C_{\alpha\beta}$	elastic coefficients
α_t	thermal expansion coefficient
$\rho(z)$	local density profile
$\eta_{l,\alpha}$	order parameter in α direction of layer l
$p_l(r_{\|\|})$	2D radial distribution function of layer l
$D_{l,\alpha}$	diffusion coefficient in α direction of layer l
$d_{i,i+1}$	distance between neighboring layers i and $i+1$
E_s	surface defect formation energy
E_f	split-interstitial defect formation energy
E_g	cohesive energy
FS	Finnis Sinclair potential
N_l	number of atoms of layer l
n_l	occupation number of layer l
v_{sl}	propagation velocity of a solid-to-liquid interface
$\bar{l}$	thickness of a quasiliquid layer

Chapter 1

Bulk melting

1.1 Preface

Of all the phenomena exhibited by condensed matter, changes of state are among the most dramatic, and of these melting and freezing are specially striking.

A melting transition occurs when a particular phase becomes unstable under a given set of thermodynamic conditions. Classical thermodynamics offers a sound framework for understanding of phase transition in term of a free energy. Usually the Gibbs free energy G is used. We relate it to a particular phase state of matter by a subscript, e.g. for a solid G_s and for a liquid G_l. When two states of matter are in thermodynamic equilibrium the Gibbs free energies are equal $G_s(P,T) = G_l(P,T)$. The free energy of the system is a continuous function of P and T during the phase transitions (See Fig. 1.1), but other thermodynamic quantities such as internal energy U entropy S volume V and heat capacity C undergo discontinuous changes. Almost all substances expand on melting, i.e. $\Delta V_m > 0$, ice being one of very few exceptions [19] (among them Sb, Bi, Ga). The behavior of ice is generally attributed to its

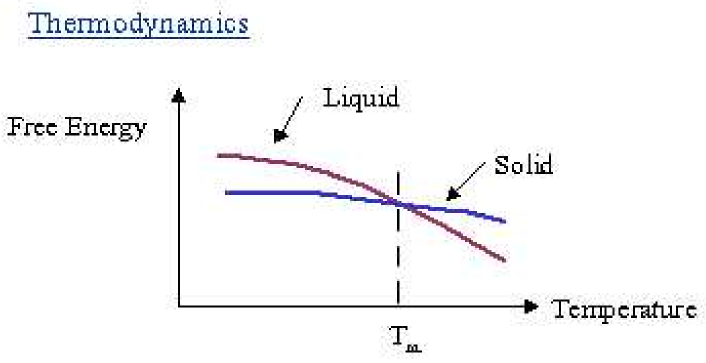


Figure 1.1: Variation of the Gibbs free energy of a simple atomic substance near the melting point as a function of temperature.

'open' network structure (it has a very small coordination number of the nearest neighbors), which collapses at the melting point, allowing atoms to adopt smaller average separation between them.

In all known materials (except He) the entropy of the liquid phase is higher than the entropy of the solid phase at the melting point:

$$\Delta S_m = Rln\left(\frac{W_l}{W_s}\right) \tag{1.1}$$

where ΔS_m is the entropy difference between the two phases, R is the gas constant and W_l is the number of independent ways of realizing the molten state and the W_s is the same quantity for the solid state. Therefore, the melting transition is a transition from an ordered state to a less ordered state, which increases the 'randomness' of the material structure.

However, the thermodynamic equations can not explain the mechanism of melting. The melting transition is determined by the detailed microscopic structure of the crystalline and melting state. Therefore, the process of melting can not be explained without knowledge of the structure of the material.

All theoretical models of melting can be divided into two groups. The first group is the group of models of ”two phases” (solid and melt). These models always involve reference to both phases, expressing the equilibrium between a solid and its melt in terms of the Gibbs free energy and calculating T_m. The second group is the group of ”one-phase” models, and consider melting as a homogeneous process occurring in the bulk of the solid. Some of those models will be considered in detail below. These theories focus on a certain type of lattice instabilities (anharmonic vibrations, anharmonic crystal elongation, vanishing of resistance to the shear stress, etc) and structural defects (vacancies, interstitials, dislocations, disclinations) which arise at particular range of temperatures and cause the solid to become unstable or “unrealizable” above the melting temperature. The bulk melting models usually overestimate the melting temperature. Real crystals, which are finite and always have boundaries, start to melt from the surface at a temperature which is lower than the temperature predicted by the theories of mechanical instability of the crystal lattice. Nevertheless, these theories play a very important rôle in our understanding of the mechanism of melting, and especially in emphasizing of the possible scenarios of melting which include point defects, dislocations, etc.

1.2 Lindemann criterion

The first theory explaining mechanism of melting in the bulk was proposed by Lindemann [1], who used vibration of atoms in the crystal to explain the melting transition. The average amplitude of thermal vibrations increases when the temperature of the solid increases. At some point the amplitude of vibration becomes so large that the atoms start to invade the space of their nearest neighbors and disturb them and the melting process initiates. Quantitative calculations based on the model are not easy, hence Lindemann offered a simple criterion: melting might be expected when the root mean vibration amplitude $\sqrt{< u^2 >}$ exceeds a certain threshold value (namely when the amplitude reaches at least 10% of the nearest neighbor distance).

Assuming that all atoms vibrate about their equilibrium positions with the same frequency ν_E (the Einstein approximation) the average thermal vibration energy can be estimated relying on the equipartition theorem as:

$$E = m4\pi^2\nu_E^2 < u^2 >= k_BT \tag{1.2}$$

where m is the atomic mass, ν_E is the Einstein frequency, $< u^2 >$ is the mean square thermal average amplitude of vibration, and T is absolute temperature. Using the Lindemann criterion for the threshold $< u^2 >= c_l a^2$, where c_l is Lindemann's constant one can estimate the melting point

$$T_m = 4\pi^2 m c_L a^2/k_B \tag{1.3}$$

Lindemann's constant c_l was assumed to be the same for crystals with similar structure, hence it could be calculated from the melting temperature of one particular crystal. A detailed experimental examination showed that c_l is not strictly a constant and the correlation is only fair (See Fig. 1.2). Recently, the validity of the Lindemann instability criterion have been tested in computer simulations of bulk melting of Lennard-Jones fcc crystals [6]. It has been found that melting occurs when a sufficiently large number spatially correlated destabilized atoms of the crystal (e.g. cluster of quasiliquid) are generated (See Fig. 1.3). These clusters are distributed homogeneously thruout the solid. The Lindemann criterion of the lattice instability is found to be valid for these clusters. The accumulation, growth and coalescence of the clusters of the liquid phase constitute, according to Jin et al. [6], the mechanism of

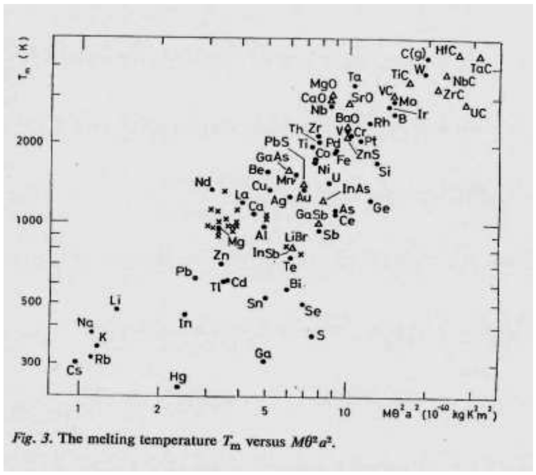


Figure 1.2: The measured melting temperature versus the melting temperature estimated using the Lindemann rule, from ref. [95]

homogeneous bulk melting. It should be stressed that the original Lindemann model for vibrational melting, like many of its more sophisticated successors, refers only to a crystal with the simplest possible structure, i.e. assemblies of closed packed atoms. Crystals containing more complex molecules as unit of structure exhibit a vibrational complexity which rules out any simple rule of lattice stability, determined merely by vibrational amplitudes of the molecular centers of mass. Futhermore, the Lindemann model is based on harmonic forces, which never give way, whereas melting must involve bond breaking. This is another serious defect of the model. Furthermore numerous experiments carried out at high pressures indicate that the Lindemann model does not estimate adequately the pressure dependence of the melting temperature [7]. The most serious defect of the model is that melting is described in term individual atomic property, i.e. mean square amplitude of vibration, while a phase transition is a cooperative process. In addition, the Lindemann model describes melting in terms of the solid alone, although the melting transition must involve both solid and liquid

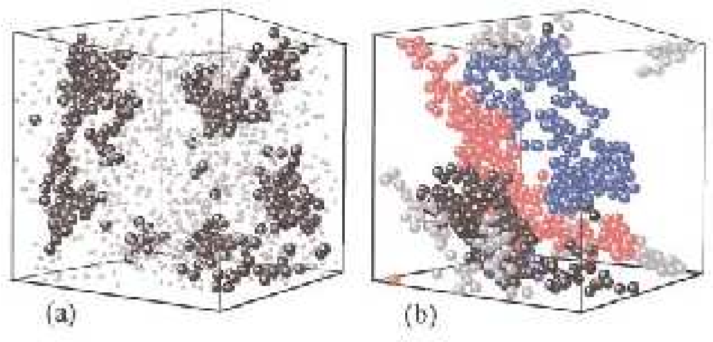


Figure 1.3: 3D visualization of the collective appearance of the Lindemann particles at $T/T_m = 0.79$. (a) a few clusters with 20-200 particles (larger black circles) against other Lindemann particles (smaller gray circles) which do not form such clusters (b) four large clusters with 219, 214, 187, and 117 particles colored with red, blue, black and gray, respectively. From ref. [6]

phases. Nevertheless the predictive success of the Lindemann melting criterion lent support to the belief that melting could be a gradual process, beginning within the solid at temperatures below the melting point. Subsequent theories and numerous experiments helped to bolster the idea.

1.3 The Born criterion

Another version of a “one-phase” theory of bulk melting suggested by Born [9] in 1937. His theory is based on the fact that a liquid differs from a crystal in having zero resistance to the shear stress. The distances between the atoms are increased due to thermal expansion, hence the restoring forces between the atoms are reduced, and therefore the shear elastic moduli decrease with rising temperature. The softening of the shear moduli leads to a mechanical instability of the solid structure and finally to a collapse of the crystal lattice at some temperature. The general conditions for stability of a crystal lattice were derived by Born. He analyzed the free energy of a solid with a cubic crystalline lattice. For a lattice to be stable, the free energy must be represented by a positive defined quadratic form, and this is fulfilled if the following inequalities for the shear elastic coefficients C_{11}, C_{12}, C_{44} are satisfied:

$$2C' = C_{11} - C_{12} > 0 \tag{1.4}$$

$$C_{44} > 0 \tag{1.5}$$

$$C_{11} + 2C_{12} > 0 \tag{1.6}$$

According to Born, C_{44} goes to zero first and the melting temperature can be found from the condition: $C_{44}(T_m) = 0$. Hunter and Siegal [10] measured the elastic coefficients of single crystal rods of NaCl over the temperature range from $20°C$ to $804°C$, up to the melting point. It was found that the shear elastic moduli C_{44} and C' decrease nearly linearly with temperature, but *reach non-zero values* at the melting point.(See Fig.1.4) The shear elastic moduli for a series of fcc metals were measured by Varishni

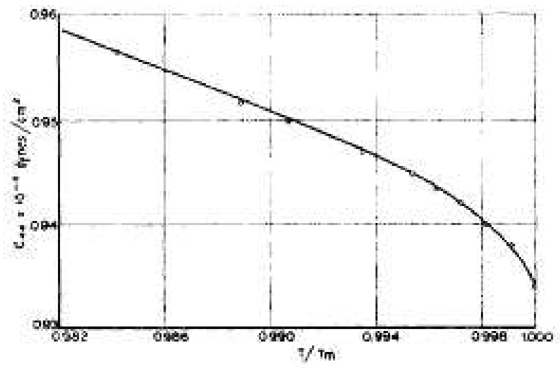


Figure 1.4: The variation of C_{44} near the melting point, from ref. [10].

[11], and was found that C_{44} decreases to 55% of its value at zero-temperature. The discrepancy between the Born theory and experimental results was partly explained

by enhanced thermal generation of defects in the crystal bulk as the melting temperature is approached [12,13]. Recently new calculations [14,15] have shown that Born did not take the contribution of the external stress, P, into account. The new generalized stability criteria are:

$$C_{11} + 2C_{12} - P > 0 \tag{1.7}$$

$$C_{11} - C_{12} - P > 0 \tag{1.8}$$

$$C_{44} - P > 0 \tag{1.9}$$

In addition, on the basis of numerous experiments it has been concluded that the shear modulus C' can become unstable first, $C'(T_m) = 0$ at the melting point in some cases.

The applicability of the Born criterion at zero and non-zero external stress could be directly tested in computer simulations. Bulk melting (the surface of the solid is eliminated by means of periodic boundary conditions) of copper under condition of zero external stress was studied in molecular dynamics (MD) simulations [15]. It was found that the shear modulus C' vanishes at some temperature T_f, but this temperature is large than thermodynamic melting temperature T_m, i.e $T_f > T_m$, measured experimentally. Thus the elastic coefficients remain finite at T_m. The conclusion was drawn that mechanical bulk melting, proposed by Born, is valid only for a perfect infinite crystal, but a real crystal with boundaries and defects undergoes thermodynamic melting at T_m, before it reaches the mechanical melting point.

The Born criterion was modified by Tallon to reach better agreement with experiment [16]. Tallon measured the shear moduli of various substances (metallic, organic, molecular and ionic crystals) as a function of molar volume. He confirmed that the shear moduli do not vanish at the melting point, though the volume of the melt at the melting point can be predicted by a continuous extrapolation one of shear modulus to the zero value at zero external stress. (See Fig. 1.5) Tallon suggested that the shear

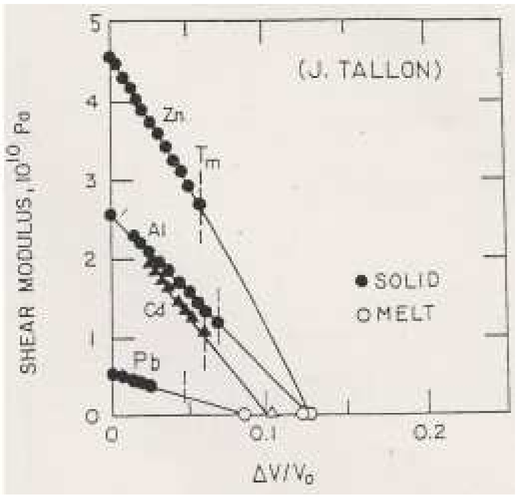


Figure 1.5: The variation of C' with molar volume for various metals [16].

modulus decreases with an increase of the volume of the solid and one of the shear moduli vanishes as a critical volume is reached. At that point each atom can access the entire substance volume due to the enhanced diffusion. The entropy of the system increases and the Gibbs free energy is lowered. At $T = T_m$ the system transforms discontinuously from the solid to the liquid state. In another words, the Born scenario would work if the crystal could be superheated until it will reach the molar volume of the liquid! Unfortunately another mechanism of melting is triggered at the surface of the real solid at temperature which lower than the melting temperature predicted by the Born model, preventing from us to observe mechanical collapse of the crystal lattice.

1.4 Theory of positional disordering

Numerous X-ray studies have made evident that regular repetition of lattice positions of atoms over long sequences in 3D disappears on changing to a melt. The first microscopic theory of melting based on positional disordering was proposed by Lennard-Jones and Devonshire [19]. A simple model of a rare-gas crystal was considered. In order to describe order-disorder transition the system was modeled to be consisted of two inter-penetrating cubic lattices with fcc structure(namely A and B lattices. If all atoms of the solid are located at the sites of the first A-lattice, than the system is perfectly ordered. However, if an atom can jump from the first lattice to the second, then a vacancy-interstitial pair is formed, i.e. atoms belonging to the B-lattice considered as self-interstitials. To place a single atom from the A-lattice on a site of the B-lattice when all the rest atoms on the A-lattice sites requires a considerable increase of energy, owing to the repulsive field of the neighboring atoms. But this energy does decrease if several atoms of the A-lattice jump to the B-lattice or some sites of the B-lattice are already occupied. This energy would decrease to zero in a state of complete disorder. This is an example of cooperative effect, which decreases considerably the defect formation energy.

The partition function for an assembly of N atoms in a state of perfect order is calculated, than it is modified to account for positional disordering. The partition function for a perfect crystal F is given by $F = f^N$, where f is contribution of each atom:

$$f = \left(\frac{2mk_BT}{h^2}\right)^{3/2} v_T \times exp\left(-\frac{\Phi_0}{Nk_BT}\right) \tag{1.10}$$

here v_T is the volume per atom, Φ_0 the potential energy of the system with all the atoms in their position of equilibrium. At higher temperatures positional disorder sets in, defects are formed, and all atoms distributed randomly between the two lattices. The order parameter Q is introduced as $Q = N_A/N$, where N_A is number of sites of the A-lattice occupied by the atoms, and $N = N_A + N_B$ is the total number of the atoms. The $Q = 1$ corresponds to an ordered state, while $Q = 1/2$ to a fully disordered state. Knowledge of the fraction of atoms in each lattice $N_A/N = Q$ and $N_B/N = 1 - Q$ allows us to estimate the total energy of interactions between the defects. Thus in order to modify the partition function of the perfect crystal to account for the positional disordering, one has to multiply the total partition function by the disorder factor:

$$F = f^N \to F = f^N Y(Q) exp\left(\frac{zWNQ(1-Q)}{k_B T}\right) \tag{1.11}$$

where W is defect interaction energy, $Y(Q)$ is a combinatorial factor. The partition function has a maximum at some values of Q, which are the roots of the equation:

$$\frac{zW(2Q-1)}{2k_B T} = ln(Q) - ln(1-Q) \tag{1.12}$$

This equation has a solution with $Q = 1/2$ (a disordered state) which is always satisfied, yet when $T \leq T_c = zW/4k_B$ there is another root with $Q > 1/2$ (an ordered state), at which the free energy has a deeper minimum. The critical point T_c is interpreted as the melting point $T_m = T_c$. The phase transition is found to be first-order. This simple theory of “order-disorder” transition was later generalized to the case of more realistic interaction potential between atoms, and numerical methods were applied to investigate the more complicated models [19].

The Lennard-Jones and Devonshire theory based on positional disordering is capable of predicting some properties of the solid and its melt at the melting transition in rather good agreement with experiment. Nevertheless, there are some flaws in this theory, for example, the theory predicts a continuous transition between the solid phase and the liquid phase at very high pressure, yet experimentally no critical melting point was found even at a very high pressure.

1.5 Melting theories based on point defects

Point defects may play a substational rôle in bulk melting transition. The theory proposed by Weber and Stillinger [20] is an example of a statistical model of bulk melting transition of crystals in which the influence of point defects on the shear moduli is taken into account. According to the model, at temperatures close to the melting point a cooperative formation of point defects takes place. The first excitation in a crystal is a displacement of an atom from its lattice site and hence a vacancy-interstitial pair is formed. The presence of the defect softens the solid, and this effect is amplified while the concentration of defects increases, thereby easing the insertion of more defects and introducing new local modes of vibration. At some critical concentration of defects the lattice of the solid becomes unstable and collapses, i.e. a mechanical melting transition occurs. Minimizing the free energy F with respect to the concentration of point defects, Weber and Stillinger obtained the temperature dependence of the defect concentration $c_{def} = l/N$ (See Fig.1.6). As is seen from the Figure 1.6 at some temperature the defect concentration jumps from $c_{def} = 0.000117$ (as $T \to T_m - 0$) to $c_{def} = 0.99882$ (as $T \to T_m + 0$). This temperature is identified as the melting temperature.

As a further development of the theory of bulk melting based on point defects, Granato [21] derived an interstitial-concentration-dependent Gibbs free energy, appropriate for calculation of all the thermodynamic properties of crystalline, liquid (melt) and amorphous (or glassy) states of metals. The Helmholtz free energy of the

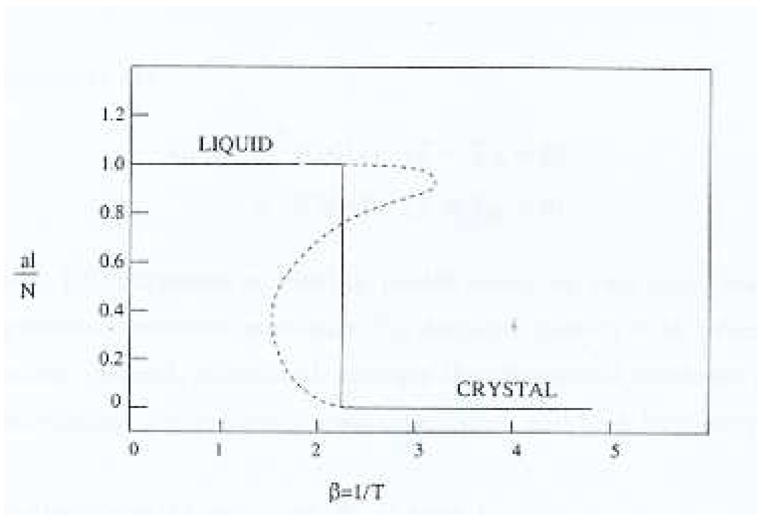


Figure 1.6: Defect concentration, solid line corresponds to stable equilibrium values, dashed line to meta-stable and unstable branches. From ref. [20].

perfect crystal F_p is given by:

$$F_p = F_0^p(V) + F_{vib}^p = F_0(V) + 3Nk_BTln\left(\frac{\hbar\omega_E}{k_BT}\right) \tag{1.13}$$

where $F_0(V)$ is the free energy of the static lattice, i.e. without thermal or zero-point vibrations, and the vibrational free energy F_{vib}, which at high-temperatures is calculated in the framework of a single frequency Einstein approximation. According to Granato, in order to take interstitials into account one has to add to the free energy the following terms:

$$F_{def} = F_w + F_{vib} + F_{conf} \tag{1.14}$$

where F_w is an energy necessary to create a concentration c of interstitials, where $c = n/N$ (n is the number of interstitials and N is the number of atoms). F_{vib} is the change of the vibrational free energy resulting from the change in frequency spectrum, and F_{conf} is the configurational free energy. A salient feature of the model is that the interstitialcy configuration is extended, strongly coupled to the shear stress, with low-frequency resonance modes providing an unusually large entropy per defect. The shear modulus G carries the burden of providing the volume, shear strain, and concentration dependence needed for thermodynamic treatment. The theory predicts that the shear elastic modulus decreases when the concentration of the self-interstitials increases (for example as a result of increase in temperature), which leads to instability of the solid and eventually to a melting transition at T_m.

When the Gibbs free energy is obtained from the Helmholtz free energy we add a term pV to F. Figure 1.7 shows variation of the Gibbs free energy with interstitial concentration. Three different regimes are found by minimizing the Gibbs free

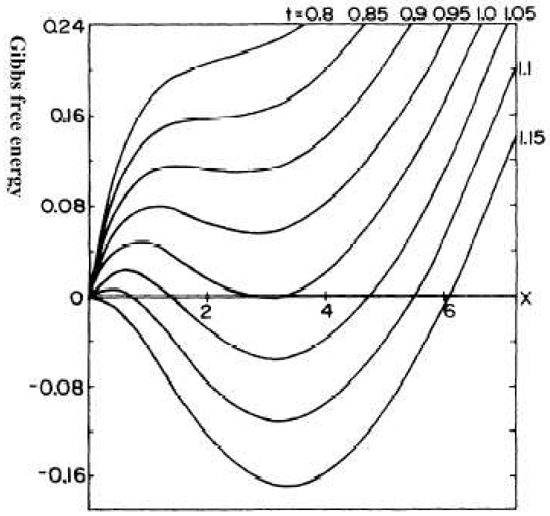


Figure 1.7: The Gibbs free energy of copper as a function of the interstitial concentration for different temperatures $t = T/T_m$, from ref. [21]

energy. For low temperatures (the first regime), when $t = T/T_m < 0.85$ the only stable configuration is the solid with the equilibrium concentration of interstitials about $c \sim 10^{-5}$ defects/atoms. In the second regime, when $0.8 < t < 1.15$, the two minima coexist, the first one corresponds to the solid state with a very low defect concentration, and the second one to the liquid state (regarded as crystals containing a few percent of interstitials, for example for copper it is about 9% at T_m). In the third regime $t > 1.15$ the only stable state is the liquid state. The model predicts the possibility of supercooling of a liquid which is possible experimentally, but also predicts the possibility of superheating of a solid which has never been observed. However, this asymmetric nature of melting is to be understood from the fact that interstitials are produced thermally at surfaces, dislocations, or other shear strain centers.

1.6 Cooperative positional disordering leading to melting

Contemporary models of bulk melting consider positional disordering as a cooperative defect, involving a number of atoms. A characteristic feature of cooperative defects is that the energy of their formation can be greatly reduced relative to isolated point defects. However, this kind of crystal disorder is subject to the condition that the appropriate sequence of neighboring sites must be displaced cooperatively from the ideal lattice positions. Within recent years much work as been done on a particular kind of cooperative defects known as lattice dislocations. As is well-known, a point defect consists either of a missing atom in the lattice or an extra atom between two normal lattice points. If more than one adjacent point defect occurs in a crystal here may be a slip along a surface causing a line defect which is called a dislocation (See Fig. 1.8). Dislocations also contribute to melting of metals. Theories of dislocation-mediated melting have several appealing features; the free energy of a crystal containing a dense array of dislocations is comparable to the free energy of its melt. The fluidity of the

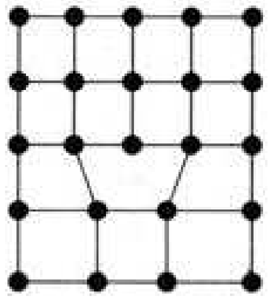


Figure 1.8: Dislocation is a line defect.

melt can be attributed to the mobility of dense array of dislocations. The part of the total energy of a solid, saturated with dislocations, attributed to these defects is comparable to the latent heat of fusion. The presence of dislocations lowers the energy of creation of additional dislocations; if this reduction is sufficiently strong it can lead to an avalanche of dislocations in a first-order transition. A number of experiments and computer simulations have given some support for these theories of dislocation induced melting, but the evidence is not clear enough to conclude that dislocation-mediated melting has been demonstrated.

Chapter 2

Surface melting

2.1 Preface

All the models considered at the previous chapter explain the mechanism of melting within the bulk material. These theories consider melting process as a breakdown of the crystal lattice which occurs uniformly throughout the solid at the melting point, i.e. melting is required to be homogeneous. However, there is ample evidences that melting is actually heterogeneous, and that it involves nucleation of the liquid phase at some preferred sites of the solid (the free surface, grain boundaries, large dislocations and disclinations, etc), followed by subsequent growth of the liquid phase. The most likely site from where the solids start to melt is the free surface of the solid. A premelting of the surface offers an elegant solution to an intriguing puzzle of melting, namely why a liquid could be cooled below the freezing point ("supercooling"), but a solid can not be heated above the melting point ("superheating"). On the basis of classical nucleation theory one expects, upon melting and freezing, that these hysteresis effects should occur. The absence of "superheating" of solids, is indicative of the general absence of an energy barrier for the nucleation of a melt, yet such a barrier does exist for solidification. Apparently the formation of a liquid surface layer at temperatures below the melting temperature eliminates the need for liquid nucleation at melting transition, thus no metastability effects exist.

Tammann [22] was the first to point out that surface may play an important rôle in initiating melting. The concept that melting may start at the surface can be inferred from an empirical criterion formulated by Lindemann [1]. Relying on the criterion one may argue that the outermost atomic layer of the crystal should disorder far below the bulk melting point. The loosely bounded surface atoms have a reduced number of neighbors, and therefore have a higher vibrational amplitude than in the bulk. Consequently, at the surface the Lindemann criterion is satisfied at a lower temperature. Surface melting involves a formation of a thin disordered layer at some relatively high temperature. Surface melting considered here has nothing to with the trivial melting caused by temperature gradient, for example, melting of an ice cube floating in a glass of water. In this case the surface melts simply because the outside is hotter than the inside.

While numerous attempts to detect surface-initiated melting phenomena have been made, it is only recently that surface melting has been observed directly on a microscopic level by employing atomically clean, well characterized surfaces. The first direct observations were made using Rutherford backscattering [23], in conjunction with shadowing and blocking. Since then, other techniques have been employed such as calorimetry, electron, neutron and X-ray diffraction, microscopy, ellipsometry, and helium scattering, and even visual inspection with the naked eye. Most experiments have been carried out near equilibrium. But even when a solid is heated suddenly, melting still tends to be initiated at the surface [24]. Under some conditions it is possible to force the solid to break down internally, but then the melting will tend to begin at internal interfaces such as grain boundaries.

Theoretical aspects of surface melting have been studied by phenomenological Landau theories [27-30], lattice models [36], and density functional theory [40]. A more detailed review of those methods is given below. Computer simulations play a leading part in the studying of the phenomena in a microscopic level. The story of surface melting is not yet complete. Still to be explored is the evolution of layer structure at the surface region in detail.

2.2 Phenomenological thermodynamics model

Surface melting can be regarded as a case of wetting [27], namely a wetting of the solid by its own melt. As in the case of adsorption of a gas onto a hard wall one may observe complete or incomplete wetting, depending on whether the quasiliquid thickness diverges or remains finite as $T \to T_m$. By the same analogy, the case of non-wetting corresponds to an absence of surface melting, e.g. the surface remains dry up to T_m. What makes the surface premelt or remain dry? If a single crystal is cleaved along it $\{hkp\}$ plane, where $h, k, p \in Z$ are the crystallographic indices, then the surface free energy per unit of area $\gamma_{sv}^{\{hkp\}}$ is defined as the work needed to create a unit area of dry surface (the subscript 'sv' refers to the solid-vapor interface). On the other hand, the free energy of a surface that at T_m is covered with a thick melt layer, is given by $\gamma_{sl}^{\{hkp\}} + \gamma_{lv}$, where the indices 'sl' and 'sv' refer to the solid-liquid and liquid-vapor interfaces, respectively (See Fig. 2.1). Surface melting will only occur if there is a gain in the free energy, that is, if

$$\Delta\gamma^{\{hkl\}} \equiv \gamma_{sv}^{\{hkl\}} - \left(\gamma_{sl}^{\{hkl\}} + \gamma_{lv}\right) > 0 \tag{2.1}$$

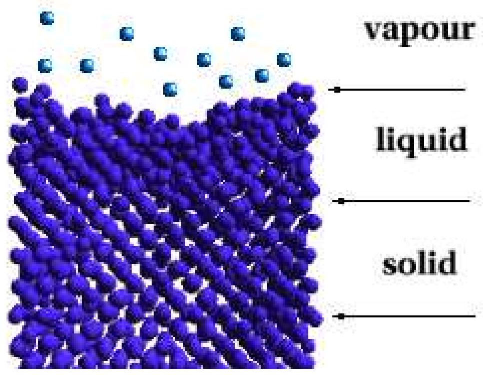


Figure 2.1: The solid-liquid and liquid-vapor interfaces

On the other hand, if $\Delta\gamma^{\{hkp\}} < 0$, then the surface will remain dry up to T_m. The sign and magnitude of $\Delta\gamma^{\{hkp\}}$ depend not only on the material, but also on the surface orientation. In general the most open crystal faces, e.g. the (011) face for fcc crystals and (111) for bcc crystals, are most likely to exhibit surface melting. For a system exhibiting complete wetting there is a unique relation between the temperature T and the equilibrium thickness of the quasiliquid layer $\bar{l}(T)$. The fact that there is a finite equilibrium thickness of the film at a given temperature T is a result of balance between two opposite thermodynamic forces. On one hand, the quasiliquid becomes more liquid-like for increasing layer thickness, which results in some gain in the free energy. This corresponds to an effective repulsive force between the solid liquid and liquid-vapor interface. The effective interaction energy between the interfaces at either side of the quasiliquid layer is given by $\Delta\gamma^{\{hkp\}}exp\{-\frac{2l}{\zeta_b}\} + Wl^{-2}$, where l is the thickness of the film, ζ_b is a characteristic length scale over which the crystalline order decays [23,26] as it is measured from the crystal-quasiliquid interface, and W is a positive constant which is called the Hadamar constant. On the other hand there is the free energy cost associated with supercooling of the quasiliquid layer. For a layer of thickness l the energy cost per unit of area is $L_m l(1 - T/T_m)$, where L_m is the latent heat of melting per unit volume. This yields an attractive force between the two interfaces.

The total free energy $F(l)$ of the surface covered with a melt layer of thickness l is:

$$F(l) = \gamma_{sv}^{\{hkp\}} + \gamma_{lv} + \Delta\gamma^{\{hkp\}} exp\{-\frac{2l}{\zeta_b}\} + \frac{W}{l^2} + L_m l(1 - \frac{T}{T_m}) \tag{2.2}$$

The equilibrium thickness $\bar{l}(T)$ is the value of l for which $F(l)$ is minimal, i.e. $dF(l)/dl = 0$. Let us define define a crossover thickness l_c as a thickness for which the long-range contribution Wl^{-2} to $F(l)$ is equal to the short-contribution $\Delta\gamma^{\{hkp\}} exp\{-\frac{2l}{\zeta_b}\}$. Consequently two different regime are considered. The first regime is for $\bar{l}(T) << l_c$, where the system is governed by the short-range exponentially decaying interactions and the equilibrium thickness is given by:

$$\bar{l}(T) = \frac{\zeta_b}{2} ln \left[\frac{2\Delta\gamma^{\{hkp\}} T_m}{L_m(T - T_m)\zeta_b} \right] \simeq -ln(t) \tag{2.3}$$

where the reduced temperature is given by $t = (1 - T/T_m)$. This kind of logarithmic divergence of $\bar{l}(T)$ is characteristic for metals and semiconductors. But for rare-gas crystals (or if T_m is approached very-very closely [23]) the long-range force must eventually dominate the melting behavior in the second regime $\bar{l}(T) >> l_c$. The short-ranged force will dampen out and one is left with van der Waals type dispersion forces. Thus minimizing $F(l)$ with respect to l yields:

$$\bar{l}(T) = \left[\frac{(T - T_m)}{2T_m W} \right]^{-1/3} \tag{2.4}$$

Here it is implied that $W > 0$ and the liquid is less dense than the solid. This power law was tested in numerous experiments carried out with rare-gas crystals. The agreement between the theoretical prediction and the experimental results is very good.

2.3 Landau model of surface premelting

Historically, the first model of surface melting in the framework of Landau-Ginsburg theory was proposed by Lipowsky [28,29,30]. The phenomenological model considered above, is a particular case of this type of models, which are more abstract and general. As it is well-known Landau theory is based on a power series expansion in the order parameter for the phase transition of interest. In the framework of the theory it is assumed that the order parameter is "small", so that only the lowest order terms required by symmetry are kept. It is most useful in the vicinity of second-order phase transitions, where the order parameter is guaranteed to be small. It, however, can be used with care to first - order transitions. Lipowsky considered a semi-infinite system which undergo a first-order phase transition at $T = T^*$ in the bulk, e.g. the bulk order parameter M_b jumps to zero at the melting point. However, the surface order parameter may nevertheless behaves continuously like $M_s \simeq |T - T^*|^{\beta_1}$.

A d-dimensional semi-infinite system with a $(d-1)$ dimensional surface is considered. The coordinate perpendicular to the surface is denoted by z. As a result of the broken translation symmetry (translation invariance) at the surface, the order parameter M depends on z: $M = M(z)$. The Landau expansion for the free energy (per unit area) has the generic form:

$$f\{M\} = \int_0^\infty dz \left[\frac{1}{2}\left(\frac{dM}{dz}\right)^2 + f(M) + \delta(z) f_1(M) \right] \tag{2.5}$$

where $(\frac{dM}{dz})^2$ corresponds to an increase of the free energy due to inhomogeneity of the order parameter, and the bulk term $f(M)$ is given by the well-known expression for a system with a bulk tricritical point:

$$f(M) = \frac{1}{2}a(T)M^2 + \frac{1}{4}uM^4 + \frac{1}{6}vM^6 \tag{2.6}$$

with $v > 0$ and $u < 0$, which leads to first-order bulk transition at $a(T^*) = a^* = 3u^2/16v$, and to a jump of the order parameter from $M_B = (3|u|/4v)^{1/2}$ to zero. The additional term $\delta(z)f_1(M)$ mimics the microscopic changes of the interaction parameters near the free surface. If $f_1(M)$ is expanded in powers of of M (up to second order), one obtains:

$$f_1(M) = \frac{1}{2}a_1 M^2 \tag{2.7}$$

where $a_1 > 0$ is a constant called *extrapolation length*, which is independent of temperature. Inclusion of the higher-order terms in this expression does not change significantly the results deduced from the theory.

By minimizing the free energy $\delta F/\delta M = 0$ one obtains a differential equation for the order parameter:

$$\frac{dM}{dz} = [2f(M) - 2f(M_B)]^{1/2} \tag{2.8}$$

together with the implicit equation for the surface layer:

$$\frac{\partial f_1(M_1)}{\partial M_1} = [2f(M_1) - 2f(M_B)]^{1/2} \tag{2.9}$$

where $M_1 \equiv M(z = 0)$. From the last equation one finds the temperature dependence of order parameter M_1 as $T \to T^*$ from below:

$$M_1 \simeq \begin{cases} const, & a_1 < \sqrt{a^*} \\ |T - T^*|^{1/4} & a_1 = \sqrt{a^*} \\ |T - T^*|^{1/2} & a_1 > \sqrt{a^*} \end{cases} \tag{2.10}$$

where a_1 is inverse extrapolation length, and a^* is a Landau coefficient at $T = T^*$. Thus, two different types of phase transition are obtained, referred as O_1 and O_2. (See Fig. 2.2) At the transition O_1 (when the inverse extrapolation length is small $a_1 < \sqrt{a^*}$), the surface order parameter M_1 is discontinuous like the bulk order

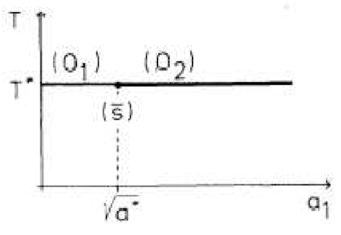


Figure 2.2: Phase diagram, from ref. [28].

parameter M_B. However, at the transition O_2 (when the inverse extrapolation length is large $a_1 > \sqrt{a^*}$) and at the the tricritical point $\bar{s}$ ($a_1 = \sqrt{a^*}$), the surface order parameter goes continuously to zero with the surface exponents $\beta_1 = 1/4$ (for O_2) and $\beta_2 = 1/2$ (for $\bar{s}$). This is rather surprising since there are no corresponding bulk exponents. An additional unexpected feature, discovered by Lipowsky, is that a layer of disordered phase intervenes between the free surface and the ordered bulk as the critical point T^* is approached from below (See Fig. 2.3). Hence, an interface

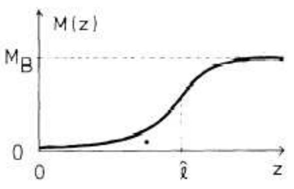


Figure 2.3: Order-parameter profile $M(z)$ as O_2 and $\bar{s}$ are approached from $T < T^*$, from ref. [28].

appears at $z = \bar{l}$, which separates the disordered surface layer from the ordered phase in the bulk. As T^* is approached, this interference becomes delocalized since $\bar{l} \sim |ln(T^* - T)|$. Within Landau theory, such a logarithmic divergence has also been found in wetting transition.

In order to decide whether the new type of continuous surface melting, i.e. the

O_2 transition may take place one has to estimate the magnitude of the inverse extrapolation length a_1. The parameter a_1 should be calculated relying on microscopic model. As a first step toward this goal, the semi-infinite g state Potts model on a lattice has been investigated by means of mean-field theory. The Hamiltonian of the g state Potts model is given by:

$$H = \sum_{<ij>}^{g} J\delta_{s_i,s_j} + J_1\delta_{s_i,s_j} \tag{2.11}$$

here J_1 is the coupling constant for a pair of spins at the surface, and J is the coupling constant for a pair of spins in the bulk, and g is the number of possible spin orientations, and the sum includes the nearest neighbors. It was found that in a three dimensional Potts model with $g = 3$ the new continuous transition occurs, provided $J_1 \leq 1.1J$. It seems very likely that the interaction parameters of real systems fulfill this inequality.

2.4 Layering effect

The surface premelting of solids is explained in terms of a repulsive and attractive interactions between the solid-liquid and the liquid-vapor interfaces. Ercolessi et al. [31] showed how a crucial part of this interaction originates from the layering effects near the liquid metal surface. The layering effect [32,98,99] is density oscillations which are observable at the liquid surface of metals and semiconductors (See Fig. 2.4). The layering effect was first noted in molecular dynamics simulations [33], and recently have been observed experimentally for Hg [34]. The layering effect in metals is very similar to a layering of a fluid near a wall. It is presumed that the liquid-vapor interface acts as a sort of a rigid wall for the liquid metal. Different models has been proposed to explain the layering effect in metals, for instance, Rice et al. [35]

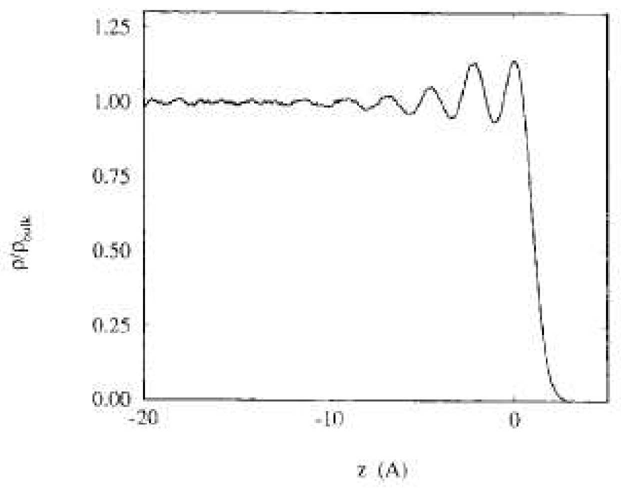


Figure 2.4: Density profile of the liquid surface of aluminum exhibits pronounced oscillations at the liquid-vapor interface. Figure from ref. [32].

claimed that the density oscillations are due to the coupling between the electronic and ionic profiles. The electron density profile decays very abruptly and generates an effective wall potential against which the ions need to rearrange themselves to reduce the energy cost and lay orderly.

The oscillation is characterized by a typical liquid periodicity, $2\pi/Q_0$, where Q_0 is the absolute value of the wave vector, at which the liquid radial distribution function has its maximum. Besides that periodicity, $2\pi/Q_0$, there is another one, a second oscillation at the solid-liquid interface. This is due to the crystal planes which induce density fluctuation in the liquid layer. The periodicity of this oscillation is given by the distance between the crystal planes, a, which in turn depends on the orientation $\{hkp\}$ of the planes of the underlying crystal.

These two distinct layering oscillations - one with the periodicity $2\pi/Q_0$ (tied to the liquid - vapor interface) and another with the periodicity of the interlayer spacing a (tied to the solid - liquid interface) overlap and interfere inside the liquid film, provided that the separation between the solid-liquid and the liquid-vapor interfaces is small. The kind of interference depends strongly on the orientation of the underlying crystal. For the close packed face, for example (111) face of fcc crystals, the periodicities are match, e.g. $2\pi/Q_0 = a$, and one have constructive interference (See Fig. 2.5). The

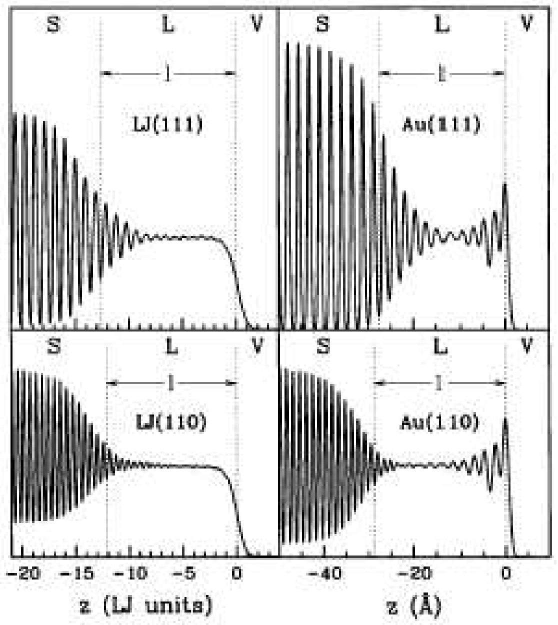


Figure 2.5: Comparison between the density profiles of the (111) and (110) surfaces of a Lennard-Jones crystal and gold obtained by MD simulations, from ref. [32].

interference induces oscillation of the free energy, and the deepest minimum of the free energy of the close packed face is at zero thickness of the quasiliquid film as it is shown in Figure 2.6. The interfaces strongly attract each other and this attraction leads to the absence of surface premelting. In the opposite case, when $2\pi/Q_0 \neq a$ the interference is destructive, and therefore there is a repulsion between the interfaces and surface premelting is observed.

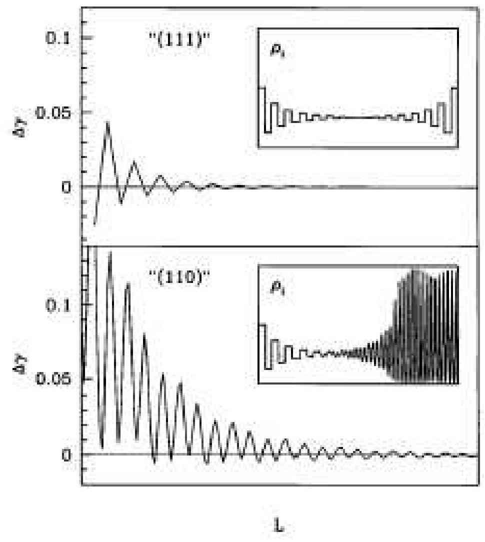


Figure 2.6: Variation of the surface free energies as a function of the interface separation l. Upper panel: the close packed (111) face of a fcc crystal, constructive interference. Lower panel: the least packed (011) face of a fcc crystal, deconstructive interference, from ref. [32].

2.5 Lattice theory

The first microscopic theory of surface melting have been developed by A. Troyanov and E. Tossati [36]. For a microscopic theory, one would need for a start a simple and accurate model, capable of accounting for the bulk phase diagram including solid, liquid and vapor phases, the triple point and the critical point of any substance. Such a complete theory has not yet been developed and the very basic building block for a microscopic surface melting theory is missing.

Nevertheless A. Troyanov and E. Tossati [36] developed a theory of surface melting, based on the fact that at the price of introducing a discrete reference lattice and using a drastic simplification like the mean-field approach, the partition function, Z_N, of a system of particles interacting via a pairwise potential can be calculated. In order to calculate the free energy F of an assembly of N atoms they introduced a reference lattice. The volume v_0 of each cell is chosen to be so small, that possibility of multiple occupancy of a lattice cell can be neglected.

The partition function Z_N is:

$$Z_N = v_0^N \sum_{\{p_i\}} \int_{v_0} .... \int_{v_0} exp\left[-\frac{\beta}{2}\sum p_i p_j U(r_i, r_j)\right] dr_1 dr_2 ... dr_N = e^{-\beta F} \tag{2.12}$$

where $U(r_i, r_j)$ is a pair-wise interaction potential of a Lennard-Jones type, $\beta = (kT)^{-1}$ is the inverse temperature, and $p_i = 0, 1$, i.e. it is zero if a considered lattice site is empty and unity otherwise. The summation $\sum_{\{p_i\}}$ is taken over all possible configurations of N atoms on the lattice sites. The main approximation of the model is splitting of Z_N onto a separate lattice sum, Q, and a free volume term Ω, e.g. $Z_N \approx Q\Omega$. The lattice term is an Ising-like spin lattice model state sum which is evaluated using the saddle-point approximation. The second term Ω depends on cooperative motion of the atoms and it is a complicated function of their positions, which calculated using the special effective volume method [36].

In order to describe order-disorder transition two order parameters - "crystallinity", c_l, and average density, ρ_0, are defined and calculated for each layer of the reference lattice separately. The "crystallinity" in each layer l is given by:

$$c_l = \frac{< p_{0,l} > - < p_{1,l} >}{< p_{0,l} > + < p_{1,l} >} \tag{2.13}$$

where l is the layer number, $< p_{0,l} >$ is the average occupation of the first reference lattice, and $< p_{1,l} >$ is the average occupation of the second one. The free energy of the system is expressed in term of density ρ_l and crystallinity c_l. The minimization of the free energy with respect to these variables leads to a set of nonlinear algebraic equations which is supplemented by the boundary conditions for the solid bulk. This system of a large number of coupled layer-by-layer equations ($M \approx 400$) is solved iteratively, with an initial guess for the set of $\{c_l^0, \rho_l^0\}$. It was found that the solution of the set of those equations is unique, i.e. it does not depend on the initial guess of $\{c_l^0, \rho_l^0\}$.

According to the calculation, surface melting takes place both on the (100) and (110) surfaces. A thin liquid-like film gradually appears at the surface region. The crystallinity inside the quasiliquid film drops rapidly to zero, and also density is jumping rather abruptly from liquid-like to solid-like (See Fig.2.7), but this drastic change of the density is seem to be due to the mean-field approximation, e.g. due to ignoring of fluctuations. Molecular dynamics studies have shown substantially broader transition from a solid-like to liquid-like region in surface premelting. The

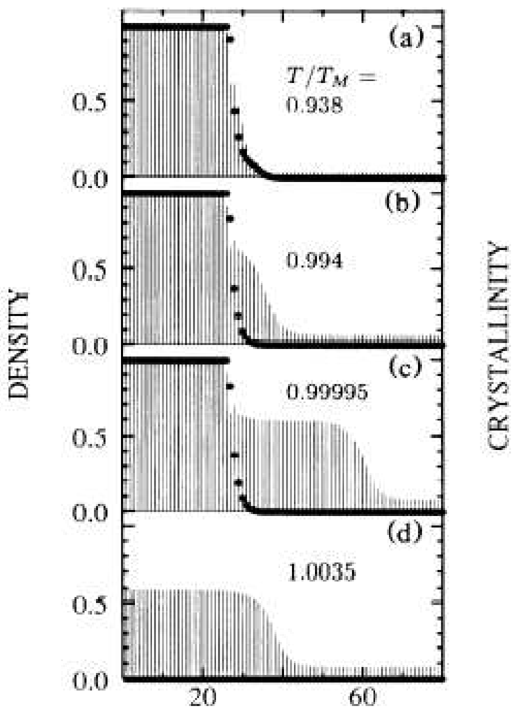


Figure 2.7: Density profiles at the solid-to-liquid and liquid-to-gas interfaces at various temperatures T/T_m, from ref. [36].

thickness of the quasiliquid film increases very fast as T_m is neared. The growth behavior is depended on the range of inter-atomic interactions, and in this case of rare-gas solids the interactions are considered to be long-ranged (van der Waals potential), and therefore the power law of the thickness growth with temperature was expected $\bar{l} \sim t^{-p}$, where $\bar{l}$ is the thickness of the quasiliquid film, $t = 1 - T/T_m$ is reduced temperature and the exponent $p = 1/3$. In the frame of this model, $\bar{l}$ is defined to be equal to the number of layers whose crystallinity is less than 1/2 (the interface with the solid) and the density is no less than 90% of the bulk liquid density (the interface with the vapor). The theory confirmed the power-law dependence of the thickness on reduced temperature (See Fig. 2.8). The surface free energy γ decreases

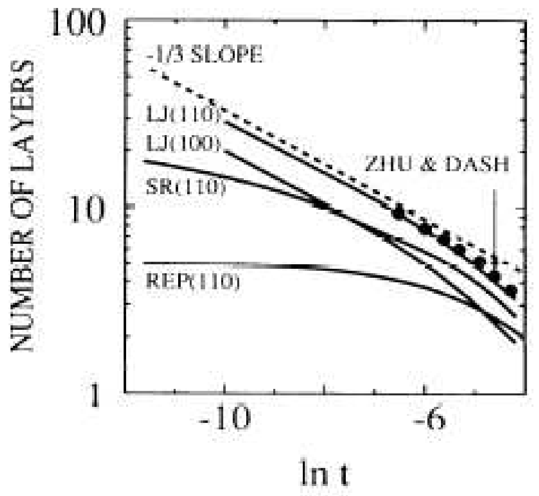


Figure 2.8: Dependence of the quasilquid layer thickness upon the reduced temperature, from ref. [36].

with temperature very fast when T_m is neared (See Fig. 2.9). The anisotropy of γ diminishes with temperature, and both the (100) and (110) surfaces free energies merge at about $0.9T_m$. The overall decrease is remarkably large, about 200% form

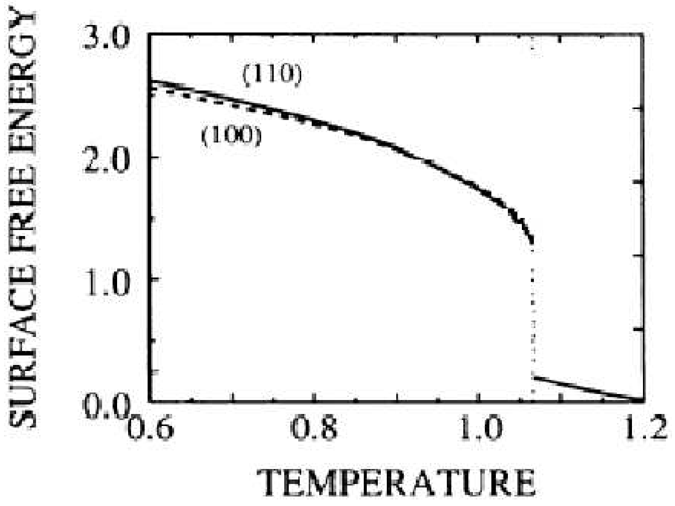


Figure 2.9: Temperature dependence (T/T_m) of the surface free energy γ of the (110) and (100) LJ faces, from ref. [36].

$0.5T_m$ to T_m, in contrast to roughening, where the same decrease of the surface free energy γ is about 1%. In this model the surface transition appears to be continuous. At T_m, γ has a vertical tangent and jumps to the value of the surface free energy of the liquid-vapor interface.

In order to understand the influence of the form of the inter-atomic potential on surface melting A. Troyanov and E. Tossati [36] investigated the phenomenon when the sign of the tail of the **LJ** potential was reversed. The physical origin of such a weak repulsive at long-range distance was not proposed and the reversion was merely considered as a tool for better understanding of the phenomenon. But it seems plausible, that many-body interactions might effectively lead to a long-range repulsion.

It turns out that a small change of the interatomic potential has a dramatic effect on surface melting. The surface starts to melt, i.e. few layers become disordered with the increase of temperature, but further increase of temperature does not increase the number of molten layers. This phenomenon is called “blocked surface melting” (an analog of incomplete wetting). This “blocked surface melting” was predicted by phenomenological theories of surface melting [37] and also was observed experimentally at the Ge(111) surface [38]. It was concluded that the mode of growth of the quasiliquid layer is extremely fragile and sensitive to the range and sign of the interatomic potential.

This theory excludes fluctuations, which may play an important rôle in surface melting. For example, both the solid-liquid and the liquid-vapor interfaces are expected to execute very close to T_m a joint “meandering”, i.e. out-of-plane fluctuations, which is typical for all surfaces about their roughening temperature. It is not clear that effect could roughening have on the results predicted by the model. Surface melting and roughening are, respectively, short-range and long-range phenomena, and need not necessarily interfere one with another. It may happen that locally, on a short-range scale, the physics is described by surface melting, while globally only roughening will matter, irrespective of whether surface melting taking place or not. If in-plane fluctuations are considered, then there is a possibility of a “first layer melting” [39] at temperatures of order $0.7T_m$. This transition is acting as a kind of “gateway” to the subsequent development of a quasiliquid layer. The surface layers become shear unstable due to anharmonic effects and diffusion of the surface atoms sets in.

2.6 Density functional theory of surface melting

The first density functional theory of surface melting was proposed by R. Ohnegson et al. [40]. In the density functional approach the central quantity is the grand canonical free energy functional $\Omega[\rho]$ of an inhomogeneous system with local density $\rho(\vec{r})$, temperature T, and chemical potential μ. The grand canonical free energy functional is given by:

$$\Omega[\rho] = F_{exc}[\rho] + \int d^3r\rho(r)\left\{V_{ext}(r) - \mu + k_BT\left(ln(\Lambda^3\rho(r)) - 1\right)\right\} \qquad (2.14)$$

where Λ denotes the thermal wavelength $\Lambda \sim \left(\frac{\hbar^2}{mk_BT}\right)^2$, $V_{ext}(r)$ is an external potential, and $F_{exc}[\rho] = F[\rho] - F_{id}[\rho]$ is an excess free energy functional. In general, the explicit form of $F_{exc}[\rho]$ is not known and one has to rely on approximations. For example, R. Ohnegson et al. [40] used the analytical Percus-Yevick expressions [41] for the excess free energy functional of particles interacting via Lennard-Jones potential. The main problem is to find the equilibrium density $\rho_{eq}(r)$ which minimizes the grand canonical free energy functional $\delta\Omega[\rho_{eq}(r)]/\delta\rho = 0$. The minimization of the free energy functional was done numerically, using the simulated quenching [40], which is similar to conjugated gradient method, but considered to be more efficient than the last one.

Surface melting and especially the onset of *anisotropic surface disordering* was also investigated theoretically for the first time (in the framework of density functional theory). This issue can not be addressed with a phenomenological approach where the existence of a wetting film viewed as an undercooled liquid is taken for granted, but requires a fully microscopic theory. Surface melting is visible for each orientation, with a clear anisotropy in the structure of interface (See Fig. 2.10). The more loosely packed (110) and (100) planes of a fcc Lennard-Jones crystal are more disordered than the dense (111) plane. The theory predicts the logarithmic growth law for

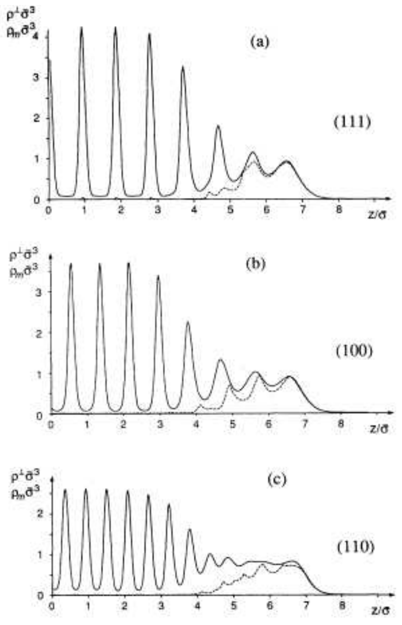


Figure 2.10: Density profile $\rho(z)$ vs z for a LJ system obtained from hard-sphere perturbation theory at the reduced temperature $t = 1 - T/T_m$, from ref. [40].

the quasiliquid film $\bar{l} \sim -ln(t)$, which is determined by a short-ranged interaction between the particles. Besides this, for the (100) surface orientation the hysteresis effect in minimizing the density functional was observed for increasing and decreasing temperatures which hints layer-by-layer growth of the quasiliquid layer via a first-order surface phase transition.

Chapter 3

Numerical Methods

3.1 Molecular dynamics

Molecular Dynamics method (**MD**) [42,54,55] has become a very powerful tool to attack many-body problems in statistical physics. This method allows studying specific aspects of complex systems in great detail via computer simulations. Simulations need specific input parameters that characterize the system in question, and which come either from theoretical models or from experimental data. These data help to fix the parameters of the model, the main part of which is interactions between atoms represented by an interatomic potential.

Once the potential is specified and the initial conditions, i.e. initial coordinates and velocities, are chosen the MD method can be applied. Molecular Dynamics are governed by the system's Hamiltonian H and consequently by Hamiltonian's equations of motion:

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \tag{3.1}$$

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \tag{3.2}$$

We integrate these equations to move particles to new positions $\{q_i\}$ and calculate new momenta $\{p_i\}$. The Hamiltonian of a system in classical MD simulations is usually written as a sum of the potential energy $V(\{q_i\})$, which depends on coordinates only, and the kinetic energy $K(\{p_i\})$, which is a function of momenta:

$$H = K(\{p_i\}) + V(\{q_i\}) = \sum_i \frac{p_i^2}{2m} + V(q_i) \tag{3.3}$$

The next step is the integration of these equations. For this purpose an integrator is needed, which propagates particle positions and velocities in time. It is a finite difference scheme, which moves particle trajectories discretely in time. The requirements for the integrator are the following: the integrator has to be accurate, in the sense that it approximates the true trajectory well enough, as well as it has to be stable, in the sense that it conserves the total energy and that small perturbations do not lead to instability. Different integrators could be used to solve Newton's equations differing in accuracy, complexity and speed. Among these the predictor-corrector (**PC**) [54-56], method was chosen for our simulations (See Appendix A).

Obviously, a statistical ensemble has to be chosen, where thermodynamic quantities like temperature, pressure and number of particles are controlled. There are algorithms, which fix temperature, volume or pressure to an appropriate value. In our simulations we use the canonical ensemble (**NVT**), with *Nose-Hoover algorithm* [60] which controls the temperature (See Appendix B) and the extended isothermal-isotension ensemble (**NtT**)(See Appendix C) of statistical mechanics [57, 58]. The last one is used to calculate the shape and the volume, as well as the shear elastic moduli(See Appendix D) of vanadium containing point defects.

The above mentioned steps essentially define the **MD** simulations. Having this tool at hand, it is possible in principle to obtain exact results, within numerical precision and round off computer errors. Of course, these results are only valid with respect to the model, which enters into simulations. The results of simulations have to be tested against theoretical predictions and experimental findings.

An important issue of MD simulations is the *accessible time and length scale*, which can be explored in MD simulations. It is clear that the more elaborate and detailed simulation technique is applied, the shorter is the time scale and smaller the length scale are accessible in simulations, due to limitation on the available CPU time and/or accessible computer memory. In classical molecular dynamics, in the case of solid state simulations, the time scale is dominated by the time scale of atomic vibrations (picosecond), and the accessible length scale is about $\simeq\ 100\ \dot{A}$.

With the development of faster parallel architecture computers the accessible time and length scales are gradually increasing. In 1999, as quoted in [59], J. Roth demonstrated on the **Cray T3E-1200** computer in J$\ddot{u}$lich, that it is possible to simulate more than **5,000,000,000** particles corresponding to the length scale of several thousand angstroms. In another demonstration run quoted in [59] Y. Duan and A. Kollman extended the time scale of a MD simulations up to 1 μs.

3.2 The interatomic potential

The heart of the MD simulations is the inter-atomic potential. In classical simulations the atoms are most often represented by point-like centers, which interact through many-body interactions potential. In that way the highly complex description of electron dynamics is abandoned and effective picture is adopted. In this picture the main features like the hard core of particles and internal degrees of freedom are modeled by a set of parameters and analytical functions, which depend on the mutual positions of the atoms in the configuration. These parameters and functions give a complete information about the system energy, as well as about the forces acting on each particle.

The best choice of a potential for simulations of metals is a many-body potential. It is well known fact that pair-wise potentials, like Lenard - Jones (**LJ**) potentials :

$$\Phi_{LJ}(r) = 4\epsilon(\frac{\sigma^{12}}{r^{12}} - \frac{\sigma^{6}}{r^{6}}) \tag{3.4}$$

do not give adequate description of all the properties of metals. For example, the **LJ** potential imposes the Cauchy relation $C_{12} = C_{44}$ ($C_{12},\ C_{44}$ are elastic constants). This relation is proved to be wrong for most of the metals. Pair-wise potentials fail to estimate the structure relaxation and reconstruction around point defects (vacancies and self-interstitials) in metals. The vacancy formation energy obtained by means of pair - wise potentials is overestimated, and is found to be about equal to the bulk cohesive energy.

The solution of the problem is an introduction of a many-body potential, which include a pair-wise interactions only as part of the full potential. This first part of the many-body potential accounts for the core - core interactions (or ion - ion interactions), while the second part incorporates the complex nature of metallic cohesion by an additional term:

$$V = V_1 + V_2 = \frac{1}{2}\sum_{i:\ i\neq j}^{N} \Phi(r_{ij}) + \sum_{i=1}^{N} U(n_i) \tag{3.5}$$

where the $\Phi(r)$ is the two-body part, and the many-body part $U(n_i)$ depends on electronic charge density n_i around the atom i:

$$n_i = \sum_{i\neq j}^{N_g} \rho(r_{ij}) \tag{3.6}$$

N_g is the number of nearest neighbors of the atom i.

Various many-body potentials for metals, include the Finnis-Sinclair potentials (**FS**) [44], the effective medium potentials [45], tight-binding type potentials [47,48], as well as potentials based on the embedded atom methods (**EAM**) [43] and the glue model [46]. Many-body potentials for **fcc** metals are well developed and thoroughly tested in numerous simulations. The situation with the many-body potentials for **bcc** metals is not so good. That can be explained by the more complicated nature of the bcc metals (especially transition bcc metals), where *d-orbitals* have to be taken into account. Different versions of the many-body potentials for bcc metals are often adopted for some specific problems, for example, to describe defect configurations in transition metals at low temperatures. Therefore, it is not clear if these potentials could be applied to simulate bcc metals at high temperatures up to the melting point. One way to find the answer is to perform simulations, using these potentials, and compare the results with available experimental data and theoretical predictions.

In our project, the Finnis - Sinclair (**FS**) potential for a bcc metal vanadium [42] was chosen, for the following reason:
First, the **FS** potential gives good description of the properties of transition bcc metals, and it is widely used in Molecular Dynamics and Monte - Carlo simulations. Second, the **FS** potential is an analytical one, which is more convenient for to use in MD simulations, than the numerical EAM type potentials. It is especially important when we calculate high-order derivatives of the potential, for example, in calculation of the shear elastic moduli.

The **FS** potential includes the pair-wise and many-body parts:

$$U = U_1 + U_2 = \frac{1}{2} \sum_{i,j:\ i \neq j}^{N} V_1(r_{ij}) + \sum_{i=1}^{N} V_2(n_i) \tag{3.7}$$

The pair-wise part $V_1(r_{ij})$ is represented by a quartic form (see below), and the many-body part $V_2(n_i)$ of the potential is given by:

$$V_2(n_i) = f(n_i) = -A\sqrt{n_i} \tag{3.8}$$

where n_i is the local electronic charge density around the atom i:

$$n_i = \sum_{i:\ i\neq j}^{Ng} \Phi(r_{ij}) \tag{3.9}$$

the function $\Phi(r_{ij})$ is given by:

$$\Phi(r_{ij}) = \begin{cases} (r_{ij}-d)^2, & r_{ij} \leq d \\ 0, & r_{ij} > d \end{cases} \tag{3.10}$$

where r_{ji} is distance between the atoms i and j,

and A, d are the fitting parameters.

The function $f(n_i)$ is obviously taken to mimic the results of the *tight-binding* theory. Indeed, n_i can be considered as the local electronic charge density at a site i, which is constructed by a rigid superposition of atomic charge densities $\phi(r)$. This is actually a viewpoint of Daw and Baskin [49]. To a first approximation, the energy of an atom at a site i is assumed to be the same as if the atom is placed into an uniform electron gas of that density. The remaining part of the potential is based on empirical fitting. The pair-wise potential represents repulsive *core - core interactions*. The pair potential can be written in the form:

$$U_1 = \frac{1}{2}\sum_{i:\ i\neq j}^{N} V_1(r_{ij}) \tag{3.11}$$

where the pair potential V_1 is a quartic polynomial:

$$V_1(r_{ij}) = \begin{cases} (r_{ij}-c)^2 \times (c_0 + c_1 r_{ij} + c_2 r_{ij}^2), & r_{ij} \leq c \\ 0, & r_{ij} > c \end{cases} \tag{3.12}$$

It is assumed that c lies between the second and the third nearest neighbors interatomic distance. The three fitting parameters c_0, c_1, c_2 can be found using experimental data. Values of a lattice spacing a_0, cohesive energy E_g, bulk modulus B, and the three independent elastic constants C_{11},C_{12} and C_{44} have to be taken as an input to find these fitting parameters. In the case of vanadium, which has a bcc structure, the values of the above mentioned physical quantities measured at room temperature, are listed in Table 3.1. The elastic coefficients C_{11}, C_{12} and C_{44} are

Exp. quantities	Numerical values
a_0	3.0399 $\dot{A}$
E_g	5.31 eV
C_{11}	2.279 $eV/\dot{A}^3$
C_{12}	1.187 $eV/\dot{A}^3$
C_{44}	0.426 $eV/\dot{A}^3$
B	1.551 $eV/\dot{A}^3$

Table 3.1: Experimental data used for a semi-empirical FS potential of vanadium, from ref. [50,51].

taken from the compilation of Bujard [50] and the cohesive energy E_g of vanadium is taken from Kittel [51]. The resulting values for the two-body part of the FS potential are summarized in Table 3.2. The most important feature of the **FS** potential is its many-body part, which is explicitly non-central. Therefore this potential is capable to reproduce correctly the elastic constants. In addition to this property, it reproduces exactly the lattice parameters, bulk modulus, cohesive energy as well as the phonon dispersion, surface and vacancy formation energy in good agreement with experiment [42,44].

Despite the fact that the **FS** potentials were explicitly constructed in order to cope with highly defective systems, such as crystals with point defects, dislocations, cracks,

Fitting parameters	Numerical values
d	3.692767 $\dot{A}$
A	2.010637 eV
c	3.8 $\dot{A}$
c_0	-0.8816318
c_1	1.4907756
c_2	-0.397637

Table 3.2: Parameters of the FS potential of vanadium, taken from ref. [42].

etc., it was found by Rebonato et al.[52], that some modification must to be introduced to the original potential for bcc metals. It was discovered in series of MD and MC simulations that the original **FS** potentials [42], appear to give unphysical results for properties involving small interatomic separations, namely the pressure versus volume relation [52, 53]. Marshese et al. [53], have shown that the **FS** potentials predict thermal expansion that is too low in comparison with experiment and in some cases even *negative.* This flaw implies that the **FS** potentials may be inaccurate at large distortions of the lattice of the perfect crystal. In order to overcome these problems Rebonato et al.[53], suggested some modifications of the original **FS** potentials. The repulsive part of the**FS** potential was changed in the following way:

$$V_1(r) \longrightarrow V_1(r) + h(r) \tag{3.13}$$

where:

$$h(r) = \begin{cases} K(FND - r)^n, & r \leq FND \\ 0, & r > FND \end{cases} \tag{3.14}$$

with FND as the first-neighbor distance at zero pressure.

$$FND = a_0\sqrt{3}/2 \tag{3.15}$$

The many-body part was left unaltered.

These fitting parameters for vanadium are in Table 3.3 This modification improves

K $eV/\dot{A}^3$	n	FND $\dot{A}$
3.3	3.0	2.63271

Table 3.3: Parameters of the modified **FS** potential of vanadium, from ref. [53].

substantially the original **FS** potentials to get adequate description of the pressure versus volume behavior, as well as overcome high-compression instability of these potentials. The energies of the stable configurations of vacancies and self - interstitials, are in good agreement with experimental data [53].

Once a potential is selected the energy and the forces can be calculated. Usually the potential is cut-off at some distance to speed up the calculations. The cut-off determines the accuracy of the simulation results, the larger the cut-off length, the more accurate the results. Besides that, the length of simulation box has to be larger than the cut-off length. An atom i interacts only with the atoms, whose distance from the atom i is less than the cut-off length of the potential. The neighbors of the i atom are accounted by means of a neighbor list, which contains information about the environment of the ith atom. The neighbor list is usually extended beyond the cutoff length to avoid the need to update it in every simulation time step.

3.3 The initial and boundary conditions

The initial configuration in our simulations is set up by a distribution of all atoms at the sites of a bcc lattice. Afterward the point defects are introduced either by insertion of extra atoms in a random fashion between the lattice sites (self-interstitials), or by removal of atoms from the lattice sites (vacancies). These point defects are located at long distances one from another to avoid, as far as it possible, their interactions. The initial velocities of all atoms and point defects are set to values which depend on the temperature according to the Maxwell distribution. When these random velocities have to be adjusted again to ensure that the center of mass of the system is stationary, (i.e the velocity of the center of mass is equal to zero).

Molecular dynamics is applied to systems containing usually a few thousands atoms. Surface effects, i.e. interactions of atoms with the container wall or effects of the free surface, are dominant in such small systems. In simulations of the crystal bulk those surface effects are not of interest and may be eliminated by means of periodic boundary conditions. In order to implement the periodic boundary conditions for N atoms in a volume V, we imagine that the volume V is only a small part of the bulk material. The volume V is called the *computational (or primary) cell*, it represents the bulk material to extent that the bulk is assumed to be composed of the primary cell surrounded by exact replicas of itself. These replicas are called image cells. The image cells are the same size and shape. Thus, the primary cell is periodically replicated in all directions to form a macroscopic sample (See Fig. 3.1).

In order to simulate the surface the periodic boundary conditions have to be altered. We use periodic boundary conditions only in the x and y directions, which are parallel to the surface. The system is represented as a slab of layers oriented

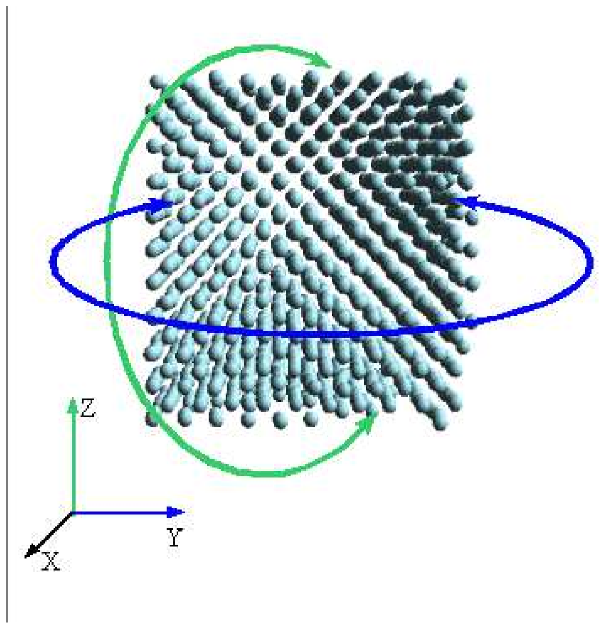


Figure 3.1: A computational cell and periodic boundary conditions

perpendicular to the z direction. Each layer contains atoms arranged in a bcc pattern. The bottom layers are fixed to mimic the effects of an infinite bulk of the solid.

3.4 Other numerical techniques: simulated annealing and simulated tempering

Sometimes, we are interested not in dynamics, but in static configurations of the system, generated by a potential, which describes particle interactions. Minimization techniques aim to find the configurations that minimize this potential (i.e. local and global minima). There are different numerical recipes, which can handle this problem and the most popular one is simulated annealing. Another interesting method, which was applied in our project, is simulated tempering. This method belongs to the wide class of the accelerated Monte-Carlo methods. These two methods have been applied to find the equilibrium configurations of point defects at non-zero temperatures in vanadium.

Let us begin with the *simulated annealing.* In this method an initial configuration, a random or a completely ordered one, is chosen and the system is heated up to a high temperature T. Then the system is cooled down in a very slow rate, which depends on specific system under investigation. The Monte-Carlo algorithm is used to equilibrate and to sample the system during this cooling process.

The sampling of the system is performed in the following way:

1.) A random atom selected and displaced randomly, therefore a new configuration of the atoms is generated. This displacement is called a *trial move.*

2.) The potential energy of the new configuration of the system E_{tr} is calculated and compared with the potential energy of the system before the trial move E_{bf}.

3.) If the energy of the system after the trial move is lowered $E_{tr} < E_{bf}$: the trial move accepted *unconditionally.*

4.) But, if the energy of the system is increased, than this trial move could be accepted with a certain probability, i.e. we generate a random number between $\eta \in [0, 1]$ and compare it with $\exp(-(E_{tr} - E_{bf})/kT)$.
If this expression larger than the $\eta < \exp(-(E_{tr} - E_{bf})/kT)$ trial move accepted, otherwise it is rejected.

5.) This procedure repeats again and again.

This method allows us to escape from local minima, while we are sampling the phase space and looking for the global minimum. However, we can not be confident in general, that using this method we will not be trapped in some local minimum, due to limited time of our Monte-Carlo run (and therefore we can not prove that the the global minimum is necessary found). If we are trapped in a local minimum most of the simulation time, then only a small part of the phase space will be explored, and hence the physical quantities will not be calculated with appropriate accuracy (pure statistics).

One of effective way to overcome those difficulties is to perform simulations in a so called *generalized (extended) ensemble*, where the probability to cross energy barriers (get away from local minima) could be increased in an artificial way, thus MC simulations can be sufficiently accelerated. The *simulated tempering* method, which belongs to the class of *MC accelerated algorithms* [69], has been applied in studying of point defects configurations at room temperature. This methods was originally invented by Lubartsev et al. [70], and independently about at the same time by Marinary and Parisi [71]. In simulated tempering we perform a random walk in the energy space, like in standard Monte-Carlo method, as well as in the temperature space. The temperature space, unlike the energy space, is chosen in a special way, i.e. it has a discrete structure:

$$T_i \in \{T_1, T_2, .., T_n\} \tag{3.16}$$

We start with an initial temperature T_{init}, belonging to the set of the temperatures $\{T_i\}$, and perform a usual MC simulation in the energy space:
a particle is picked up randomly, its position changed, and the new configuration is accepted or rejected according to the Metropolis scheme. After several thousands of MC steps in the energy space we try to update the current temperature, i.e. to perform random walk in the temperature space. The current temperature could be increased or decreased to the value of the nearest neighbor in the temperature

space. The higher the temperature, the larger the probability to escape form a local minimum. In order to accept or reject the temperature update, we have to introduce a criterion for acceptance (or rejection) of these updates. This can be done by extension of the canonical Metropolis scheme. We introduce a new probability distribution $P(\{x\},\beta)$, $\beta = 1/(k_bT)$ for the extended ensemble $(\{x\},\beta)$, where by $\{x\}$ we denote coordinates of the atoms:

$$P(\{x\}) \approx \exp(-\beta E(\{x\}) \longrightarrow P(\{x\},\beta) \approx \exp(-\beta E(\{x\} + g(\beta)) \tag{3.17}$$

where an arbitrary function $g(\beta)$, that controls the $\{\beta_i\} = \{1/(k_bT_i)\}$ distribution is introduced. The function $g(\beta)$ is determined for the set of $\{\beta_i\} => g(\{\beta_i\})$, and we can use the Metropolis update when we try to change the current temperature. The algorithm steps are following:

1.) Trial move - change temperature or β_i:

$$\begin{cases} \beta_i \longrightarrow \beta_{i+1} \\ or \;\; \beta_i \longrightarrow \beta_{i-1} \end{cases} \tag{3.18}$$

2.) Calculate Δg :

$$\begin{cases} \Delta g_{i,i+1} = g(\beta_{i+1}) - g(\beta_i) \\ \Delta g_{i,i-1} = g(\beta_{i-1}) - g(\beta_i) \end{cases} \tag{3.19}$$

3.) If $\Delta g_{i,i\pm1} < 0$ accept it

4.) Otherwise, pick up a random number η from $\eta \in [0,1]$.

$$\begin{cases} if \quad \exp(-\Delta g_{i,i\pm1}) \geq \eta \quad accept \;\; it \\ else \quad \exp(-\Delta g_{i,i\pm1}) < \eta \quad reject \;\; it \end{cases} \tag{3.20}$$

5.) Back to the usual MC steps.

The only question is how to determine $g\{\beta_i\}$ function? Well, unfortunately, there is no prescription how to find this function in general. Actually, for each system there is its own special $g(\beta)$ function, which reflects the system properties. This function has to be chosen very carefully, if we want to be efficient, otherwise the system gets stuck around of some, usually uncontrolled value of β, and there is no good sampling in the temperature space, and we return to a standard Monte-Carlo scheme. The simplest way to choose $g(\beta_i)$ is following:

$$g(\beta_i) = -log(Z(\beta_i)) \tag{3.21}$$

where $Z(\beta_i) = exp(-\beta F)$ is the state sum. In this case the system visits all points of the temperature space and for a long computation run spends at each value of $\{\beta_i\}$ approximately the same amount of time. But how do we know value of $Z(\beta)$ **a priori** before the simulation? Any algorithm that requires the value of $Z(\beta)$ as inputs appears to be unrealistic because $Z(\beta)$ is usually unknown quantity that is computed , for example, by means of Monte Carlo simulation. The simple solution is to perform a preliminary MC run, calculate $Z(\beta)$ and find the initial parameters of $g(\beta_i)$ which will be optimized by iterations in several consequent runs. Hence it is an algorithm which learns to find the optimal values of $g(\beta_i)$ iteratively by means of the preliminary runs [69-71].

3.5 Analysis of errors

It is often stated that a computer simulation generates “exact” data for a given model. However, this is true only if we can perform an infinitely long simulation, which is impossible in practice. Therefore, the results of simulations are always subject to statistical errors which have to be estimated. Besides that, finite-size effects, unreliable generator of random numbers, inaccurate potential and numerical techniques have to be taken as sources of systematic errors into account.

The main source of systematic errors could be the interatomic potential, which is in our case the Finnis-Sinclair potential. The validity of the potential is tested in our simulations, and it was found that the potential is sufficiently reliable and gives adequate description of vanadium. The predictor-corrector methods is accurate enough, provided the time step is chosen appropriately [55,59]. A random number generator proposed by Ziff et al. [60] which is used in our simulations was checked earlier and considered to be very dependable [60].

Special attention was payed to statistical errors. In order to gather better statistics, we repeated our simulations with different initial conditions several times, e.g. the distribution of the initial velocities of all atoms, and positions of point defects were changed using different seeds for the random number generator. Various correlation functions were monitored during our simulations, and corresponding characteristic decay time of these functions were estimated to make sure that equilibrium is indeed achieved. An average statistical error is calculated according to recipes of statistical analysis:

$$\Delta\overline{x} = \frac{s}{\sqrt{k}} \tag{3.22}$$

where s is a standard deviation:

$$s = \sqrt{\frac{\sum_{i=1}^{k}(x_i - \overline{x})^2}{k-1}} \tag{3.23}$$

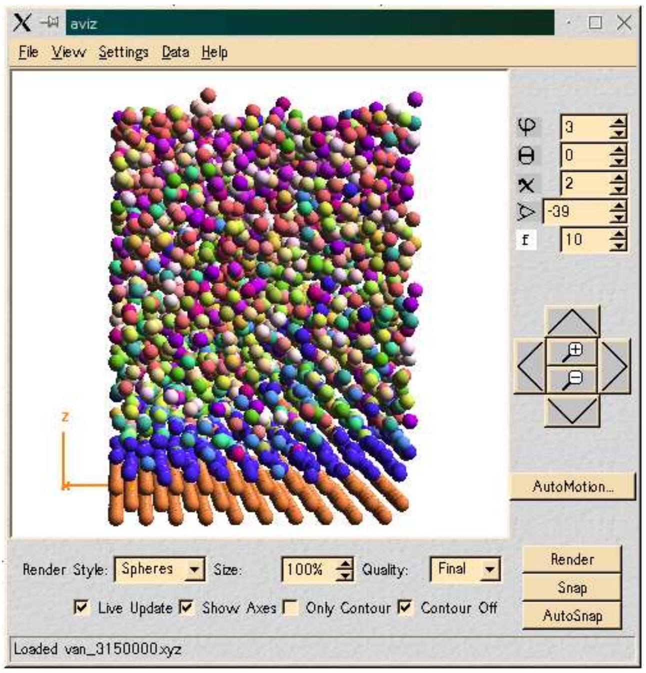
and $\overline{x}$ is a simple arithmetic mean value:

$$\overline{x} = \frac{1}{k}\sum_{i=1}^{k} x_i. \tag{3.24}$$

3.6 Visualization of MD and MC simulations

Visualization was essential for the development of this project. The complicated geometries of the crystal surfaces and point defect configurations inside the bulk, and the correct implementation of periodic boundary conditions are best tested by visualization tools. Once the simulation programs were prepared and debbuged, the visualization of the intermediate and final states helped identify phenomena suitable for a quantitative studying. In one word, it may be stated that *"a picture is worth a thousand words"*.

The Atomic Visualization package (**AViz**) was used extensively in all stages of this project. Our computational physics group developed the **AViz** package [72]. This is a very powerful visualization tool which helps to enhance the 3D perception. It includes a lot of various options, which let one to rotate the still sample, change relative sizes of atoms, create animations and movies, add and remove the bonds and borders of the sample, use color coding, slice of the sample and much more. The *user-friendly interactive interface* (See Fig. 3.2) simplifies the use of all these options in the visualization of computer simulations.



Render Style: Spheres Size: 100% Quality: Final

[✓] Live Update [✓] Show Axes [ ] Only Contour [✓] Contour Off

Loaded van_3150000.xyz

Figure 3.2: Atomic Visualization package developed by the Computational Physics Group, Technion.

Chapter 4

Results: bulk melting transition

4.1 Bulk melting: the objectives of the research

The main question, that is addressed in this part of our research, is how point defects influence on properties of the solid, and especially on its bulk melting temperature. To answer this question we investigated the various properties of a crystal of vanadium, containing point defects at different concentrations, as a function of temperature. The first task was to investigate the "structure" of point defects, i.e. to find all possible configurations of atoms around a self-interstitial, and choose the most stable one (at low temperatures). The second objective was to find whether the various properties of the solid are affected to the same extent by self-interstitials and vacancies. Futhermore, to gain a better understanding of the rôle of the lattice structure a comparison was made between the results of our simulations (vanadium, bcc lattice) and the results obtained by A. Kanigel et. al [67] (copper, fcc lattice).

A. Kanigel et al. discovered that the bulk melting temperature of copper depends on the concentration of point defects (namely self-interstitials). Point defects expand the volume of the solid to a critical value at which the mechanical melting transition is triggered. According to A. Kanigel et. al the bulk melting transition is *lowered by point defects!* However, it was not clear if this mechanism of bulk melting is universal or it is specific for the fcc lattice of copper. Therefore, the primary aim of our investigation was to establish whether this mechanism is applicable for vanadium and point defects could lower the bulk melting temperature of this solid.

4.2 Investigation of the properties of a perfect crystal of vanadium

The aim of this project is to investigate the rôle of point defects in the bulk melting transition. With this aim in mind we examine the properties of a perfect crystal of vanadium before introducing defects into the bulk of the solid. We have chosen a set of physical properties of the solid, which are relevant in studying of melting transition.

First of all, a **structure order parameter** η was considered which can help in distinguishing between the crystalline and liquid phases, as well as in the detection of the melting transition. The structure order parameter is defined as:

$$\eta = \left\langle \frac{1}{N} \left| \sum_{i=1}^{N} exp(i\vec{k}\vec{r_i}) \right| \right\rangle \tag{4.1}$$

where: $\vec{k} = [0, 0, \frac{2\pi}{a}]$ is a vector of the reciprocal lattice, $\vec{r_i}$ is a vector pointing on the atom i, N is the number of the atoms in the system, $<>$ is ensemble average. The structure order parameter is unity at zero temperature, when all atoms are at rest at their lattice sites (See 4.2), but at non-zero temperatures $T \leq T_m$ its value is less than unity, $0 < \eta < 1$ due to the atom vibrations and defect formation (See Fig. 4.3). This parameter fluctuates around zero ($\eta \simeq 0$) for a molten crystal $T \geq T_m$, when the bcc structure is completely lost (See Fig. 4.1). The shape and the volume of the sample

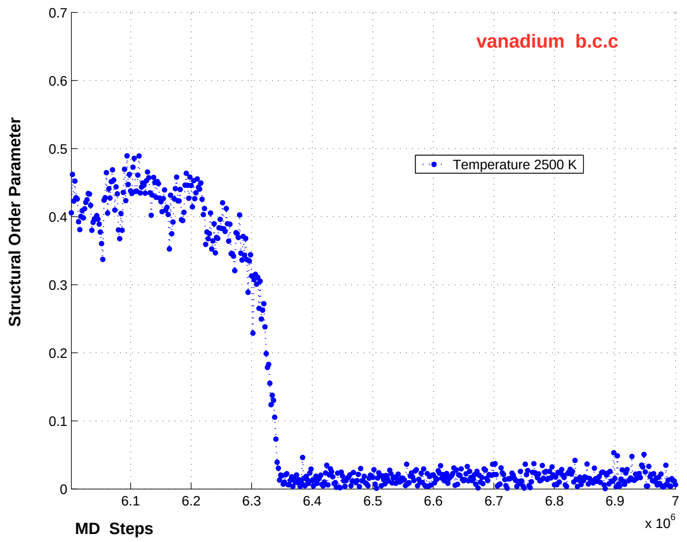


Figure 4.1: Evolution with time of the structural order parameter at melting transition, initial conditions are those of a solid at $T = 2500K$.

at various temperatures are other properties of interest to us. Those properties could be investigated by means of the Parinello - Rahman (**PR**) method [68]. Using this method we inspected the diagonal as well as the off-diagonal elements of the $H_{\alpha\beta}$ matrix which describes the shape and the volume of the sample (See Appendix D for more details). The elements of $H_{\alpha\beta}$ were recalculated in each MD step, and their mean values were obtained by averaging over more than $4,000,000$ steps in a steady-state. The variation of the diagonal elements, of a sample containing 2000 atoms is shown in Figure 4.4, and the variation of the off-diagonal elements are represented in Figure 4.5. We found that the shape of the computational cell, at zero stress and pressure, is a cubic one. The values of the off-diagonal elements fluctuate around zero, and the difference between the diagonal elements of the $H_{\alpha\beta}$ matrix is negligible.

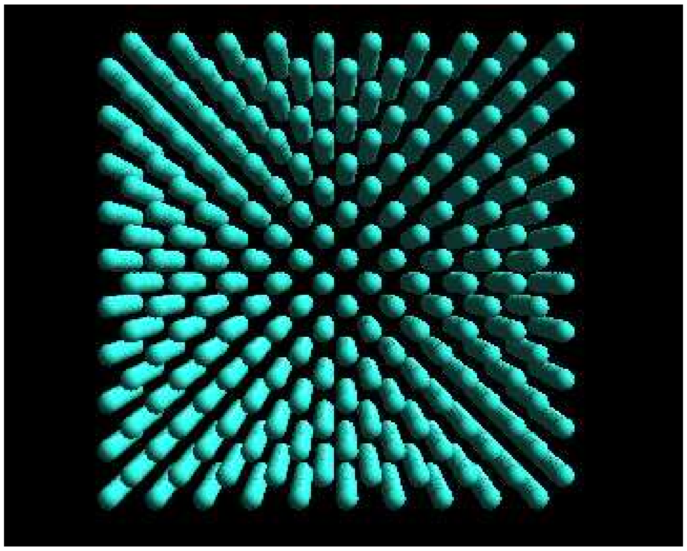


Figure 4.2: A perfect bcc lattice of vanadium at $T = 0K$, 2000 atoms.

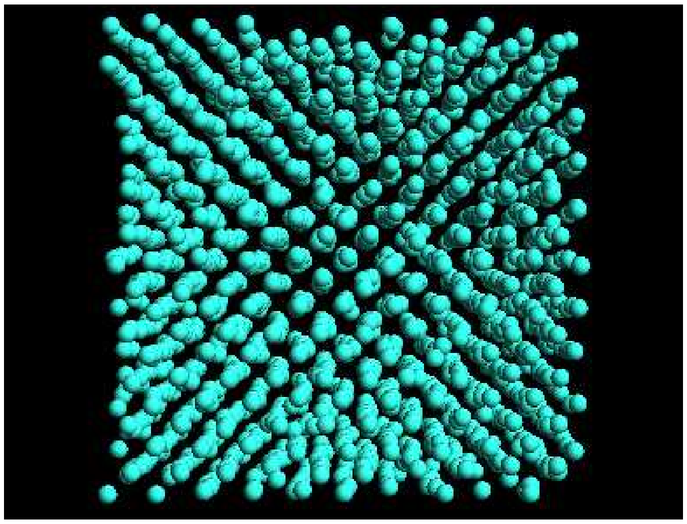


Figure 4.3: A bcc lattice of vanadium at T=2100K, 2000 atoms.

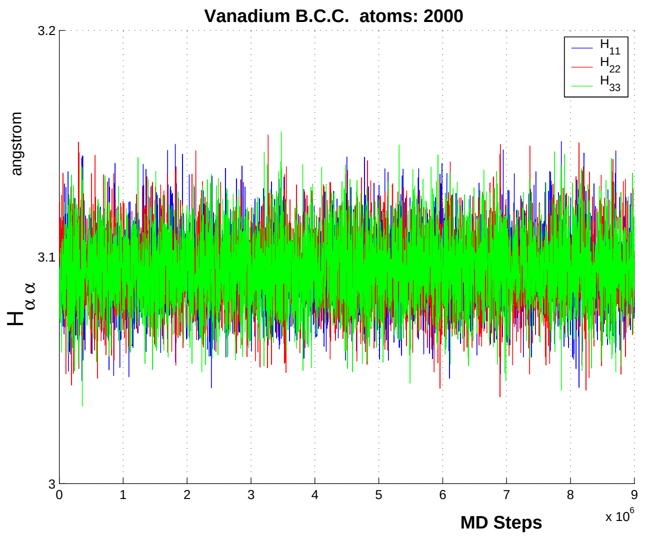


Figure 4.4: Variation of the diagonal elements of $H_{\alpha\beta}$ in time at $T = 2450$ K.

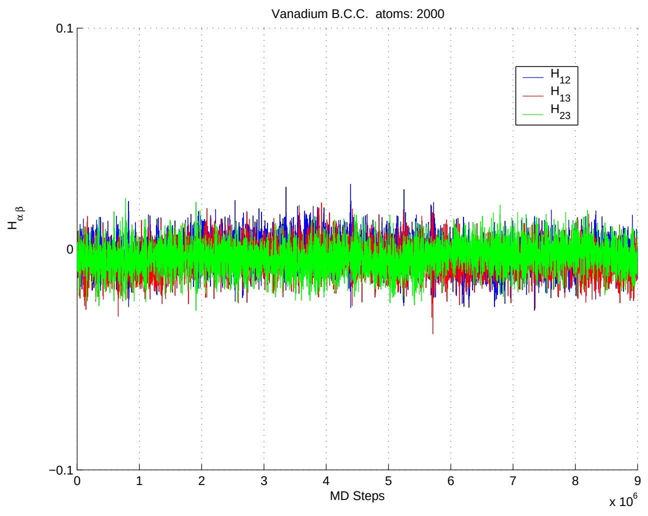


Figure 4.5: Variation of the off - diagonal elements of $H_{\alpha\beta}$ in time at $T = 2450$ K.

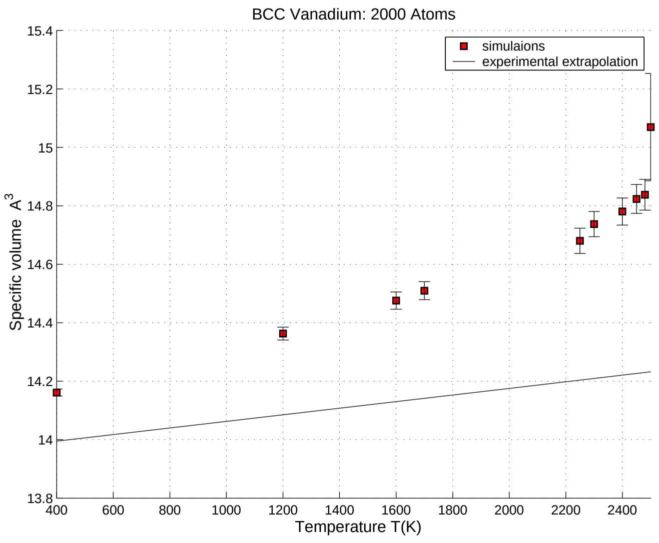


Figure 4.6: Specific volume of a sample with 2000 atoms as function of temperature in comparison with extrapolated experimental data.

Knowledge of the elements of the matrix $H_{\alpha\beta}$ allows us to calculate the volume of the sample $V = det(H_{\alpha\beta})$ as a function of temperature (See Fig. 4.6). Relying on these data the thermal expansion coefficient is estimated

$$\alpha_t = \frac{1}{V}\frac{\partial V}{\partial T} \tag{4.2}$$

We found that the thermal expansion coefficient at low temperatures is $\alpha_{calc} = (18 \pm 6) \times 10^{-6}$ $[1/K]$ and its value is the same order as the experimental value measured at room temperature $\alpha_{exp} = 8.6 \times 10^{-6}$ $[1/K]$. The thermal expansion of vanadium was studied experimentally only at the room temperatures, far from the melting point. Thus, in order to compare the results of our simulations at high temperatures with the experiment we would have to extrapolate the available experimental data to the high temperature region. The simplest approximate extrapolation formula for the thermal expansion was applied:

$$v(T) = v_0 \exp(\alpha_{\exp}(T - T_0)) \tag{4.3}$$

where v_0 is the molar volume at room temperature T_0 (See Fig. 4.6). The calculated values are closer to the experimental data at low temperatures. This result could be expected, since the lattice constant a measured at that temperatures was used as an input parameter for the **FS** potential. At higher temperatures deviation from the experiment is larger, possibly due to the large anharmonicity introduced by the **FS** potential.

The above mentioned quantities: the structure order parameter, the shape and volume as well as the total energy and pressure of the system were calculated in the framework of the Parinello Rahman method (NtT ensemble). The bulk melting temperature (strictly speaking its upper limit) was also estimated by slowly heating the perfect sample up to the temperature at which the crystal lattice becomes unstable and collapses. At the same time the order parameter jumps to zero and the shape of the sample changes from cubic to tetragonal. In addition, we calculated the shear elastic moduli $C_{\alpha\beta}$ using method proposed by Parinello and Ray (NVT ensemble) (See Appendix C). The stress tensor fluctuations as well as the mean value of the Born term were calculated to find the shear elastic coefficients. The Born term converges very fast in comparison with the stress tensor fluctuations which reach their equilibrium value at a given temperature after a very long time (about $1,500,000$ steps in average). The variation of the elastic coefficients as a function of MD time steps is shown in Fig.4.7

4.3 Bulk melting and point defects

After investigation of a perfect crystal of vanadium, point defects, i.e. the simplest structural imperfection in solids, are introduced either by removal of atoms (vacancies)

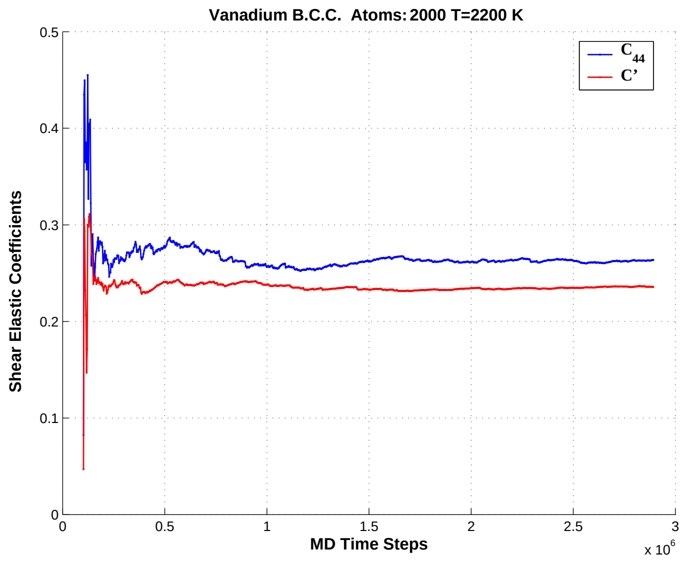


Figure 4.7: The shear elastic moduli vs. time at temperature 2200 K.

from lattice sites or by insertion additional atoms of the same kind (self - interstitials) between the lattice sites (See Fig. 4.8). Initially these point defects are distributed homogeneously inside the bulk of the solid. In our simulations we introduced point defects of one type only to avoid their mutual annihilation or recombination. The off - lattice Monte Carlo method, namely simulated tempering, was implemented to find the most stable configuration of atoms in the vicinity of a point defect inside the bulk at low temperatures. Our simulations were carried out for a sample containing 128 atoms plus a self-interstitials (128 + 1), and the temperature set was chosen to be $\{75\ K, 80\ K, 85\ K, 90\ K\}$. The most energetically favored configuration was found to be the $< 011 >$ *dumb-bell split - interstitial* with a formation energy of $E_f = 4.18 \pm 0.02\ eV$ (See Fig. 4.9). The defect formation energy was calculated in following way:

$$E_f = E(N-1,1) - E(N,0) \tag{4.4}$$

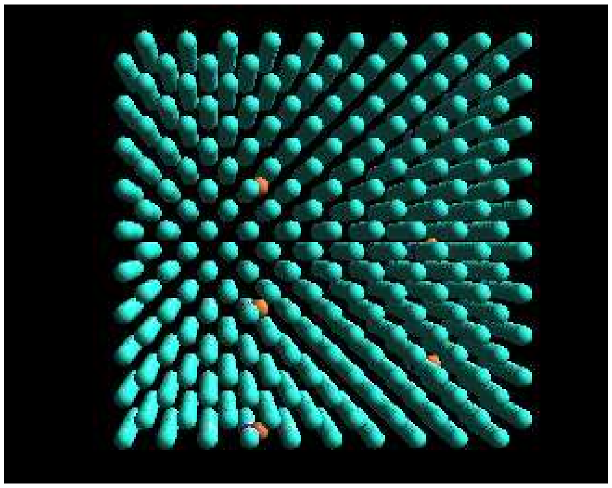


Figure 4.8: An initial configuration of 5 interstitials in a sample with 2000 atoms.

where $E(N, N_{def})$ is the potential energy of a sample which contains N lattice atoms and N_{def} point defects. The potential energy of a perfect crystal with N atoms is given by:

$$E(N, 0) = NE_{coh} \tag{4.5}$$

where E_{coh} is the cohesion energy per atom calculated for a pure sample. The calculated value of the defect formation energy $E_f = 4.18eV$ is close enough to the results obtained by Ackland et. al [73] using the DEVIL program which based on the conjugate-gradient method. (See Table 4.1).

Type of split - interstitial	Formation energy, eV
$< 001 >$	4.963
$< 011 >$	4.163
$< 111 >$	4.608
crowdion	4.6

Table 4.1: Defect formation energy of various split-interstitial defects, from ref. [73].

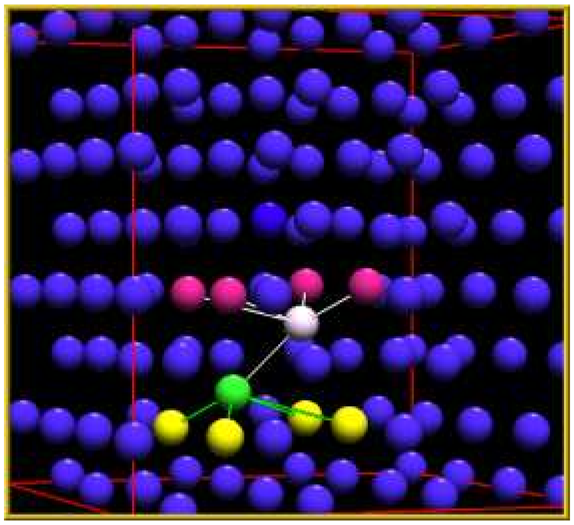


Figure 4.9: A $< 011 >$ dumb - bell split - interstitial in a bcc metal vanadium.

Other possible configurations (octahedral, tetrahedral, crowdion) possess larger defect formation energy, and therefore they are less energetically favored and less stable. In our simulations we implemented various initial configurations:either we started very close to the most stable configuration, i.e. $< 011 >$ dumb-bell split - interstitial, or inserted an additional atom in a random fashion between the lattice sites (See Fig. 4.10) In each case, the configuration with lowest formation energy was found (See Fig. 4.11). When the most stable configuration of point defects inside the bulk of vanadium was found, we began to study how point defects influence the various properties of vanadium which is interested to us. To simulate bulk properties of vanadium we prepared various samples with different concentrations of point defects. The **MD** simulations were performed in the **NtT** ensemble by using the **PR** method. We found that introduction of point defects leads to the structural disordering (see Fig. 4.12). Increase in the concentration of self - interstitials results in noticeable decrease of the structure order parameter η, while the same effect of vacancies is

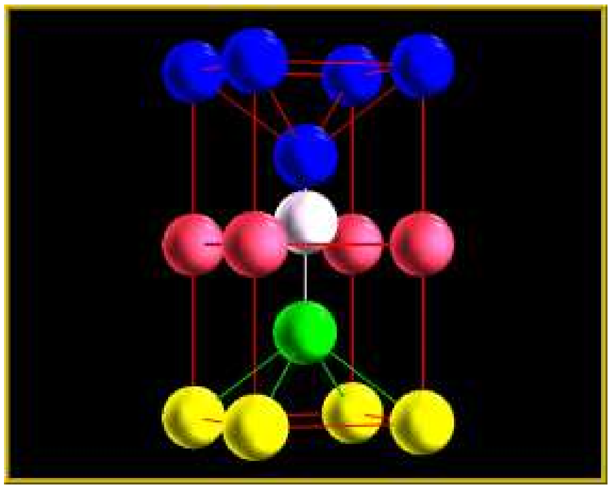


Figure 4.10: An initial configuration: an interstitial (white color) and its neighbors.

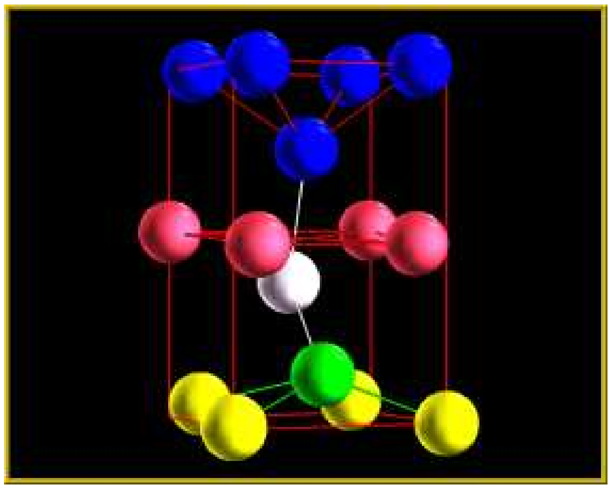


Figure 4.11: Equilibrium configuration of the $< 110 >$ split interstitial.

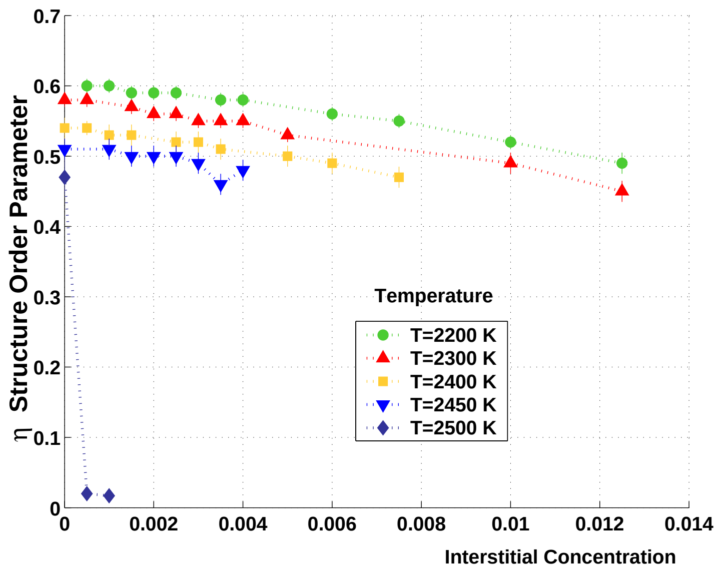


Figure 4.12: Structure order parameter as a function of concentration of self-interstitials at several temperatures.

relatively weak (See Fig. 4.13). Self - interstitials expand the volume of the sample as it is shown in Fig. 4.14 where the lattice parameter $a \simeq (V/N)^{1/3}$, while vacancies lead to decrease of the volume (See Fig. 4.15). It is interesting to compare the dependence of the specific volume on the concentration of self - interstitials for vanadium (bcc lattice) and copper (fcc lattice). In both cases the specific volume is increased, but for fcc lattice the effect is more noticeable. This effect can be attributed to the more compact structure of the fcc lattice, where even a small concentration of self - interstitials lead to a large distortion of the fcc lattice, and therefore is more noticeable in comparison with the bcc lattice.

The next stage of our bulk simulations is the most important one - investigation of the rôle of point defects in bulk melting transition. According to the Born criterion [9, 16] bulk melting transition takes place when the specific volume of the crystal reaches a critical value. As shown by Kanigel et. al [17,67] and it does not matter

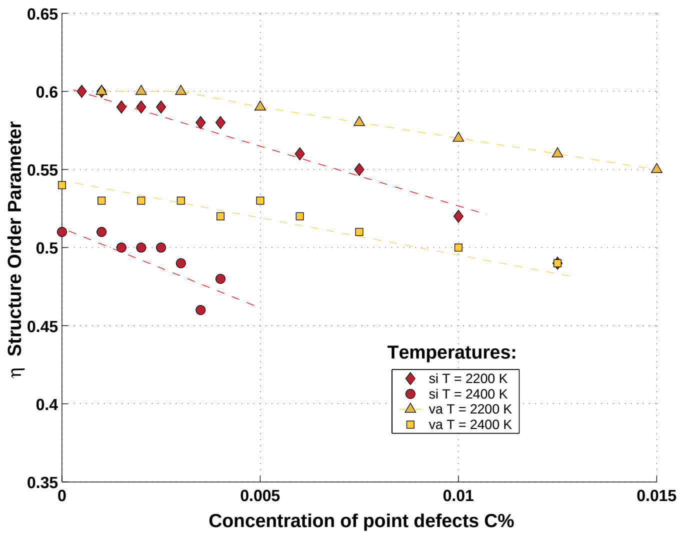


Figure 4.13: Structure order parameter as a function of point defect concentration. A comparison between interstials and vacancies. The lines to guide the eye.

in which way the critical value is reached. The critical volume at which crystal melts could be attained either by heating of the sample or by doping it with point defects at a constant temperature which leads to the expansion of the sample and in the end to melting. The temperature at which melting occurs can lower than the bulk melting point of a perfect sample, i.e. *point defects lower the bulk melting temperature!* In this sense the mechanical melting process is universal, e.g. it determined only by the sample expansion up to the critical volume.

In our simulations we verified that theoretical prediction. We prepared samples with a specified concentrations of point defects, and heated them gradually up to the melting point. In this way the value of the critical volume and the bulk melting temperature T_b was obtained. By repeating this procedure for various defect concentrations we found the dependence of the bulk melting temperature on the concentration of point defects.

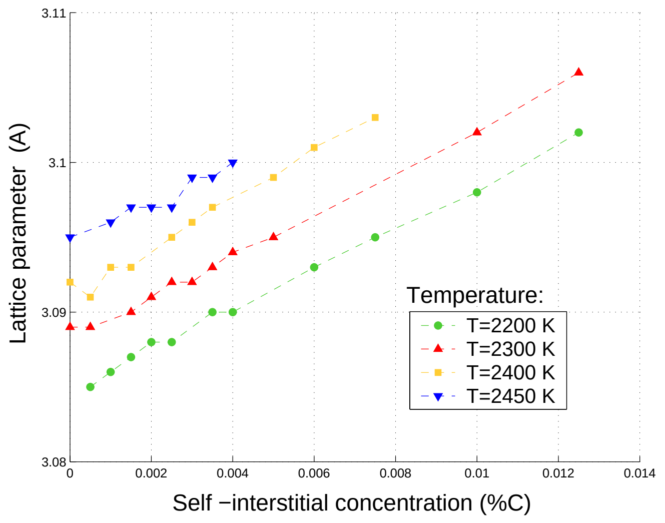


Figure 4.14: The lattice parameter a as a function of the concentration self - interstitials.

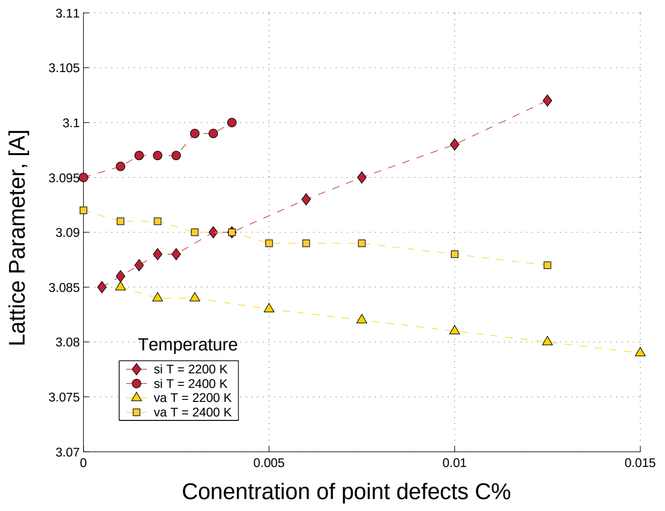


Figure 4.15: The lattice parameter a as a function of the point defect concentration. A comparison between self - interstitials and vacancies.

The initial temperature is far below the melting point of a perfect sample $T \simeq 0.7T_b$, (the bulk melting temperature $T_b = 2500 \pm 5\ K$ is calculated in the simulations of perfect sample). After each 100,000 MD steps we increased the temperature by 100 K until we reached 2100 K. After that the temperature was increased by in a smaller step of 50 K followed by 200,000 MD steps. In the end we reached the temperature of 2400 K, and from this point and onward we increased the temperature incrementally by 10 K. In this region each sample configuration (positions and velocities of all atoms) was saved before the elevation of the temperature, in order to use the stored configurations again if needed. The number of MD steps between two successive temperature changes was increased to 2,000,000 MD steps. At some temperature we observed an abrupt decrease of the structure order parameter, and a drastic increase of the total energy and the volume of the sample (See Fig. 4.16 and Fig. 4.17).

At that temperature one sees a sharp bifurcation in the lattice dimension where

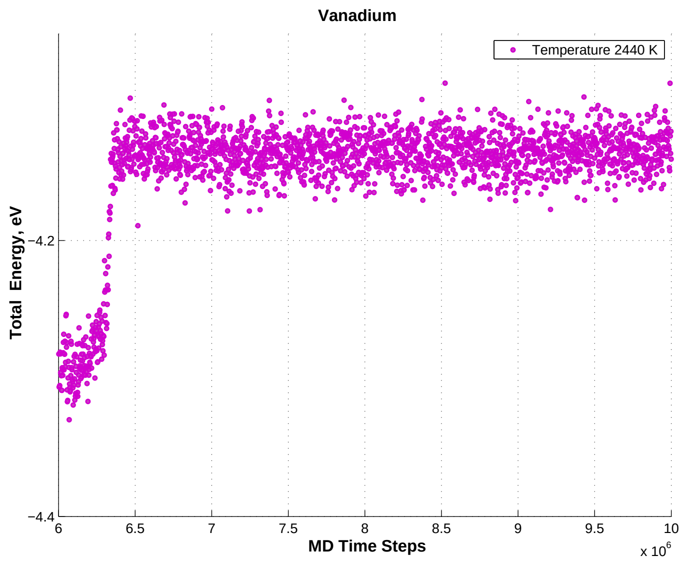


Figure 4.16: Increase of the total energy at the melting point.

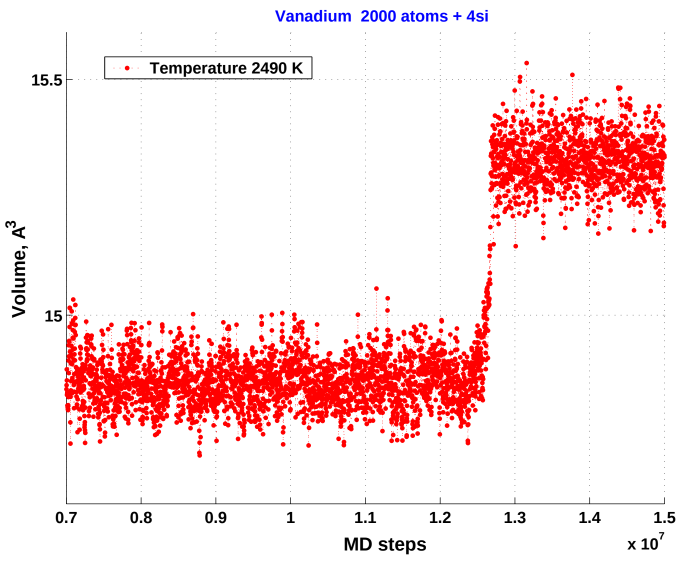


Figure 4.17: Jump of the sample volume at the melting transition.

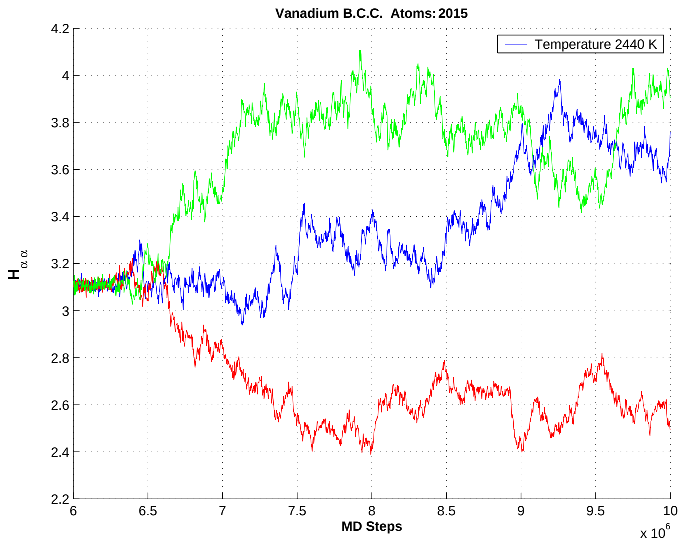


Figure 4.18: Variation of the diagonal elements of the $H_{\alpha\beta}$ matrix at the bulk melting transition.

the system elongates in two directions and contracts in the third (See Fig. 4.18). This is a clear sign of symmetry change, from cubic to tetragonal. The same effects were observed at melting transition of fcc metals [63]. Bulk melting transition occurs during a very short time scale corresponding approximately to the several vibration periods of atoms.

It is not improbable that we do not encounter the true bulk melting temperature, but find only its upper limit. It is not known in advance how long the simulation has to be carried out before the expected phenomenon will be observed. There is a possibility that we missed the melting point during the heating of the sample and the melting transition would occur at a lower temperature, provided we could run our simulations for a longer time. Therefore, after the upper limit is detected, we returned to one of the previous configurations. The temperature of the recovered configuration is lower than the bulk melting temperature, but close enough (actually we took the closest one). The simulation were repeated for a quite long time up $15,000,000$ MD steps, in the hope to observe a possible melting transition. If the transition was observed, we repeated the procedure again.

The results of the various simulations performed at the different temperatures and the defect concentrations can be summarized in a phase diagram (See Fig. 4.19). We see that increase in concentration of the self - interstitials leads to decrease of the bulk melting temperature, while the vacancies almost do not affect the bulk melting temperature, at least if their concentration is small. The same effect of decrease of the bulk melting temperature induced by point defects was obtained by A. Kanigel [67] for the fcc metal copper (See Fig. 4.20). If we compare the phase diagram of vanadium with the phase diagram of copper, then it can be immediately seen that the melting point of vanadium is lowered by 50K at a concentration of self-interstitials

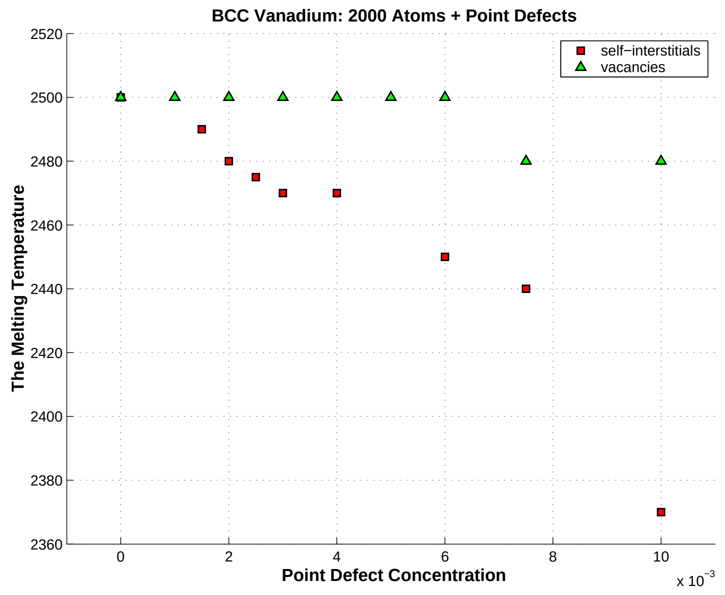


Figure 4.19: The influence of point defects on the melting temperature of vanadium obtaining using periodic boundary conditions.

about 0.006%, while the melting point of copper is lowered by 80K at the same defect concentration. Interstitials in copper induce larger distortion of its close packed fcc lattice, than interstititials in vanadium. Therefore, the specific volume of copper is increases more than that of vanadium, and a smaller temperature increase is needed to expand copper up to the critical volume, at which melting occurs.

In conclusion, some additional remarks are necessary:
First, the concentration of point defects in the simulations is increased up to a high enough value $n_{def} \simeq 10^{-2}$ $[defects]/[atoms]$ to see their possible effect on the bcc lattice of vanadium. This is unrealistically large value in comparison with the typical laboratory values $n_{def} \simeq 10^{-6}$ $[defects]/[atoms]$. At these high defect concentrations the effect of self - interstitials and vacancies could not be considered as a simple lattice expansion in analogy with the thermal expansion. One has take interactions between

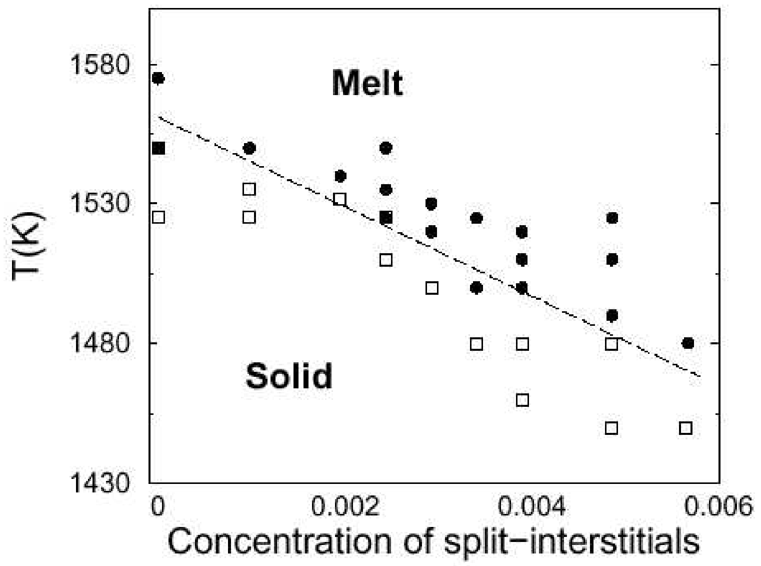


Figure 4.20: The influence of point defects on the melting temperature of Cu, a sample of 1372 atoms, from ref. [67].

these defects into account, which alter in some way the physics of the phenomenon Second, we can not neglect the fact that MD simulation with the Noose - Hoover thermostat is plagued by temperature fluctuations due to the small sample size. That means it is hard to approach T_b to an accuracy of better than about $\sim$ 1%. In summary, it could be noted that the calculated phase diagram is qualitative, because of the finite sample size and the finite time of our computer simulations.

4.4 Influence of interstitials on the shear moduli

Some time ago it was shown that for the bulk melting transition a strong correlation exists between the volume dependence of shear elastic coefficient and melting. It was discovered [16,67] that the volume of the melt at the melting point can be predicted by a continuous extrapolation of the shear modulus to zero as a function of volume (at zero external stress). In another words, a bulk melting transition will occur in an infinite crystal when it will be expanded up to a critical volume (molar volume of the liquid phase) at which the shear moduli (or at least one of them) will vanish. This observation lends considerable credibility of the Born mechanism of melting. According to the Born criterion, mechanical melting at zero external stress, is described by the criterion $C' = 0$, either in perfect or imperfect crystal containing point defects. This could be verified directly in MD simulations by calculation of the shear elastic moduli at various concentrations of point defects. For any concentration of defects the number of atoms and unit cells in the sample is kept constant in our simulations, so that the number of externally introduced defects is conserved. Under this constraint, atoms and point defects are free to diffuse, agglomerate, etc.

The simulations at various temperatures and concentrations showed that the shear moduli C' and C_{44} decrease when temperature or/and concentration is increased (See Figs. 4.21 -4.24). The common trend is that self - interstitials induce a noticeable softening of the shear moduli. This softening is anisotropic, in the sense that C_{44} softens more than C'. The absolute change of the shear elastic coefficients per percent of interstitals is larger for C_{44} at the high temperatures, as summarized in Table 4.2.

Temp(K)	$\Delta C'/\Delta S \quad (GPa/\%inters)$	$\Delta C_{44}/\Delta S \quad (GPa/\%inters)$
2100	$-8.1 \ \pm 2.6$	$-11.3 \ \pm 0.5$
2300	$-6.27 \ \pm 1.8$	$-8.97 \ \pm 1.0$
2400	$-5.25 \ \pm 2.4$	$-10.07 \ \pm 0.7$

Table 4.2: Change in shear elastic moduli C' *and* C_{44} of vanadium per percent of self - interstitials (S) at various temperatures.

The same effect of anisotropy in softening of the shear moduli was observed for copper [67] (See Table 4.3). Qualitatively, a similar dependence of the C' and C_{44} on point

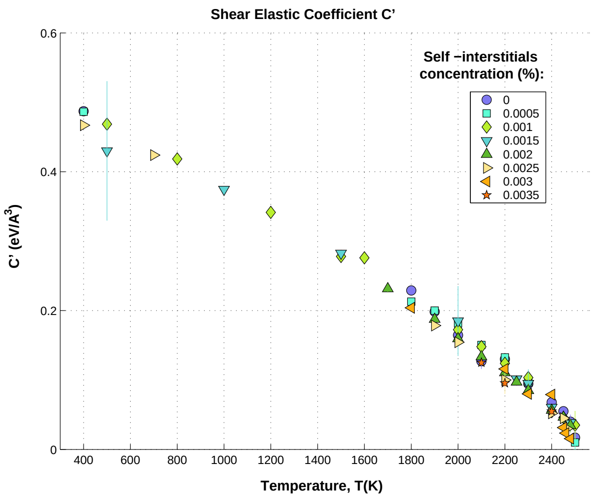


Figure 4.21: Variation of C' with temperature.

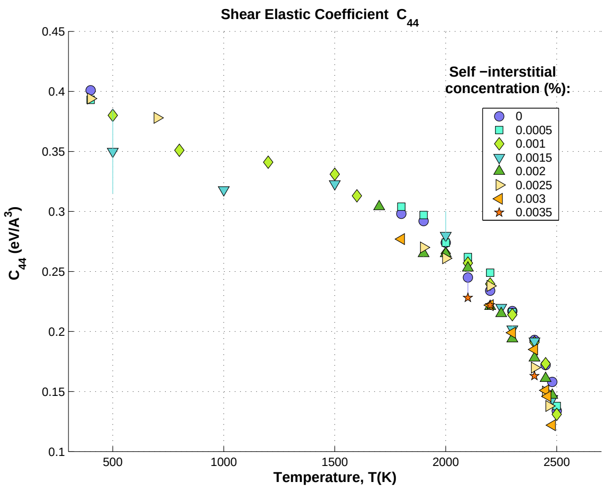


Figure 4.22: Variation of C_{44} with temperature.

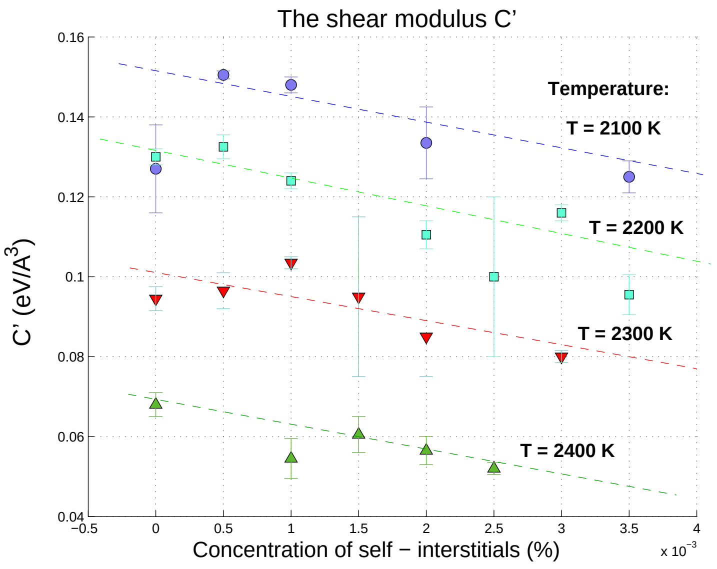


Figure 4.23: Variation of C' with self-interstitial concentration. The dashed lines to guide the eye.

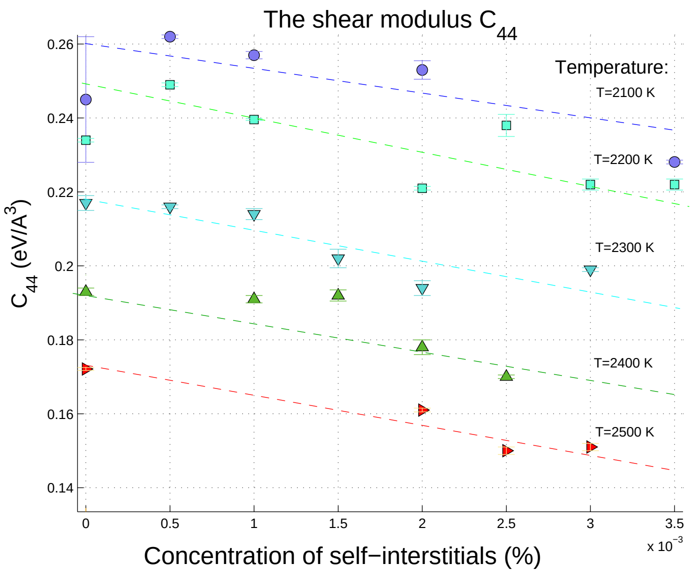


Figure 4.24: Variation of C_{44} with self-interstitial concentration. The dashed lines to guide the eye.

Temp(K)	$\Delta C'/\Delta S \quad (GPa/\%inters)$	$\Delta C_{44}/\Delta S \quad (GPa/\%inters)$
1400	$-3.5 \ \pm 2.0$	$-23 \ \pm 1.7$
1450	$-4.1 \ \pm 2.0$	$-18 \ \pm 2.5$
1480	$-7.8 \ \pm 3.5$	$-15.5 \ \pm 1.3$

Table 4.3: Change in the shear elastic moduli C', C_{44} of copper per percent of self - interstitials (S) , obtained by A. Kanigel et al. [67].

defects concentration is obtained (See Figs. 4.25 and 4.26), yet quantitatively bcc lattice is less softened by the presence of self - interstitials, especially for the shear modulus C_{44}.

We compare the change of the shear elastic moduli caused by point defects with the similar change induced by thermal expansion of a perfect sample, i.e. we heated a sample without point defects up to a temperature at which its volume is equal to the volume of a sample with point defects but at a lower temperature and compared the shear moduli. We knew from the simulations in canonical ensemble (**NVT**), how the shear moduli depend on the concentration of interstitials. Dependence of the specific volume on the concentration of defects was obtained by means of the **PR** method. If we take into account that there is a one-to-one correspondence in both cases, then we can find how the shear moduli depend on the specific volume (See Fig. 4.27).

We found that the dependence of the shear moduli on the specific volume is the same whether the volume expansion is induced by the thermal expansion or by point defects. That is to say that effect of interstitials is mainly to expand the bcc lattice of vanadium, and when the critical volume is reached the solid melts. It was obtained in the **PR** simulation of melting that specific volume per atom at the melting point is $v_{cr} = 15.02 \pm 0.1 \ \dot{A}^3$. An identical value (within the stated accuracy) of $v_{cr} = 14.95 \pm 0.08 \ \dot{A}^3$ is obtained by extrapolating the dependence of C' on specific

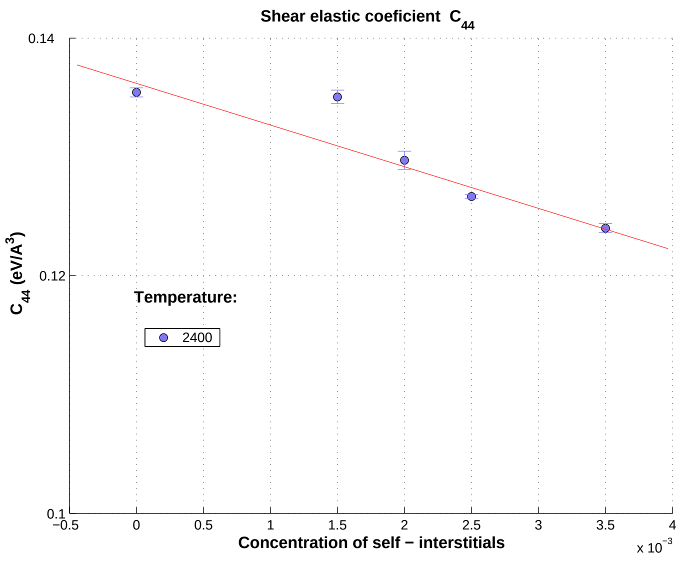


Figure 4.25: Dependence of C_{44} on the self-concentration of interstitials.

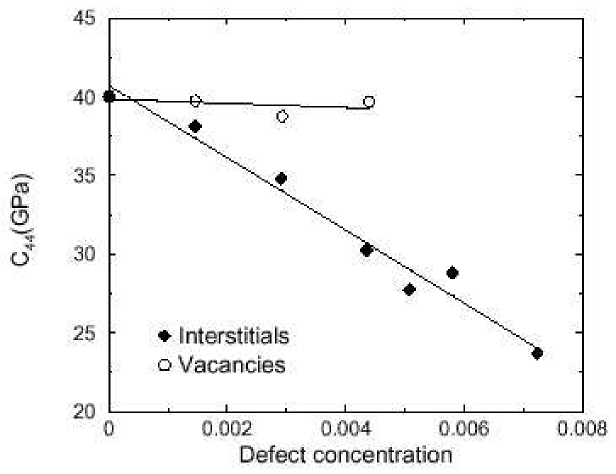


Figure 4.26: Dependence of C_{44} on the concentration of defects at T=1400K, obtained by A. Kanigel et al. [67].

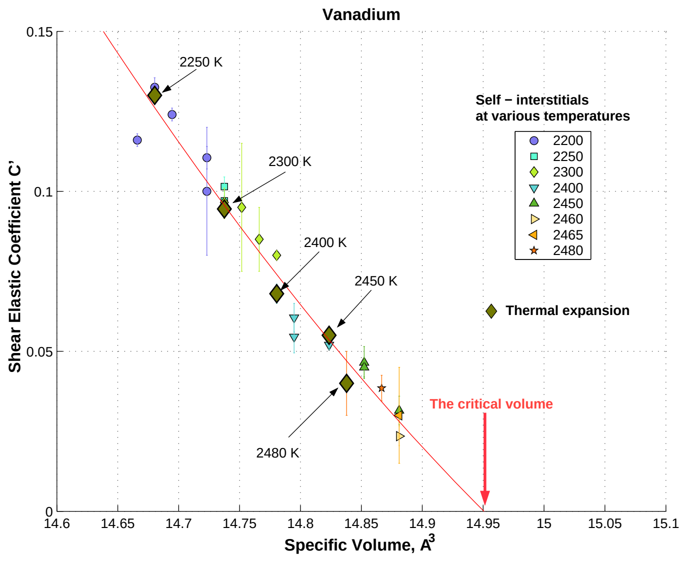


Figure 4.27: Variation of the elastic modulus C' with specific volume. Solid line is a quadratic extrapolation.

volume up to the point at which $C'(T_b) = 0$. The values of the critical specific volume are close to the specific volume of liquid vanadium at the thermodynamic melting temperature [74] is $v_{liq} = 15.431\ \dot{A}^3$

Similar results were obtained for copper (fcc lattice) by A. Kanigel et al.[67,92](See Fig 4.28), and J. Wang et al.[15]. Using the molecular dynamics method, it was shown that the Born's melting criterion is valid for the mechanical melting which occurs when the free energy based heterogeneous melting starting from surface or grain boundaries is kinetically suppressed. It was predicted that the incipient instability $C' = 0$, occurring at the observed lattice strain $a/a_0 = 1.024$, where a lattice parameter at $T = 1350K$, and a_0 lattice parameter at $T = 0K$. Therefore the specific volume ratio of copper is $(a/a_0)^3 = 1.074$ which is very close to the value obtained for Va $v(T_m)/v(T_0) = 1.075$. This result hints that the ratio a/a_0 could be a univesal and independent of lattice structure, but to check that assumption it is necessary to investigate the bulk melting transion of other bcc crystals.

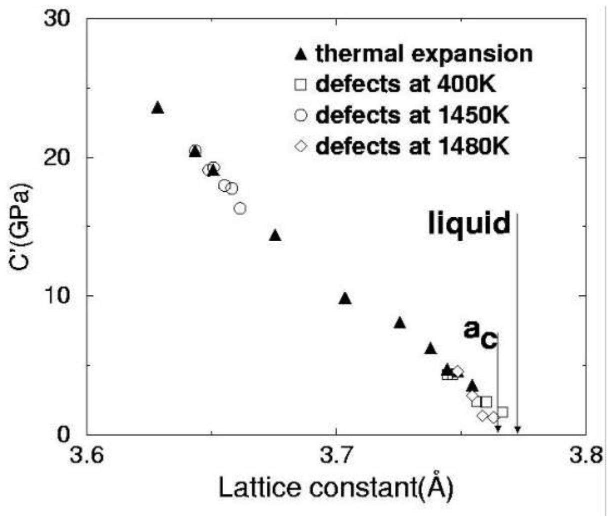


Figure 4.28: Dependence of C' on the lattice constant for bulk Cu, showing the equivalent effect of addition of point defects at constant T and heating without point defects, from ref.[67,92].

4.5 Influence of vacancies on the shear moduli

The significant influence of self - interstitials on the shear elastic moduli raise the question of whether vacancies could have a similar effect. It is known that if the lattice is distorted in the neighborhood of a vacancy some new vibrational modes of atoms are created. The effect could lead to softening of the solid. Yet, according to our simulations the shear elastic coefficients are less affected by the vacancies in comparison with self - interstitials, as it is shown in Figs. 4.29-4.30. It is interesting to notice that at high vacancy concentration of $20/2000 = 1\%$ we observe the noticeable reduction of the shear elastic moduli of vanadium, in contrast to the results obtained for copper by A. Kanigel [67] (See Fig. 4.26). However, at these concentrations interactions between the vacancies are not negligible, and that complicates the physics of the phenomenon.

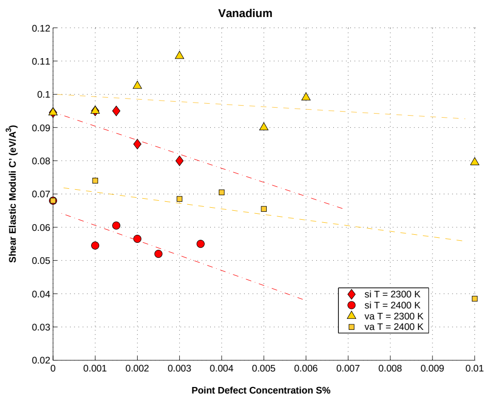


Figure 4.29: Dependence of C' on the concentration of point defects at T = 2300 K and T = 2400 K , showing the difference between influence of vacancies and interstitials. The dashed lines to guide the eye.

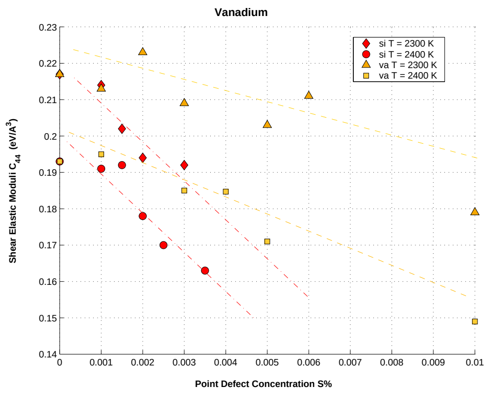


Figure 4.30: Dependence of C_{44} on the concentration of point defects at T = 2300 K and T = 2400 K , showing the difference between influence of vacancies and interstitials. The dashed lines to guide the eye.

Chapter 5

Results: surface melting

5.1 Surface melting: the goal of research

As is stated above the bulk melting transition is preempted by the surface melting transition and therefore can not be observed experimentally (at standard conditions). The aim of this part of our research was to investigate the phenomenon of surface disordering and premelting.

The questions that are addressed in our computer simulations are following: What is the thermodynamic melting point at which the solid starts to melt from the surface in our model? How the structural, transport and energetic properties of the solid change at the surface region as a function of temperature? At which temperature surface disordering and premelting begins and an adlayer appears on the top of the surface layers ? Whether surface premelting of various low-index bcc faces of vanadium is different (anisotropy)? If the phenomenon is anisotropic, whether the anisotropy "follows" the packing structure of the vanadium faces (e.g. one could expect the least packed surface (111) begins disorder first, while the close packed surface (011) may preserve its crystalline structure up to T_m)? Of course, this is not a complete list of questions which might be asked, the list can be easily continued furthermore. However, the main question, to which we want provide the answer, was whether the Born scenario for the bulk melting transition (e.g. the concept of a critical volume) can be applied to explain the surface melting? In the following section we will try to answer these questions.

5.2 Initial configuration

In our simulations a semi-infinite system is modeled via a thick slab: 3 fixed layers at the bottom (to mimic the effect of the presence of the infinite bulk of the crystal), and on top of these 3 layers there are 24 layers free to move (See Fig.5.1). Due to the relatively short range of the repulsive part of the modified **FS** potential, the 3 fixed layers are sufficient to represent the static substrate. We use a slab with [10x10] geometry (100 atoms in a layer), for samples with Va(001) and Va(111) faces (see Fig. 5.2 and Fig. 5.3), and a slab with [7x7] geometry (98 atoms in a layer) for the Va(011) sample as shown in Fig. 5.4. A free boundary at the top (z axis is perpendicular to the free surface) and periodic boundaries at the sides along the x and y axes are implemented.

In order to construct samples with the different low-index faces one can use primitive unit cell vectors. Alternatively, it is possible to construct a large Va(001) sample, rotate the axes and cut a subsample with the desired low-index face geometry (See Fig 5.5 and Appendix E.) The geometric properties of the different samples are summarized in Table 5.1. The lattice parameter a_0 is calculated for various temperatures at zero external pressure by means of Parinello-Rahman method. In each simulations of the surface premelting a_0 is rescaled with temperature by interpolating the data obtained in the bulk simulations.

Surface	a_x	a_y	a_z	ρ_s
Va(001)	a_0	a_0	$a_0/2$	$1/a_0^2$
Va(011)	$\sqrt{2}a_0$	a_0	$a_0/\sqrt{2}$	$\sqrt{2}/a_0^2$
Va(111)	$\sqrt{2}a_0$	$\sqrt{3/2a_0}$	$a_0/(2\sqrt{3})$	$1/\sqrt{3}a_0^2$

Table 5.1: A list structure parameters of different low-index faces of a bcc crystal; $a_x,\ a_y,\ a_z$ are the nearest - neighbor distances in the $x,\ y$ and z direction, correspondingly, (indicated in units of the lattice parameter a_0), and ρ_s is in-plane surface density

5.3 Equilibration and calculation

In order to reach an equilibrium at various temperatures T

(See Fig. 5.6) a sample is equilibrated for $N_{eq} = 10,000$ integration time steps (with time step $dt = 1.02 \times 10^{-15} sec$). Thereafter the various structural and transport properties of the system were calculated, accumulated and averaged over a long period of $N_{meas} \simeq 6,000,000$ MD steps. The trajectories of the atoms of the sample are produced for data analysis by using the canonical ensemble (**NVT**).

Various physical properties are monitored continuously during the simulation. An equilibrium state is considered to be achieved when there are no significant temporal variations (beyond the statistical fluctuations) in the total energy, layer occupation number, structure order parameters, and diffusion coefficients are seen in any of surface layer. A uniform profile of kinetic temperature across the sample is observed at the equilibrium for each temperature (See Fig. 5.7). The statistical averages have been calculated typically over 3,000 configurations, separated by 1000 time steps

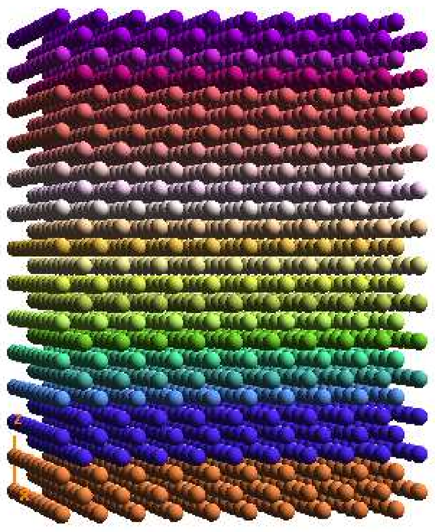


Figure 5.1: An initial configuration (Va(001) sample) : different colors are given to the atoms of the different layers to visualize their mixing in the course of surface premelting.

Figure 5.2: Geometry of the $Va(001)$ surface. Top view.

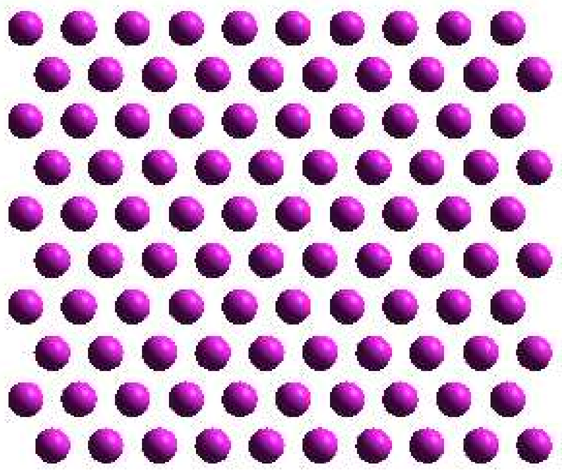


Figure 5.3: Geometry of the least packed $Va(111)$ surface. Top view.

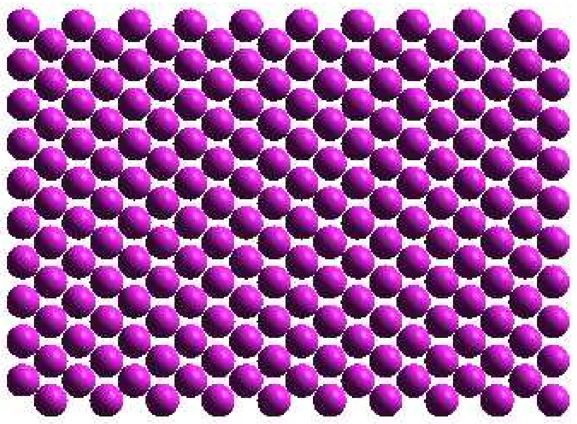


Figure 5.4: Geometry of the close packed $Va(011)$ surface. Top view.

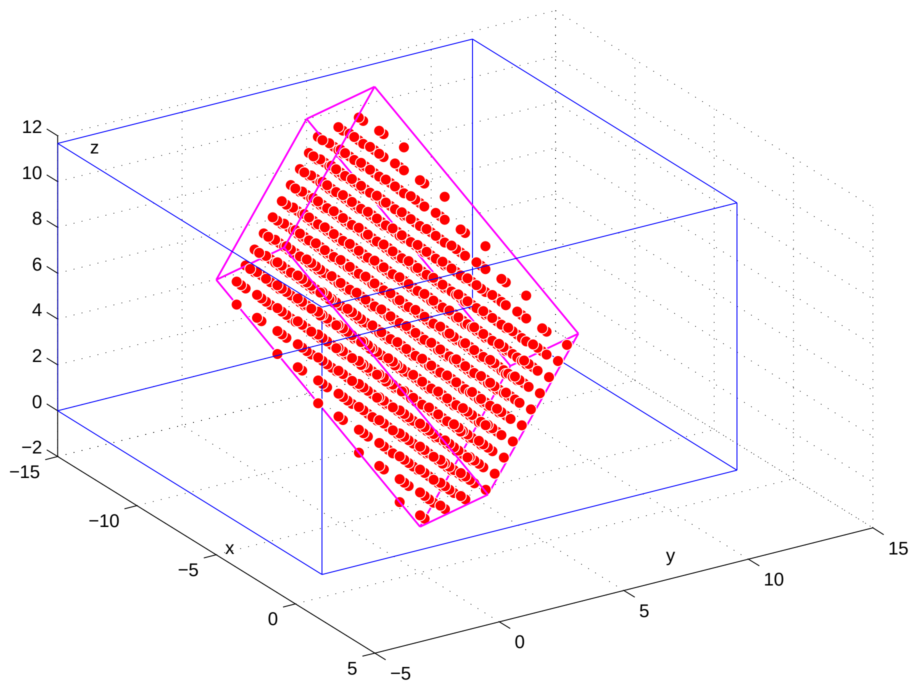


Figure 5.5: Construction of the $Va(111)$ sample, see Appendix B for details.

which is concluded, relying on analysis of behavior of the time correlation function of fluctuations in the various variables of the system (i.e. temperature, pressure, energy, velocity, etc), to be sufficient to make the collected configurations statistically independent.

5.4 Thermodynamic melting point

In order to study the phenomenon of premelting, we have to find the thermodynamic melting point T_m of our model. At least three different methods for estimation of the melting point have been given in the literature. The most elegant and thermodynamically valid is based on the calculation of the Gibbs free energies of the bulk solid and liquid phases as a function of temperature at given pressure. The melting point

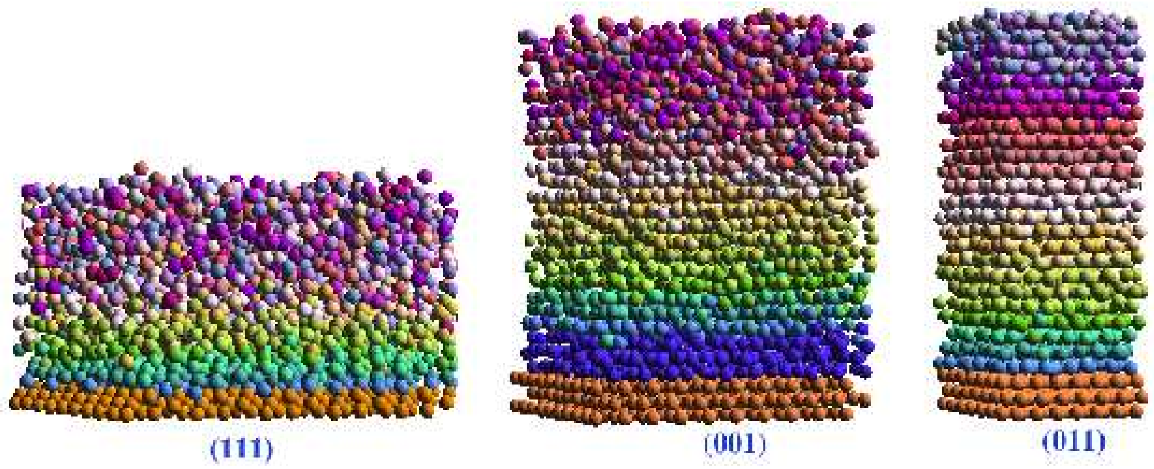


Figure 5.6: Snapshots of $Va(001)$,$Va(011)$, and $Va(111)$ at 2200 K.

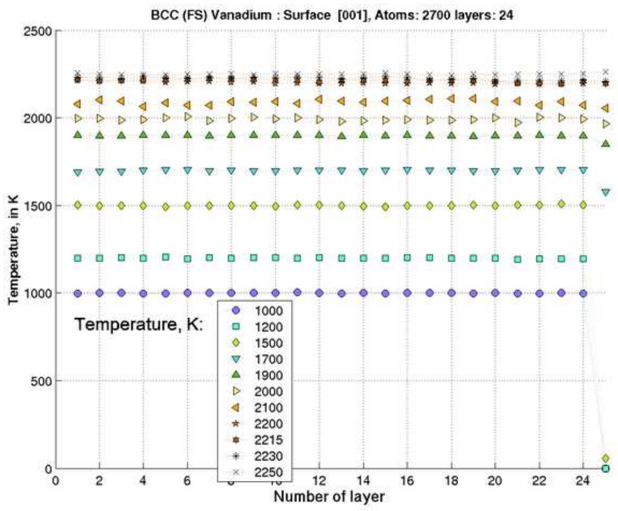


Figure 5.7: Temperature profiles across the $Va(001)$ sample in the equilibrium state (simulations at different temperatures).

is then found, by definition, by making the Gibbs free energies of solid and liquid equal [19]. Hoverer, this method is computationally quite elaborate and contains a fairly large source of inaccuracy, due to the fact that the Gibbs free energies usually depend quite weakly on temperature, and T_m is then an intersection point of two flat curves. Therefore, it is quite difficult to find the position of the intersection point precisely and a small error in the obtained position of the point leads to a large error in estimation of T_m.

In the second method, proposed by J.F. Lutchko et al. [77] one determines the temperature dependent velocity of the moving solid-to-liquid interface $v_{sl}(T)$ at $T > T_m$ and thus, by extrapolating this velocity to the zero $v(T_m) = 0$ one can obtain the solid-liquid coexistence point, which is interpreted as T_m. This method has been shown to give a melting point comparable to what is obtained from the free energy method.

The third method, is computationally the most straightforward one. In this method, an excess amount of kinetic energy is supplied to the atoms of the surface region of the sample. This results in a temperature gradient and in a nucleation of liquid phase at the surface. Heating is then stopped and the sample is allowed to evolve back to equilibrium. The temperature at which a solid-liquid coexistence takes place is interpreted as the thermodynamic melting point T_m [89]. All these methods can estimate T_m with comparable accuracy. However, the second method is easier for implementation and less CPU time consuming than the first and the third methods, at the same level of accuracy.

We chose the second method and determined the velocity of the solid-melt interface at different temperatures (in the range of 2300 - 2500 K) in the samples with various low-index free surfaces. The value of T_m varies with the geometry of the simulation cell, and especially with the number of the surface layers. To test such effects we performed simulations for the samples with the different number of layers, N_l, using 64 atoms in a layer (Va(111), Va(001)) and 72 atoms (Va(011)).

A temperature above T_m (the upper limit of T_m was found in the bulk simulations) was reached using the following method. We started the simulation at a low temperature (usually at $T = 1800\ K$) far below T_m, and increased the temperature at a high rate (by $\Delta T = 50\ K$) after each 1000 steps. In this way the system could be superheated up to $T = 2500\ K$. A very simple method of the velocity rescaling in each MD step is used, e.g. the instantaneous kinetic energy is calculated as $E_{kin} = \sum_i mv_i^2/2$ and velocity of each atom is rescaled by the same factor to satisfy $E_{kin} = 3/2k_BT$. This simple method of the velocity rescaling was chosen instead of the Nose-Hoover mechanism to significantly reduce the fluctuations of temperature. Another alternative to feedback control is the of mechanical constraints [54], which enforce constant temperature very effectively, and therefore could be applied in calculation of T_m. When the desired temperature is reached, the velocity of the solid-melt interface is determined by monitoring the decrease of the structure order parameters, η, as a function of time (See Fig. 5.8). The order parameter was calculated only over the top 10 surface layers to avoid the ordering effects introduced by the static substrate. The temperature dependence of the velocity v_{sl} is governed by the thermodynamic free-energy driving force, given by the free energy difference, $\Delta G = G_s - G_{liq}$, between the solid G_s and the liquid G_{liq} phases [90,91].

$$v_{sl} \sim \frac{D_{liq} d_L}{\Lambda^2} \left(1 - exp\left(\frac{\Delta G}{RT}\right)\right) \tag{5.1}$$

where Λ is the mean free path in the liquid phase, d_L is the thickness of the liquid film, D_{liq} is the self-diffusion coefficient of atoms in the liquid phase. One can write

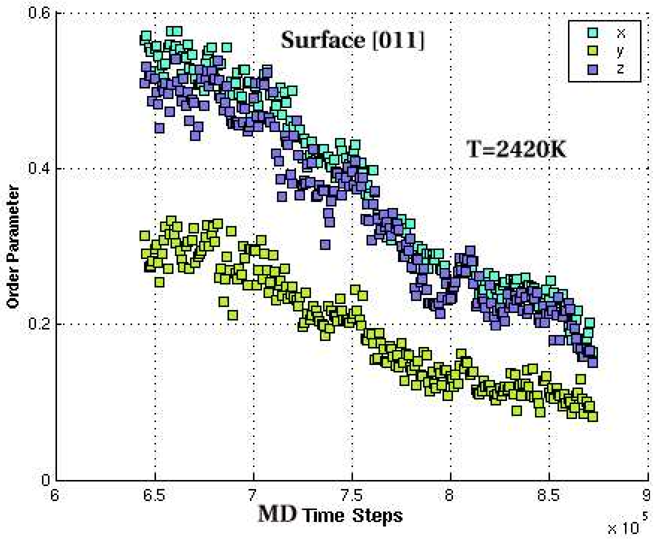


Figure 5.8: Order parameters of the superheated $Va(011)$ system vs. time (MD steps) at temperature $2420K$. The sample contains 2520 atoms.

with a good approximation [90] $\Delta G \approx (T - T_m)\Delta S$, where ΔS is entropy difference between the solid and liquid phases, and obtain at temperatures close to the melting point

$$v_{sl} \sim \frac{D_{liq} d_L}{\Lambda^2}\left(1 - exp\left(\frac{(T - T_m)\Delta S}{RT}\right)\right) \approx const \times (T - T_m) \tag{5.2}$$

Using this expression for $v_{sl}(T)$ as a function of temperature, T, we calculate the thermodynamical melting point by a linear fit (See Fig 5.9). The temperature at which the propagation velocity vanishes is considered to be the thermodynamical melting point T_m [89]. The melting temperatures obtained for the samples with different number of surface layers and the various types of low-index faces are summarized in Table 5.2. The smaller the number of the layers, the larger the apparent T_m. This is due to the influence of the underlying static substrate, which stabilizes and orders the samples. Therefore, to obtain more accurate and reliable estimation for the melting point T_m we use the sample with the largest number of free surface layers.

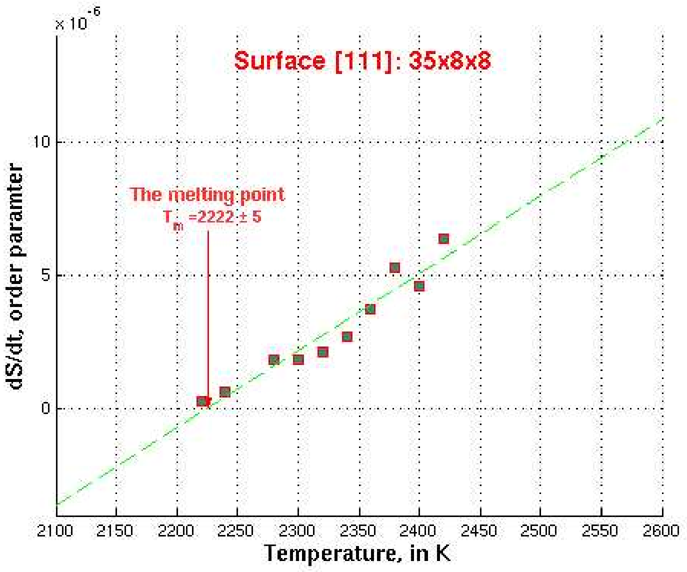


Figure 5.9: Velocity of propagation of the solid-to-liquid interface, v_{sl}, as a function of temperature and extrapolation to zero velocity. The $Va(111)$ sample, 2240 atoms, 35 layers.

The calculated melting temperature of the sample with the least packed face Va(111) is $T_m = 2222 \pm 10K$ is the close one to the experimental value $T_m = 2183K$. For other surfaces T_m is insignificantly larger due to superheating effects on the solid-liquid interface just above the thermodynamical melting point T_m. This effects are

N_l	T_m $Va(111)$	T_m $Va(001)$	T_m $Va(011)$
23	$2241 \pm 10K$	$2230 \pm 10K$	$2260 \pm 10K$
26	$2234 \pm 10K$	$2200 \pm 12K$	$2252 \pm 10K$
29	$2220 \pm 8\ K$	$2235 \pm 11K$	$2260 \pm 12K$
32	$2224 \pm 10K$	$2237 \pm 8\ K$	$2270 \pm 11K$
35	$2222 \pm 9\ K$	$2228 \pm 10K$	$2240 \pm 6\ K$

Table 5.2: The melting temperatures for the various surfaces of vanadium, calculated for the samples with different number of the surface layers, N_l.

very significant for the close packed Va(011) surface. This dependence of the thermodynamic melting point on the geometry of the crystal face is also observed in MD simulations when temperature is increased gradually up to the melting point. It was found the Va(111) and Va(001) surfaces melt approximately at the same temperature $T_m \approx 2230 \pm 30\ K$, while the sample Va(111), with the close packed free surface, melts at more elevated temperature about $T_m \approx 2245 \pm 10\ K$.

We want to point out again that in the bulk simulations, where we employed periodic boundary conditions in all directions, melting is observed to happen at a temperature $T_b = 2500 \pm 5K$. This melting temperature is not the true thermodynamic melting point T_m, but a temperature at which mechanical instability arises, leading ultimately to a mechanical collapse of the crystal lattice. The temperature at which mechanical instability occurs can be determined by using the Born criterion at zero-external stress. This shows that the superheating region, $T_b - T_m$, in MD simulations of metals may be large enough (250 K in case of vanadium). Therefore, it is very important to distinguish between the thermodynamic melting tempertaure T_m and the temperature T_b above which the lattice become mechanically unstable.

5.5 Local density profile

A local density profile $\rho(z)$ is defined as the average number of atoms in a slice of width Δz, which is parallel to the solid substrate. The proper choice of the width Δz of a slice is a trade-off between two factors. First, a very small width results in too few particles in each slice, and therefore one observes large statistical errors and data scattering. Second, a very large width of a slice will not show the actual dependence of the properties on the distance from the surface. Hence a balance between those two requirements must be achieved. To facilitate the presentation we use $\rho(z)$ represented by a continuous function (See Fig. 5.10) defined according to Chen et al.[79] :

$$\rho(z) = \left\langle \frac{1}{\sqrt{2\pi\Delta z}} \sum_i exp(-\frac{(z-z_i)^2}{2\Delta z}) \right\rangle \tag{5.3}$$

where z_i is z coordinate of atom i, with $z = 0$ set at the bottom of the non-fixed slab, and the angular brackets indicate time average. We use $\Delta z = 0.1a_0/2\sqrt{3}$, where a_0 is the lattice parameter. The premelting of the crystal surface exhibits itself

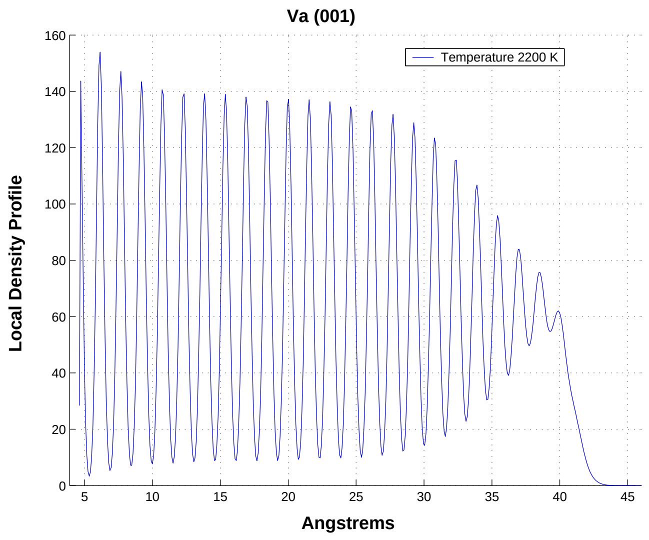


Figure 5.10: Density profile of $Va(001)$ at T=2200 K

in the loss of long-range order. This transition can be examined by monitoring the layer-by-layer modulation of the density profile of the system at various temperatures up to the melting point T_m. Sets of the plots of the local density profiles for the samples Va(111),Va(001), and Va(011) are shown in Figs. 5.11-5.13, respectively. At low temperatures the density profile $\rho(z)$ consists of a series of sharp, well resolved peaks. The atoms are packed in the layers with constant density in each layer and virtually no atoms in between these parallel layers. As the temperature is increased the effective width of the each layer becomes broader due to the enhanced atomic vibration, and the position of the peaks move to larger values of z due to thermal expansion.

At the temperatures close to the melting point the atomic vibration becomes so large, especially in the first layer, that disorder sets in, with atomic migration taking place between the layers, as evidenced by the fact that the minima of $\rho(z)$ between two peaks rise to non-zero values. This is a reminiscent of a liquid-like structure (which is also observed in the plane radial distribution function). Hence, the system crosses over to a state of "premelting" at elevated temperatures. At some temperature ($T^* \simeq$ 1000 K for the Va(111), $T^* \simeq$ 1500 K for the Va(001), $T^* \simeq$ 2200 K for the Va(011)) the density of the topmost layer becomes slightly lower than that of the underlying layers. The loss of density is compensated by appearance of atoms on top of the first surface layer. These atoms are called *adatoms* and the additional surface layer is termed as an *adlayer.* As the temperature is elevated atoms from the deeper subsurface layers start to diffuse toward the adlayer. The distinction between the layers becomes blurred. The generation of adatom-vacancy pairs induces disorder and converts the topmost layers into a thin quasiliquid film.

A general conclusion is made concerning the formation of an adlayer relying on analysis of the temperature dependence of the density profiles; the formation begins first on the least packed surface Va(111), thereafter at higher temperature a quasiliquid film appears on the Va(001) face, which has an intermediate density between the Va(111) and Va(011) faces. Finally, surface premelting occurs on the close packed face Va(011) at a temperature which is very close to the melting point T_m, and practically all adatoms come from the first surface layer.

These results are in good agreement with the of investigation of surface premelting

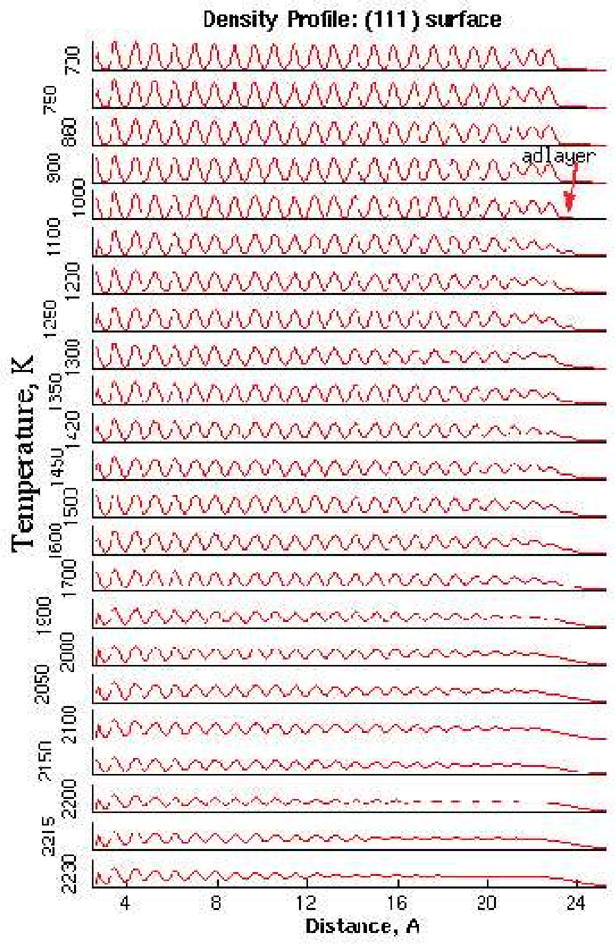


Figure 5.11: Density profile across $Va(111)$ along the z direction perpendicular to the surface at various temperatures as indicated.

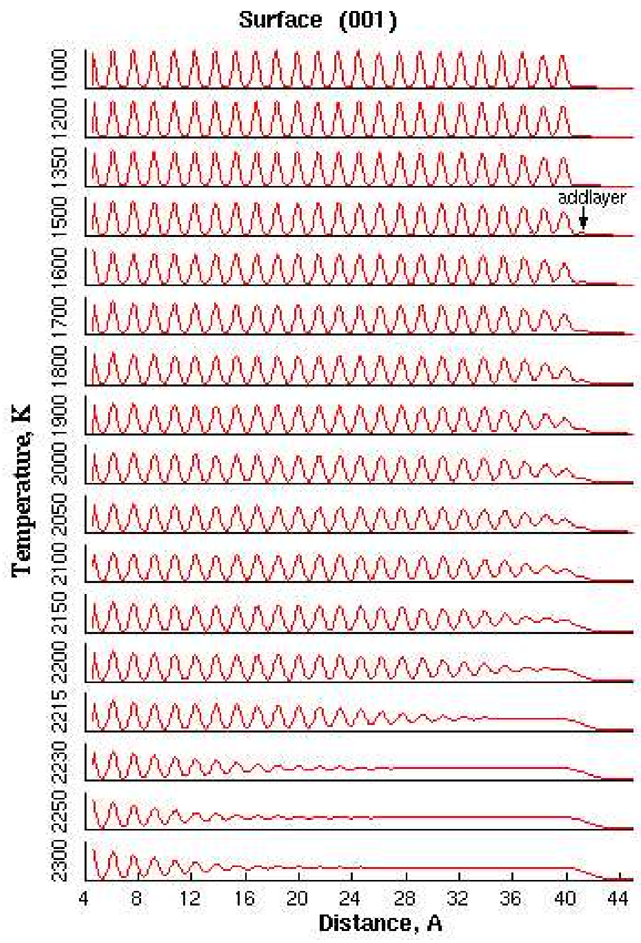


Figure 5.12: Density profile across $Va(001)$ along the z direction perpendicular to the surface at various temperatures, as indicated.

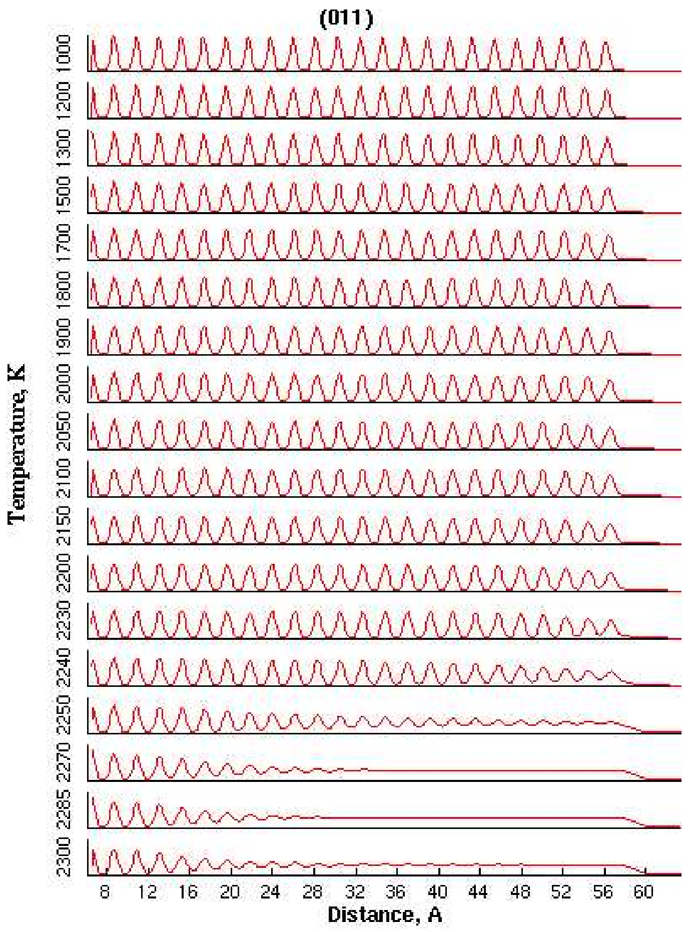


Figure 5.13: Density profile across $Va(011)$ along the z direction perpendicular to the surface at various temperatures, as indicated.

of fcc metals Al [78], Ni [79], and Cu [80]. It was found that an adlayer appears first on the least packed (011) surface, then at higher temperature on the (001) face, and finally the onset of disorder is observed on the close packed (111) surface at a temperature close the melting temperature.

5.6 Structure order parameters

Structure order parameters (structure factors) are useful for monitoring the order-disorder transition in the course of surface premelting. The structure order parameter is also related to low energy electron diffraction (**LEED**) intensity [97], which can be measured experimentally. Atomic vibrations break to some extent the periodicity of lattice and diffraction effects provide essentially direct information about the vibration amplitude. The structure factor is defined as a Fourier transformation of the atomic density of the system.

$$\eta_{\vec{q}} = \left\langle \frac{1}{N^2} \left| \sum_j exp(i\vec{q}\vec{r}_j) \right|^2 \right\rangle \tag{5.4}$$

where N is the number of atoms, and the vector $\vec{r}_j$ describes the position of the atom j, while the vector $\vec{q}$ is related to elastic moment transfer (diffractive scattering).

In the case of a surface, the order parameter is often defined for each layer separately:

$$\eta_{l,\alpha} = \left\langle \frac{1}{n_l^2} \left| \sum_{j \in l} exp(i\vec{g}_\alpha\vec{r}_j) \right|^2 \right\rangle \tag{5.5}$$

where $\alpha = 1, 2, 3 \equiv x, y, z$ are indices of the Cartesian axis x, y, z and $\vec{g}_1, \vec{g}_2, \vec{g}_3 = \frac{2\pi}{a_x}\hat{x}, \frac{2\pi}{a_y}\hat{y}, \frac{2\pi}{a_z}\hat{z}$, is a set of vectors which define a set of different directions (the order parameter is calculated along those directions), a_α is the nearest-neighbor distance in α direction (See table 5.1), n_l is the instantaneous number of atoms in the layer l, the sum extends over the particles in the layer l, and the angular brackets denote averaging over time.

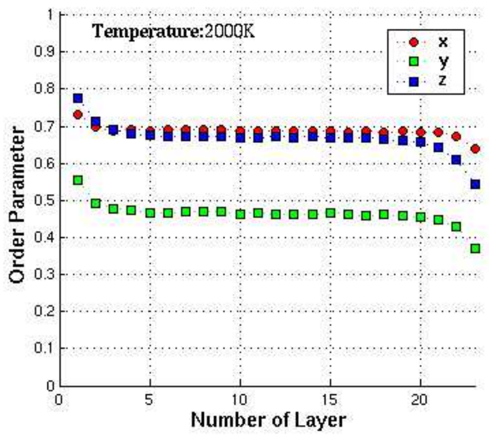


Figure 5.14: Order parameters of $Va(011)$ at 2000 K. The difference between $\eta_{l,x}$ and $\eta_{l,y}$ reflects the anisotropy of the Va(011) surface.

For an ordered crystalline surface the order parameter is a unity at zero temperature. The deviation of the $\eta_{l,\alpha}$ from the unity originates from thermal vibrations and from formation of surface defect. The structure order parameters of the Va(001) sample at $T = 2000\ K$ is shown in Fig. 5.14. Note the decrease of the order parameter in the surface region, which reflects enhanced atomic vibrations and adatom-vacancy pair creation. The existence of vacancies does not directly affects the order parameter, since a normalization procedure is employed during each measurement by using the instantaneous layer occupation n_l of a layer. Nevertheless, vacancies have an indirect effect on the order parameter by introducing a lattice distortion around them.

As is evident from the Figs. 5.15 the structure order parameter of the Va(011) sample is lower along the y-direction than along the x-direction. The same effect is observed for the Va(111) sample, but is absent for the Va(001) sample. This anisotropy $\eta_{l,y} < \eta_{l,x}$ actually arises from the anisotropic structure of the low-index

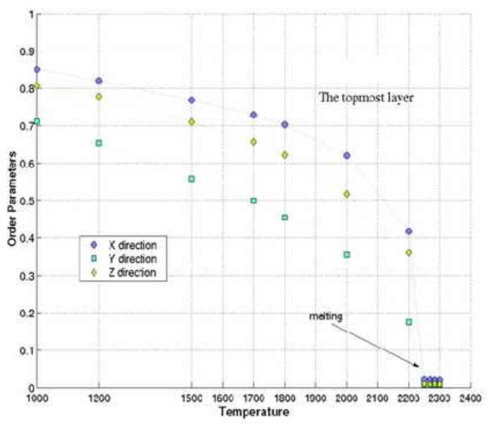


Figure 5.15: Comparison of order parameter of the layers of $Va(011)$ as a function of temperature.

faces Va(011) and Va(111), where the distances between the nearest-neighbors are different in the x and y directions $a_x > a_y$. Assuming, in the first approximation, that each atom oscillates with the same amplitude in both the x and y directions,i.e. $< u_x^2 > \simeq < u_y^2 >$, and expanding the structure order parameter in term of $< u^2 > /a_\alpha^2$ (it is found in our MD simulations that mean square amplitude of vibration is order of $< u^2 > \simeq 10^{-2}\dot{A}^2$ while the lattice parameter squared is about $a_0^2 \simeq 3.05^2\dot{A}^2$) we obtain:

$$\eta_{l,\alpha} \simeq 1 - \sum_{j \in l} \frac{4\pi^2 \langle u^2 \rangle}{n_l^2 a_\alpha^2} + ... \tag{5.6}$$

Hence it follows that if $a_x > a_y$ then $\eta_x > \eta_y$, because the smaller $< u^2 > /a_\alpha^2$, the less the decrease of the order parameter. The same consideration can be applied to explain the difference between in-plane components of the order parameter (x and y directions) and the out-of-plane component (z-direction), but one has to take into account that mean square vibrational amplitude in the plane direction is larger than in the out-of-plane ones.

The structure order parameter profiles at various temperatures are plotted in Figs.

5.16-5.18. Note a continuous decrease of the order parameter for the Va(111) sample, which begins to premelt first. In contrast, one can see a relatively abrupt decrease of the order parameter of the close packed Va(011) sample, which takes place only in vicinity of the melting transition.

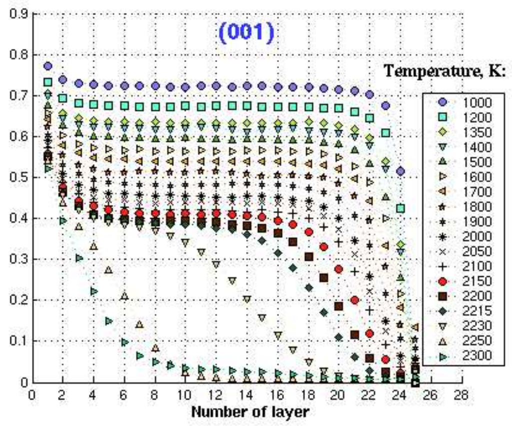


Figure 5.16: x component of the order parameter of $Va(001)$ at various temperatures.

5.7 Plane radial distribution function

The structure of the system, and in particular the formation of a quasiliquid film can be analyzed by using a plane radial distribution function defined as

$$p_l(r_{||}) = \left\langle \frac{1}{n_l} \sum_{i,j \in l} \frac{\delta(r_{ij,||} - r_{||})}{2\pi r_{||}} \right\rangle \tag{5.7}$$

where $r_{ij,||}$ is the component of the $\vec{r}_i - \vec{r}_j$ parallel to the surface plane, n_l is the instantaneous number of atoms in the layer l, the sum extends over all particles in the layer l, and the angular brackets denote averaging over time (See Fig. 5.19) . In practice, the equilibrium $p_l(r_{||})$ function calculated using a histogram method [54,82],

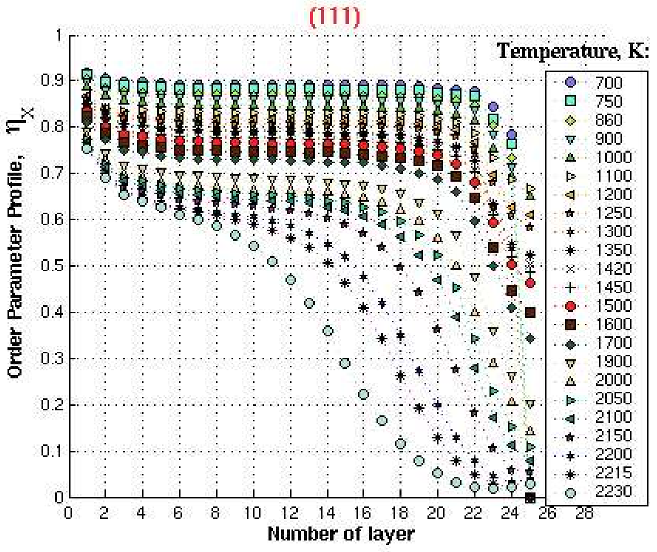


Figure 5.17: x component of the order parameter of $Va(111)$ at various temperatures.

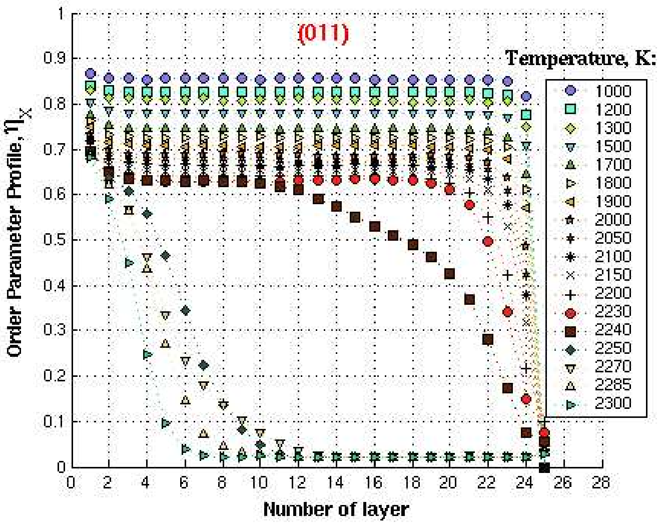


Figure 5.18: x component of the order parameter of $Va(011)$ at various temperatures.

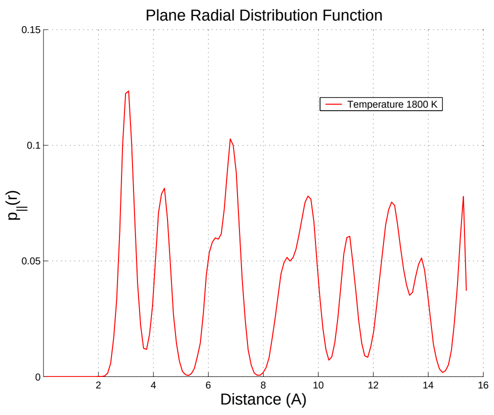


Figure 5.19: Radial distribution function $p(r_{||})$ of the 6th surface layers of Va(001) system at 1800 K.

based on counting and binning of the atom pair separations. Let $h_k(r, \Delta r)$ to be the number of the atom pairs (i, j) in a kth bin (or shell):

$$(k-1)\Delta r \leq r_{ij,||} \leq k\Delta r \tag{5.8}$$

then the plane radial distribution function (**RDF**) is obtained from:

$$p_l(r_{||}) = \left\langle \frac{A}{n_l} \frac{h_k(r, \Delta r)}{2\pi r_{||} \Delta r} \right\rangle \tag{5.9}$$

where A is the area of a layer, n_l is the instantaneous number of atoms in layer l. The two-dimensional radial distribution function $p(r_{||})$ for the top surface layers of Va(001) at different temperatures are shown in Figs. 5.20. As seen from the figure the intra-layer structure in these layers changes gradually from crystalline to liquid-like as the temperature increased. Particularly noticeable is disappearance of the crystalline features in $p(r_{||})$ corresponding to the second, third and other nearest neighbors. In addition to the heights of the peaks, the area under the $p(r_{||})$ curve changes, which reflects the change in the density across the solid-liquid interface.

We note that for the adlayer at $T < T_m$ the probability of finding particles with

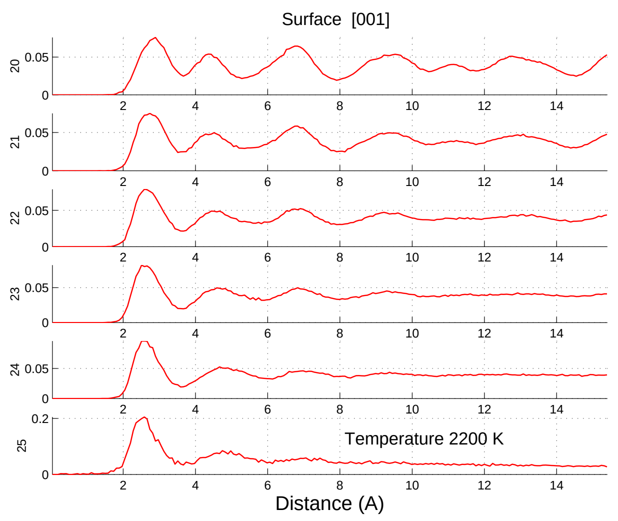


Figure 5.20: Two-dimensional radial distribution function $p(r_{||})$ of the five top layers of Va(001) at temperature T=2200K. The layer n=25 corresponds to the adlayer, n=24 corresponds to the (first) surface layer, n=23 corresponds to the second one.

separation beyond the first-neighbor shell is relatively small, indicating a tendency for clustering which persists, though to a smaller extent, even to $T \simeq T_m$.

The plane pair correlation functions of the first surface layer at elevated temperatures for the various low-index faces is shown in Fig. 5.21. It is clear that the crystalline order vanishes gradually and the quasiliquid film thickness increases when the melting point T_m is approached. We can conclude relying on the analysis of the $2D$ radial distribution functions of the various faces, that surface premelting begins first on the least packed Va(111) surface at temperature around 2000 ± 50 K, e.g. 200 K below the estimated thermodynamical melting point, while at the face Va(001) liquid phase starts nucleate at around 2050$\pm$50 K, and on the most close packed Va(011) surface noticeable changes in the $p(r_{||})$ functions occur at temperature 2150 ± 50 K, which is close to the melting point.

Similar results were obtained in studies of surface premelting of low-index faces

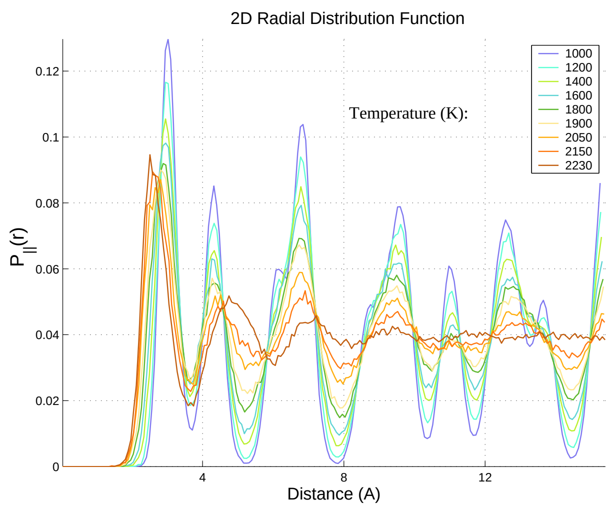


Figure 5.21: Two-dimensional radial distribution function $p(r_{||})$ of the surface layer of $Va(001)$ at various temperatures.

of fcc metals in computer experiments by Hӓkinen et al. [83] (Cu), by Chen et al [79] (Ni) and by Carnevali et. al [84] (Au), as well as in some real experiments [23,26,85]. The close packed face (the (111) face of a fcc lattice) preserves its crystalline order up to the melting point. This non-melting behavior of (111) is in striking disagreement with theoretical predictions and results of computer simulations based on a simple type of Lennard-Jones potentials, which are a crude approximation for fcc metals. More sophisticated MD simulations, which use many-body potentials do not confirm the theoretically predicted very pronounced premelting effects for the close packed (111) surface below the triple point [86-88].

5.8 Diffusion coefficients

Diffusion coefficients of the surface layers are calculated to investigate transport properties of the Va(111),Va(001) and Va(011) samples. The coefficients are found from the particle trajectories, $\vec{r}_{i,\mu}(t)$, by calculating the average mean square displacement $R^2_{l,\mu}$ (See Fig 5.22):

$$R^2_{l,\mu} = \left\langle \frac{1}{n_l} \sum_{i \in l} [\vec{r}_{i,\mu}(t+\tau) - \vec{r}_{i,\mu}(\tau)]^2 \right\rangle \tag{5.10}$$

where $\mu = x, y, z$ is a coordinate index, the sum includes atoms in the layer l, and the angular brackets denote averaging over time from the origin (τ). The diffusion coefficients $D_{l,\mu}$ are calculated separately in the x, y and z directions, according to Einstein relation for each layer:

$$D_{l,\mu} = \lim_{t \to \infty} \frac{R^2_{l,\mu}}{2t} \tag{5.11}$$

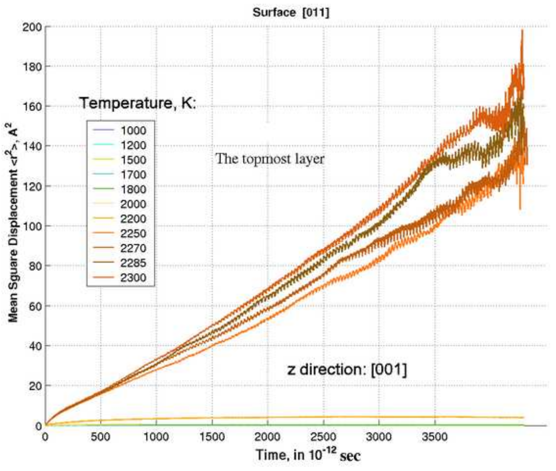


Figure 5.22: Mean square displacement $R^2_{l,z}$ in the z direction of an atom in the surface layer of Va(111) vs. time. Note the increased mobility of particles with temperature.

The diffusion coefficients are larger at the surface region.(See 5.23) The mobility of the atoms increases with elevation of the sample temperature and converges to the liquid bulk values. These observations correlate with the structural variations in the surface region exhibited in the pair correlation functions, the structural order

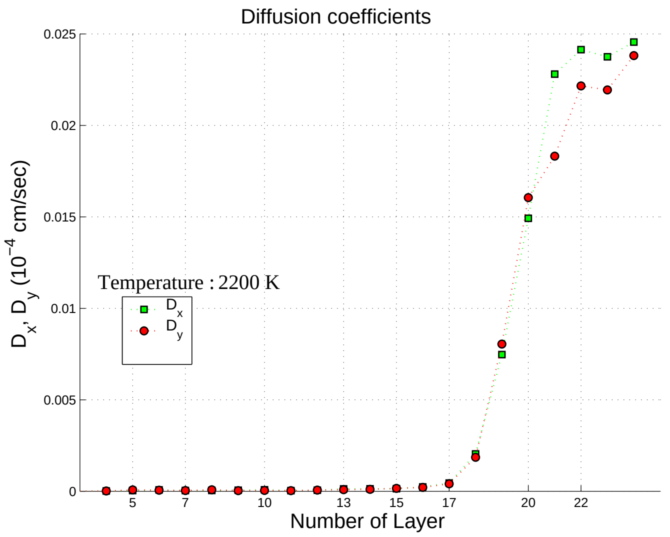


Figure 5.23: Diffusion coefficients of $Va(001)$ as a function of layer number at temperature T=2200 K.

parameters, and the local density profiles. The diffusion coefficients of the first crystal layer of the Va(011) as a function of temperature are shown in Figs. 5.24. The diffusion coefficients are different in different directions $D_x > D_y$ for Va(011) and Va(111). In the course of diffusion an atom jumps from one point (its current position) to another one (the nearest vacant place on the lattice), the distance between these two points is termed as *jump distance*. This jump distance is larger along the x-direction than in the y-direction, because the nearest-neighbor distance is larger in the x-direction than in the y, i.e. $a_x > a_y$ (See Table 5.1). Hence the diffusion coefficients are larger in the x-direction $D_x > D_y$. The diffusion coefficient along the z-direction is smaller than along the x and y directions. That difference can again be explained by the fact that $a_z < a_x,\ a_y$ (See Table 5.1), and therefore, the jump distance is the smallest along the z direction, thus $D_z < D_x,\ D_y$.

The diffusion coefficients of the least packed Va(111) surface are the largest one.

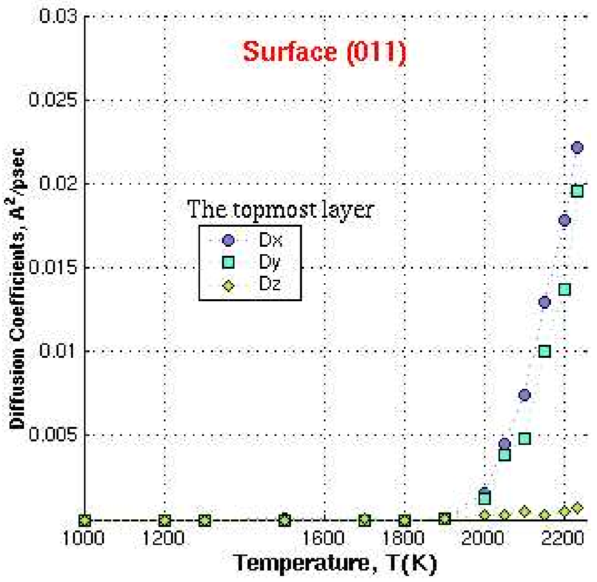


Figure 5.24: Diffusion coefficients of the surface layer of Va(011) vs. temperature in different directions. Note an anisotropy of the in-plane diffusion coefficients : $D_x > D_y$

The diffusion coefficients of the Va(001) are smaller than the Va(111) ones, yet they are larger than the diffusion coefficients of the close packed Va(011) surface, which are close to zero even at the elevated temperatures (See Fig 5.25).

5.9 Distance between layers

Structural information, like an interlayer relaxation and surface thermal expansion can be calculated directly from the difference between the average heights of the i and $i+1$ layers:

$$d_{i,i+1} = \left\langle \frac{1}{n_{i+1}} \sum_{j \in i+1} z_j - \frac{1}{n_i} \sum_{j \in i} z_j \right\rangle \tag{5.12}$$

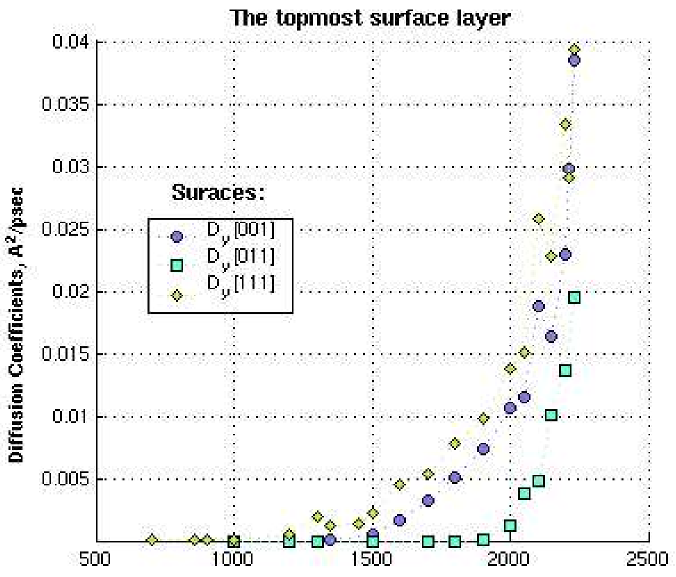


Figure 5.25: In plane diffusion coefficients as a function of temperature for the surface layer of Va(111),Va(001) and Va(011) calculated for the y direction).

where z_i is a z-coordinate of the atom i, the sum includes atoms in the layer i and $i+1$, and the angular brackets denote averaging over time. Distances between the neighboring layers could be calculated at temperatures up to the temperatures where surface premelting is sets in. Above such temperature the very concept of distinct layers becomes somehow doubtful, even though some structure is still visible in the local density profiles. At low temperatures the topmost surface layers exhibit an inward relaxation, i.e. $\Delta_{1,2} < 0$, where

$$\Delta_{1,2} = \frac{d_{1,2} - d_{bulk}}{d_{bulk}} \tag{5.13}$$

here $d_{1,2}$ is the distance in the z-direction between the first and the second surface layers, and d_{bulk} is distance between the two neighboring layers in the bulk, as it is shown in Figs. 5.26. One of explanation of the inward relaxation is following [83]; the

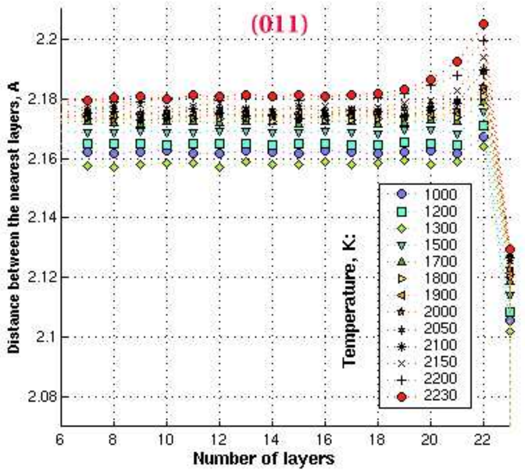


Figure 5.26: Distance between the neighboring layers vs. temperature of Va(011). Note the inward relaxation of the topmost surface layers.

surface suffers a “deficit” of the electron density, relative to the electron density in the bulk, and therefore it compensates the “deficit” by contracting the distance between the first and the second layers. The second and third layers exhibit a downward relaxation. The distance between neighboring layers increases with the temperature, and the thermal expansion of the surface layers is sufficiently larger than the thermal expansion of the layers close to the bulk. The observed “anomalous” thermal expansion of the surface layers is a direct manifestation of the broken symmetry of positional inversion at the crystal surface. Therefore, atoms rearrange their equilibrium positions in the surface layers and probe the more anharmonic region of the potential. The difference in the geometry of the faces is reflected in the thermal expansion at the surface region, a surface with the less packed layers expands more than a surface with close packed ones (See Fig. 5.27)

We also compare the averaged distances between the first and second surface

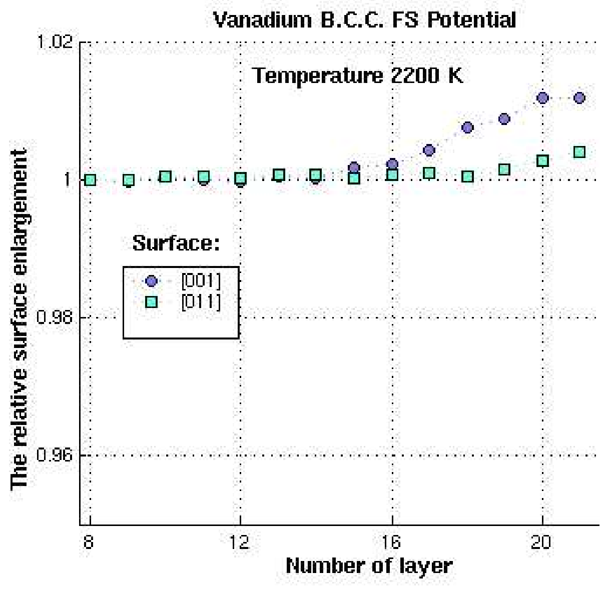


Figure 5.27: Interlayer distances, normalized to the distance between bulk neighboring layers at T=0 K, of $Va(001)$ and $Va(011)$ at T=2200 K.

layers of the Va(011),Va(111) and Va(001) samples (See Fig. 5.28). It is found that the Va(111) surface layers exhibit the largest thermal expansion in comparison with the other faces, and therefore the onset of "anomalous expansion" takes place at lower temperatures, than for the Va(011) and Va(001) systems. (Beyond the temperature $T = 1500K$ the first crystal layers of the Va(111) are molten and therefore the concept of distinct layers becomes meaningless) The larger thermal expansion of the Va(001) surface layer in comparison with the Va(011) surface layer can be followed to much higher temperatures.

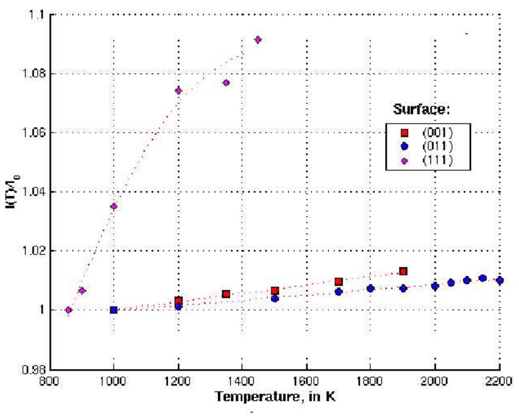


Figure 5.28: Interlayer distance between the first and second surface layers d_{12} as a function of temperature.

5.10 Layer occupation and energetics

The occupation of the topmost surface layers at various temperatures is calculated. The first surface layer is the most affected by disordering effects, i.e. adatom-vacancy pair formation.

Since the layers beneath the first are fully occupied at low temperatures, this leads to the formation of an adlayer on the top of the first layer (See Fig. 5.29). The adatoms leave behind an equivalent number of vacancies.

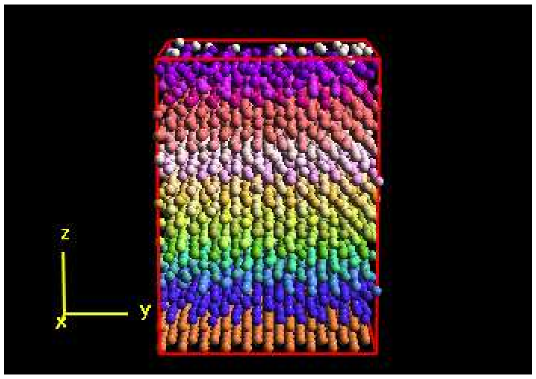


Figure 5.29: Snaphot of Va(001) at 1800K, notice the adatoms (white color) on the top of the surface layer.

The occupation of the surface layers at different temperatures for the various faces of vanadium are shown in Figs. 5.30-5.32. The formation of an adatom layer

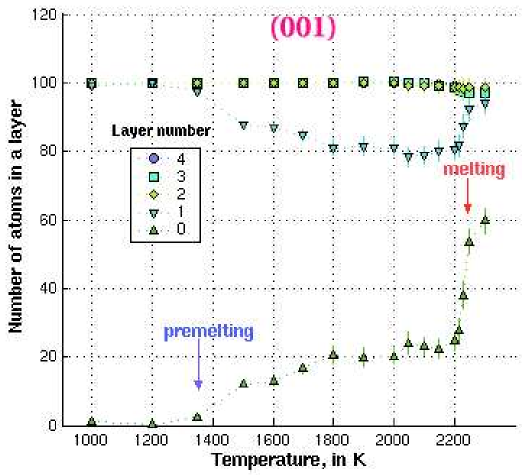


Figure 5.30: Layer occupation of the $Va(001)$ sample as a function of temperature for the adatom layer.

on the least packed surface Va(111) begins at around T = 800 K. The onset of an adlayer involves generation of vacancies in the first surface layer, while at higher temperatures ($T \geq 1600K$) vacancies in the underlying layers (the second and third layers) begin to appear via promotion of atoms to vacant places of the first surface layer, i.e. so called interlayer vacancy migration mechanism. Atom migration from the deeper layers increases significantly as the temperature approaches the melting point. In contrast to the Va(111) sample, an adlayer formation and generation of adatom-vacancy pairs in the other samples becomes observable at more elevated temperatures (about T=1400 K for the Va(001) and T = 2000K for the Va(011)). Practically, all adatoms come from the first surface layer.

The same effect of disordering and gradual thickening of the surface region was observed in computer simulations of surface premelting of fcc metals [24,79,80,86,87,89]. The least packed face (110) of fcc metals (in analogy with the (111) bcc face) begins to disorder first, while the close packed (111) face (in analogy with the (011) bcc face) preserves its ordered crystalline structure almost up to the melting point.

Knowledge of the equilibrium averaged number of atoms in the adlayer allows us to estimate the surface defect formation energy E_s according to the formula:

$$n = exp(-E_s/k_BT) \tag{5.14}$$

where n is the adlayer occupation, e.g. the equilibrium averaged number of atoms in the adlayer at a current temperature divided by the number of atoms in the layer at a zero temperature. The adlayer occupation vs. temperature fitted to the Boltzmann factor $n = exp(-E_s/k_BT)$ are shown in Figs. 5.33-5.35 for the Va(111), Va(001) and Va(011), respectively.

The calculated surface defect formation energies of the various low-index faces of

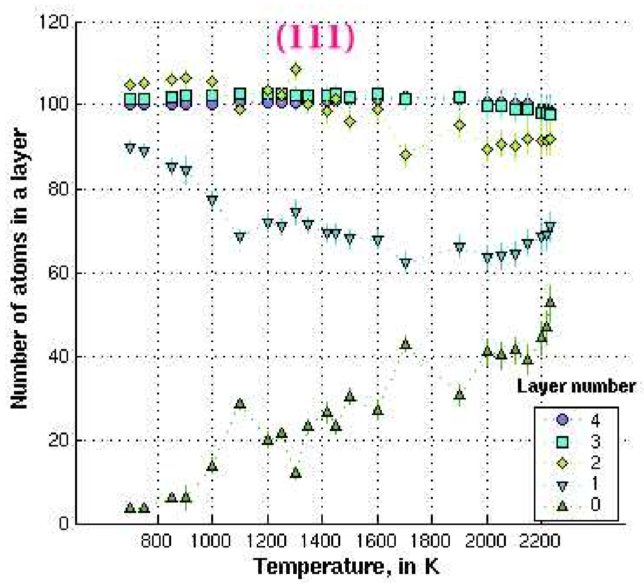


Figure 5.31: Layer occupation of the $Va(111)$ as a function of temperature for the adatom layer and the first few crystal layers.

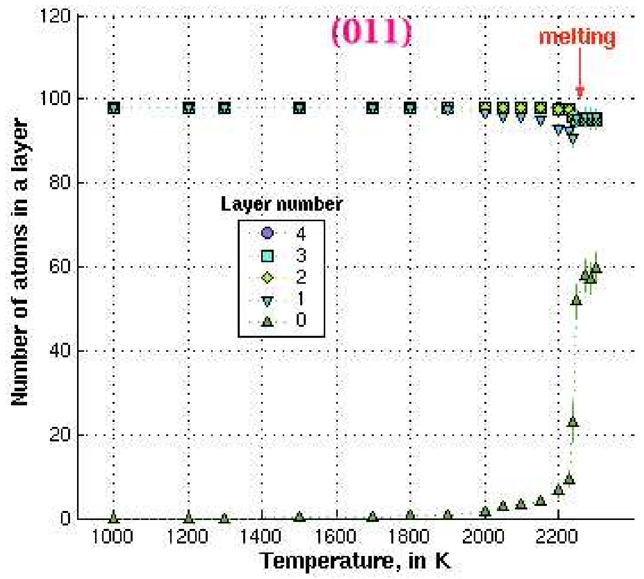


Figure 5.32: Layer occupation of the Va(011) as a function of temperature for the adatom layer and the first few crystal layers.

vanadium are tabulated in Table 5.3. Data for various faces of copper (fcc lattice) [80] are given for comparison. The surface defect formation energy is largest for the close packed surfaces Va(011) in case of a bcc lattice, and Cu(111) in case of a fcc lattice, and the lowest one for the least packed surfaces, the Va(111) and Cu(011), respectively.

Surface	(111)	(001)	(011)
$Va\ (bcc)\ E_s$	$0.27 \pm 0.04\ eV$	$0.84 \pm 0.02\ eV$	$2.36 \pm 0.01\ eV$
$Cu\ (fcc)\ E_s$	$1.92\ eV$	$0.86\ eV$	$0.39\ eV$

Table 5.3: Energy of surface defects (adatom-vacancy pairs) is calculated using the adlayer occupation data of the various surfaces of vanadium as a function of temperature. The data for copper is from ref. [80]

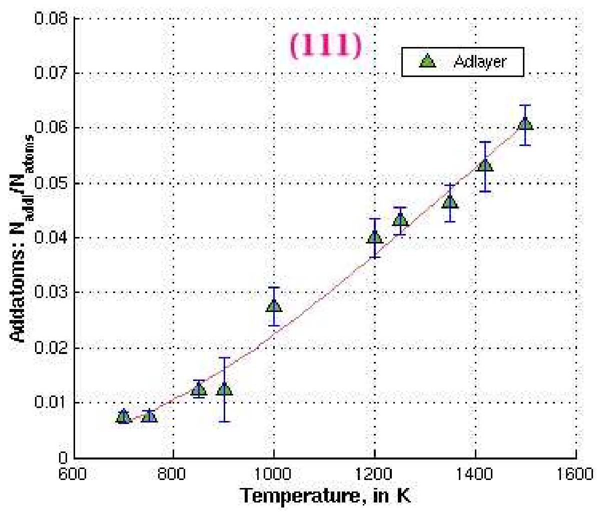


Figure 5.33: Equilibrium adlayer occupation vs. temperature for the $Va(111)$.

It is possible, that the adatom-vacancy pairs creation in the first crystal layers at high temperature is not the only one surface defect formation mechanism. Recent

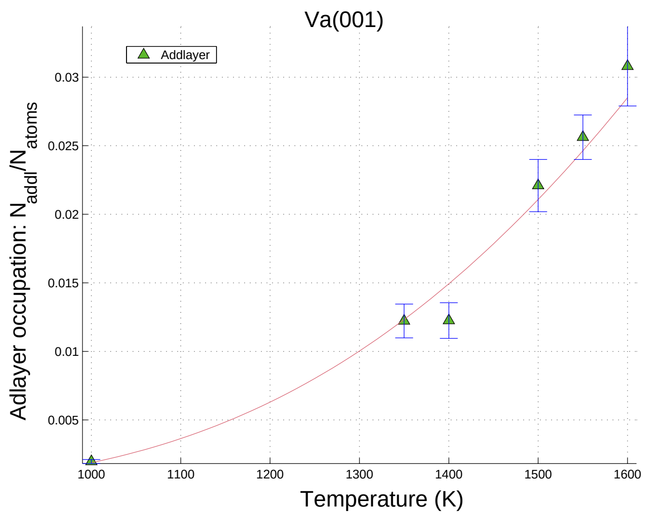


Figure 5.34: Equilibrium adlayer occupation vs. temperature for the $Va(001)$.

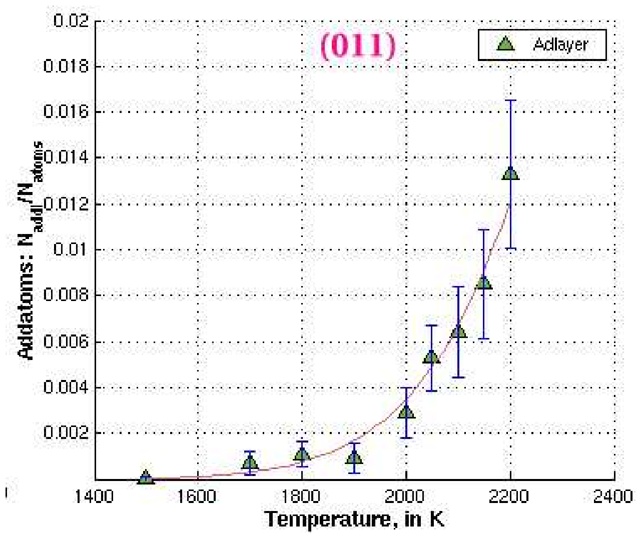


Figure 5.35: Equilibrium adlayer occupation vs. temperature for the $Va(011)$.

calculations carried out for fcc metals [79,89] indicate that it is energetically more favorable for the least packed surface to form a pair consisting of a divacancy and two adatoms, than to form two independent vacancy-adatom pairs, i.e. surface defect formation is a kind of a cooperative phenomenon. Therefore, surface defect formation energy, calculated on the basis of information about the adlayer occupation number, may be related to more complicated mechanism of defect formation. We did not study in detail the adatom-vacancy formation mechanism and used the obtained equilibrium averaged adlayer occupation numbers for a straightforward estimation of the adatom-vacancy surface defect formation energy.

Surface defect formation energy, calculated for the least packed surface, can be related to the thermodynamic melting point T_m according to E. Polturak et al.[92], i.e. $E_{def} \simeq ck_BT_m$, where c is a constant which is determined below and k_B is Boltzmann factor. In this scenario, the nucleation of a liquid phase on the free surface of a solid can be associated with the same mechanism which describes melting in a bulk of a solid (the Born criterion). Bulk melting occurs once the specific volume of a crystalline phase attains a critical value, which is close enough to the specific volume of a corresponding liquid phase. The same conditions, leading to melting in a bulk of a solid, are thought to occur at the surface, but at a lower temperature. In line with this assumption, we observe that the quasiliquid film appears first on the least packed surface Va(111), where surface defect formation is minimal with respect to the other low-index faces. Formation of the surface defects is considered to be a very effective mechanism of a stress release. The structure order parameters, plane radial distribution function, transport properties of the top surface layers in the premelting region exhibit liquid-like properties. We can assume that at high temperatures the shear elastic stress at the surface is insignificant. Therefore the Born criterion (at zero external stress) can be applied to the surface layers, and in this way it is possible to give explanation why do melting of a crystal begins at the free surface.

To obtain a linear relation between the melting temperature and surface defect formation energy, let us consider the evolution of the volume of a surface layers with temperature [92]. At low temperature, the volume of the first crystal layer is given by $V = v_0 N$, where v_0 is the volume per atom, and N is the number of atoms in the layer. While the temperature increases, the layer expands, and the atoms are displaced of the layer forming adatom-vacancy pairs. The temperature dependence of $V(T)$ can be written as:

$$V(T) = v_0(T)N + v_d N_d = v_0(T)N\left(1 + \frac{v_d N_d}{v_0(T)N}\right) \tag{5.15}$$

here v_d is the volume added to the layer by a creation of one defect, and N_d is the number of surface defects, which dependence on temperature is given by the Boltzmann factor:

$$\frac{N_d}{N} = exp\left(-\frac{E_s}{k_B T}\right) \tag{5.16}$$

where E_s is the surface formation energy. According to the Born criterion, at $T = T_m$ the V/N reaches its critical value that of the liquid phase (melt), then volume per atom is $V/N \approx v_{liq}$. Taking a natural logarithm of V/N (eq. 5.15) and rearranging this expression, we can restate the Born criterion as a linear relation between E_s and T_m.

$$E_s = \left\{ln\left(\frac{v_d}{v_0(T_m)}\right) - ln\left(\frac{v_{liq}}{v_0(T_m)} - 1\right)\right\} k_B T_m = c k_B T_m \tag{5.17}$$

It is assumed, in the lowest order approximation, that $v_d \simeq v_0(T_m)$. Indeed, upon creation of an adatom, one atomic volume is added to the first surface layer. To be more precise, since the adatom is weakly bounded to the surface, its vibration amplitude and therefore effective volume should be somewhat larger than $v_0(T)$. However, due to the reconstruction of the atoms around the created vacancy, it is expected that the volume of the vacancy to be smaller than the average atomic volume $v_0(T)$. Within the logarithmic accuracy of the equation (5.16) it is expected that these correlations cancel each in the first approximation, i.e. $v_d \simeq v_0(T_m)$ and consequently

$$E_s = \left\{ -ln \left(\frac{v_{liq}}{v_0(T_m)} - 1 \right) \right\} k_B T_m = ck_B T_m \tag{5.18}$$

The right hand side of this equation predicts the value of E_s without any adjustable parameters. To check it validity, we compare value of E_s (for the least packed surface Va(111)) calculated using this equation and E_s obtained directly in the MD simulations of bulk and surface melting. According to the results of the MD study of mechanical melting the volume per atom in the bulk at T_m is $v_0(T_m) = 14.7 \pm 0.3\ \dot{A}^3$, while the volume per atom in the melt is $v_0(T_m) = 15.47 \pm 0.3\ \dot{A}^3$. The thermodynamic melting point is found to be $T_m = 2220 \pm 18\ K$ by using the method proposed by Lutchko et al. [77]. Hence we obtain $E_s = 0.52 \pm 0.04\ eV$ by means of the formula (5.30). We can also estimate the surface defect formation energy using experimental data: the melting temperature $T_m = 2183\ K$, the volume per atom in the melt is $v_0(T_m) = 15.431\ \dot{A}^3$, and the volume per atom in the crystalline bulk at T_m can be estimated approximately as

$$v_b(T_m) \approx v_0(T_0) \exp(\alpha(T_m - T_0)) \tag{5.19}$$

where $T_0 = 300\ K$, the volume at $v_0(T_0) = 14.03\ \dot{A}^3$ and $\alpha = 8 \times 10^{-6} 1/K$ is the thermal expansion coefficient of vanadium measured at T_0. Thus we can calculate in approximation $v_b(T_m) \approx 14.25\ \dot{A}^3$ and consequently $E_s \approx 0.468\ eV$. The result of the MD simulations of surface melting the surface defect formation energy E_s of the least packed surface Va(111) is found to be $E_s = 0.27 \pm 0.05\ eV$ relying on the analysis of the adlayer occupation as a function of temperature. We may conclude that agreement between the theoretically predicted and calculated values of E_s is encouraging.

Finally we remark, that the Born melting criterion was applied in a study of surface melting of fcc metals [92]. It was found that a linear relation between the surface defect formation E_s energy and the melting temperature T_m is valid for metals with the fcc lattice structure (See Fig. 5.36 and Tab. 5.4) This linear relation agrees quantitatively with the experimental and simulation values of the activation energy of the surface defects, without any additional adjustable parameters. Our results imply that the Born criterion correctly describes both surface and bulk melting, and may well provide the "missing link" which will finally tie together these two scenarios for the melting transition.

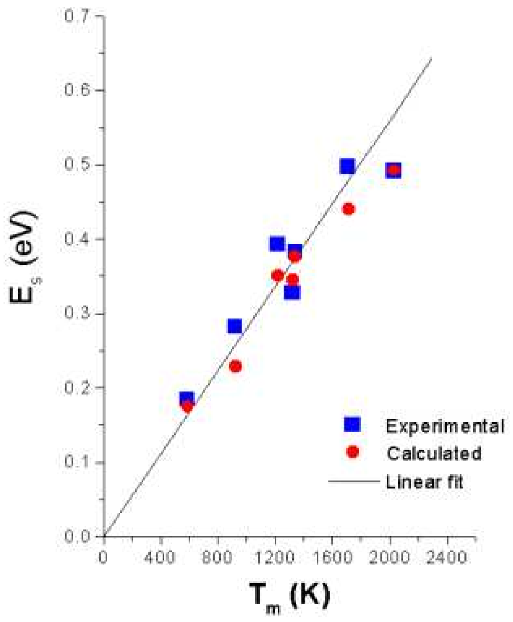


Figure 5.36: The activation energy of surface defects E_s vs. T_m for Pb, Al, Ag, Au, Cu, Ni and Pt. The solid line is a linear regression. From ref. [92]

Metal	Pb	Al	Ag	Au	Cu	Ni	Pt
T_m K	600	933	1235	1337	1356	1726	2045
E_s eV	0.18 ± 0.04	0.26 ± 0.02	0.37 ± 0.03	0.32 ± 0.03	0.41	0.49 ± 0.04	0.49 ± 0.05

Table 5.4: The activation energy of surface defects E_s vs. the melting temperatures for fcc metals. The data from ref. [92]

Chapter 6

Summary and conclusions

The existing theories aimed to explain melting transition are still far from being complete and raise futher questions. Hence the main purpose of the present project was to gain a better understanding of the mechanism of melting transition, and especially to investigate the rôle of point defects and the surface of the solid in this phenomenon.

An interatomic potential proposed by Finnis and Sinclair [42] was chosen to investigate the bulk and the surface of vanadium at different temperatures. In order to validate the computer program and to check the potential we calculated thermal expansion coefficient and the shear moduli of a perfect sample of vanadium at various temperatures. The results were compared with available experimental data, and the agreement between was found to be reasonable.

After validation of the pure system, point defects were introduced into the lattice by either removal an atom (vacancy) or by addition (self-interstitial). The most stable, energetically favorable configuration of atoms around an interstitial at low temperatures was found by using simulated tempering method. The configuration of point defects in the bulk of vanadium (bcc lattice) was found to be a dumb-bell $< 110 >$ split-interstitial with the formation energy $E_f = 4.18 \pm 0.02$ eV, in contrast to $< 100 >$ split-interstitial of copper (fcc lattice) with the formation energy $E_f = 3.28$ eV.

Point defects change the volume of the solid, while the shape of the sample, below the melting point, is almost unchanged, provided that the concentration of point defects is small enough. Self-interstitials expand the sample, while the introduction of vacancies leads to decrease in the volume of the sample. These changes of the volume due to point defects are less noticeable for vanadium in comparison with copper. This can be ascribed to the more loose-packed structure of a bcc lattice, which is less distorted by the presence of defects than a fcc lattice. In addition, variation of the shear elastic moduli with the concentration of defects and temperature was investigated. We found that the shear moduli are softened as a result of the volume expansion of the solid, which is associated with increase in temperature and/or interstitial concentration. Comparison of our results with the corresponding data for copper shows that softening of the shear moduli is less pronounced for vanadium. This again is attributed to the more compact structure of the fcc lattice of copper.

There is a strong evidence that the Born instability is the trigger for melting. This instability could be set in by self-interstitials which expand the solid up to a critical volume, where the lattice of crystal becomes mechanically unstable and collapses. This defect mediated mechanical melting occurs at temperatures below the melting temperature of the perfect crystal. We found that the critical volume at which crystal melts is independent of the path throw the phase space by which it reached, either by heating the crystal without defects, or by adding defects at a constant temperature. We performed computer simulations with various concentrations of point defects using the Parinello-Rahman method. Starting with a given concentration of point defects we gradually heat the solid up to the melting transition at T_b. In addition, T_b is determined relying on the dependence of the shear elastic moduli on the temperature at a given concentration of point defects. The melting point is extracted by a continuous extrapolation of the shear modulus C' to zero (at the zero external stress). We found that interstitials lower T_b by approximately 50 K for the defect concentration 0.5%, however for copper the effect is more pronounced. Vacancies almost do not affect T_b(at least at small concentrations).

The temperature at which bulk melting transition occurs is found to be $T_b = 2500 \pm 8K$. This temperature is larger than the thermodynamic melting temperature, $T_m = 2220 \pm 10K$ at which the vanadium crystal with the free surface melts (the same effect of was observed for copper [67] which has $T_b = 1580K$ and $T_m = 1323K$). The thermodynamic melting temperature T_m of vanadium was determined using the method proposed by Lutchko et al.[77]. The simulations were carried out for various low-index surfaces of vanadium, and it was found that a sample with the least packed surface melts at lower temperature, than a sample with the close packed surface structure (the difference is about 10 K).

Surface premelting, i.e. formation of the quasiliquid melt in the surface region, was studied in detail. The structural, transport and energetic properties of low-index faces Va(001), Va(011) and Va(111) were examined at various temperatures. We found that upon increasing the temperature, the vibrations of atoms in the surface region become larger than in the bulk, they start to disturb each other. As a result a formation of point defects begins in the surface region. These defects migrate between the surface layers, opening the way to a “disordered” surface regime (premelting). The regime of premelting is characterized by enhanced diffusion of atoms in the surface region, and by an increased formation of the surface defects. Finally, an adlayer emerges on the top of the surface layer and a quasiliquid phase appears as an intermediate layer between the solid and gas phases. The transition region between the solid and quasiliquid is the sharpest one for the close packed Va(011) surface. The surface premelting of the Va(011) is observed only in the close proximity of T_m, reflecting a large activation energy need for formation of surface defects. The surface premelting is more noticeable for the loose packed surface, and especially for the least packed face Va(111).

The results of our simulations of surface premelting of the bcc metal vanadium are similar to the result obtained for various fcc metals. The least packed surface (011) of those metals (with fcc lattice) begins to disorder first at low temperatures, while the close packed (111) surface preserves its crystalline structure almost up to T_m (in some cases could be superheated).

Why do crystals start to melt at their surface? To answer this question we applied the Born criterion of melting transition to the surface region. The generation of point defects is enhanced with increase of the temperature in the surface region. Therefore, these defects expand the solid up to the point where the mechanical instability of the lattice (in the surface region)sets in. According to E. Polturak [92], the Born criterion is being applied to the surface melting leads to a simple linear relation between the activation energy of surface defects and T_m. This theoretical prediction was tested by results of experiments and computer simulations of fcc metals. In order to test the validity of the theory for vanadium (bcc lattice), we evaluated the activation energy of the surface defects for the least packed Va(111) face and compared it with the value predicted by the theory. The available experimental data and results of bulk simulations were used as input parameters in the theoretical calculations of surface defects formation energy. The activation energies of surface defects estimated from the Born criterion and obtained in our computer simulations are in reasonable agreement. Therefore, a general conclusion is made that the Born criterion correctly describes both surface and bulk melting, and may well provide the "missing link" which will finally tie together these two scenarios of the melting transition.

Many aspects of melting transition were investigated in our computer simulations, nevertheless many other questions remain to be answered. First, one could investigate the mechanism of bulk and surface melting using alternative versions of the many-body potential for vanadium (or any other bcc metal) and compare with the present results. In particular, it would be very interesting to apply the latest version of an interaction potential for bcc metals, the second nearest-neighbor modified embedded atom potential (**MEAM**) developed by B. Lee et al. [100]. In order to considerably improve the accuracy of calculations we one could to increase the number of atoms (and number of the free layers in case of surface melting). This would be achieved by substantional improvement the parallel version of our **MD** code and more intensive use of supercomputers.

Another interesting problem, which was outside of the scope of the present research, is the nature of the phonon spectrum in the vicinity of point defects. It is expected that point defects create additional local and resonance modes in the phonon spectrum. These modes play an essential role in the bulk and surface melting transition. In addition, it would be very interesting to verify whether the inhomogeneous bulk melting transition (i.e. formation of the "clusters" of the liquid phase inside the crystalline phase) observed in computer simulation with the simple LJ potential [6] could be observed if more a complicated EAM potential is used. Besides that, it would be instructive to investigate whether the bulk melting transition could be initiated around complexes of point defects (e.g. polyvacancies or polyinterstitials). It would be very important to investigate the relation between surface premelting and surface roughening as well as the surface reconstruction at low temperatures.

Finally, it is a fascinating problem to investigate bulk melting as well as surface premelting of metals with *hcp lattice structure*. Furthermore, next in order of complexity to crystals containing point defects are crystals with more complex defects (dislocations, disclinations, etc). The challenge is how to study the mechanism of the melting transition of those crystals by means of computer simulations?

Appendix A

Predictor - corrector method

Amongst the different versions of integrators, the predictor-corrector (**PC**) [54-56], method was chosen for our simulations. The goal is to solve the second order ordinary differential equations:

$$\ddot{x}_i = f_i(x_i, \dot{x}_i, t) \tag{A.1}$$

where

$$\dot{x} = \frac{dx}{dt} \tag{A.2}$$

and

$$f(t) \equiv f(x(t), \dot{x}(t), t) \tag{A.3}$$

This is Newton's equations of motion for the x component of the ith atom, and the same equation could be written for y and z. We omit the i index now for simplicity. The **PC** method is composed of three steps: *prediction, evaluation, and correction.* In particular, from the initial position $x(t)$ and velocity $\dot{x}(t)$ at time t, the steps are as follows:

Prediction

1.) Predict the position $x(t+h)$, where h is the time step of integration:

$$x(t+h) = x(t) + h\dot{x}(t) + h^2 \sum_{i=1}^{k-1} \alpha_i f(t + h(1-i)) \tag{A.4}$$

In our simulation we use k=4, and therefore we can write explicitly the formula, using the $\{\alpha_i\}$ from [54] :

$$x(t+h) = x(t) + h\dot{x}(t) + h^2 \left\{ \frac{19}{24} f(t) - \frac{10}{24} f(t-h) + \frac{3}{24} f(t-2h) \right\} \tag{A.5}$$

2.) Predict the velocities:

$$\dot{x}(t+h) = \frac{x(t+h) - x(t)}{h} + h \sum_{i=1}^{k-1} \alpha'_i f(t + h(1-i)) \tag{A.6}$$

and therefore:

$$\dot{x}(t+h) = \frac{x(t+h) - x(t)}{h} + h \left\{ \frac{27}{24} f(t) - \frac{22}{24} f(t-h) + \frac{7}{24} f(t-2h) \right\} \tag{A.7}$$

Evaluation

3.) Evaluate the force using the predicted values:

$$f(t+h) \equiv f(x(t+h)) \tag{A.8}$$

Correction

4.) The final step is to correct the predictions by using some combinations of the predicted and previous values of position and velocity:

$$x(t+h) = x(t) + h\dot{x}(t) + h^2 \sum_{i=1}^{k-1} \beta_i f(t + h(2-i)) \tag{A.9}$$

or

$$x(t+h) = x(t) + h\dot{x}(t) + h^2 \left\{ \frac{3}{24} f(t+h) + \frac{10}{24} f(t) - \frac{1}{24} f(t-h) \right\} \tag{A.10}$$

and for velocity:

$$\dot{x}(t+h) = \frac{x(t+h)-x(t)}{h} + h\sum_{i=1}^{k-1} \beta'_i f(t+h(2-i)) \tag{A.11}$$

and therefore:

$$\dot{x}(t+h) = \frac{x(t+h)-x(t)}{h} + h\left\{\frac{7}{24}f(t+h) + \frac{6}{24}f(t) - \frac{1}{24}f(t-h)\right\} \tag{A.12}$$

The set of parameters $\{\alpha_i, \alpha'_i, \beta_i, \beta'_i\}$ $i = 1, 2, 3$ promotes numerical stability of the algorithm. Gear [56] determined their values by applying the predictor-corrector algorithm to linear differential equations and analyzing the stability of the method.

The values of $\{\alpha_i, \alpha'_i, \beta_i, \beta'_i\}$ were chosen specially to make the local truncation error of order of $O(h^4)$, and the global error for the second order differential equations is order of $O(h^3)$. It worth to mention, that interactions are evaluated using the results of the predictor step, but they are not re-evaluated again following the corrector step. It was shown that mean error, induced by the absence of force re-evaluation, is insignificant [55].

The main ingredient of the predictor - corrector method is the corrector step, which accounts for a *feedback mechanism.* The feedback can damp the instabilities that might be introduced by the predictor step. This method provides both the positions and velocities of the atoms at the same time and it can be used to calculate forces, which depend explicitly on the velocity, this is in turn usually needed in algorithms which control temperature and pressure.

Appendix B

Nose-Hoover algorithm

The simplest extension of the NVE ensemble is the canonical one (**NVT**), where the number of particles, the volume and the temperature are fixed to the prescribed values. The temperature T, in contrast to the number of particles N and volume V, is an intensive parameter. The extensive counterpart is kinetic energy, related to T through:

$$T = \frac{2}{3}\frac{E_{kin}}{Nk_b} = \frac{1}{3Nk_b}\sum_{i=1}^{N}\frac{p_i^2}{m} \tag{B.1}$$

There are different methods which control the temperature: the differential, proportional, stochastic and the integral thermostat [59]. The integral thermostat method was selected for our MD simulations. The integral thermostat method (which sometimes called the *extended system method* or the *Nose-Hoover algorithm* [60]) introduces additional degrees of freedom into the system's Hamiltonian, for which equation of motion can be derived. These equations for the additional degrees of freedom are integrated together with "usual" equations for spatial coordinates and momenta.

The idea of the method proposed by Nose [59,96] was to reduce the effect of an external system, acting as heat reservoir, to an additional degree of freedom. This heat reservoir controls the temperature of the given system, i.e. the temperature fluctuates around target value. Actually, the thermal interaction between the heat reservoir and the system results in exchange of the kinetic energy between them. Nose introduced two sets of variables: *real* $\{p_i, q_i\}$ and *virtual* $\{\pi_i, \rho_i\}$ ones. The virtual variables are derived from the so called Sundman's transformation [62]:

$$s = \frac{d\tau}{dt} \tag{B.2}$$

where τ is the virtual time, t is the real time and s is a resulting scaling factor, which also treated as a dynamical variable. The transformation from the *virtual variables* $\{\pi_i, \rho_i\}$ to the *real ones* $\{p_i, q_i\}$ is performed according to:

$$p_i = \pi_i \tag{B.3}$$

$$q_i = \rho_i \tag{B.4}$$

The introduction of the effective mass M_s connects also a momentum to the additional degree of freedom π_s. The resulting Hamiltonian, expressed in terms of the virtual coordinates can be written as:

$$H^* = \sum_{i=1}^{N} \frac{\pi_i^2}{2ms} + U(\rho_1, \rho_2, ..., \rho_N) + \frac{\pi_s^2}{2M_s} + gk_bTln(s) \tag{B.5}$$

where $g = 3N+1$ is the number degrees of freedom of the extended system (N particles + 1 the new degree of freedom). It was shown, that this Hamiltonian H^* leads to a density of probability in phase space, corresponding to the canonical ensemble[61].

The equations of motion obtained from the Hamiltonian are:

$$\frac{d\vec{\rho}_i}{d\tau} = \frac{\partial H^*}{\partial \vec{\pi}_i} = \frac{\vec{\pi}_i}{ms^2} \tag{B.6}$$

$$\frac{d\vec{\pi}_i}{d\tau} = -\frac{\partial H^*}{\partial \vec{\rho}_i} = -\frac{\partial U}{\partial \vec{\rho}_i} \tag{B.7}$$

$$\frac{ds}{d\tau} = \frac{\partial H^*}{\partial \pi_s} = \frac{\pi_s}{M_s} \tag{B.8}$$

$$\frac{d\pi_s}{d\tau} = -\frac{\partial H^*}{\partial s} = \frac{gk_bT}{s} + \sum_{i=1}^{N} \frac{\pi_i^2}{2ms} \tag{B.9}$$

If one transforms these equations back into the real variables $\{p_i, q_i\}$ and introduces a new variable ζ:

$$\zeta = s\frac{ds}{dt} = s\frac{ds}{d\tau}\frac{d\tau}{dt} = s\frac{\partial H^*}{d\pi_s}\frac{d\tau}{dt} = s^2\frac{\pi_s}{M_s} \tag{B.10}$$

then one obtains (according to Hoover [63]):

$$\frac{d\vec{q}_i}{dt} = \frac{\vec{p}_i}{m_i} \tag{B.11}$$

$$\frac{d\vec{p}_i}{d\tau} = -\frac{\partial U}{\partial \vec{q}_i} - \zeta\vec{p}_i \tag{B.12}$$

$$\frac{\partial ln(s)}{\partial t} = \zeta \tag{B.13}$$

$$\frac{d\zeta}{dt} = \frac{1}{M_s}\left(\sum_{i=1}^{N} \frac{p_i^2}{2m_i} - gk_bT\right), \qquad p_i \equiv |\vec{p}_i| \tag{B.14}$$

These equations describe the *Noose-Hoover thermostat* [64]. The parameter M_s is a thermal inertia parameter, which determines rate of the heat transfer. The value of this parameter must be set carefully, because if it is chosen to be too small the phase space of the system will not be canonical [65], and it is chosen to be too large the temperature control will not be efficient. In our simulation, for example, we set $M_s = 1/6$. By means of the *Noose-Hoover* thermostat we can impose time averaged value of temperature to be equal to the prescribed value.

Appendix C

Calculation of the shear moduli

If an external force acting on a body or if one part of the body applies force on another part, it is said that the body is in the state of stress. Stress is defined in units of force per unit area $\sigma = F/A$ and can be characterized in general case by the stress tensor. Strain describes the state of deformation of a solid; there is dilatational strain which changes the volume, but not the shape and deviatoric strain which in contrast changes the shape, but not the volume.

For a field of deformations $\vec{u}(x, y, z)$, we can define a symmetric strain tensor in the following way:

$$\epsilon_{\alpha\beta} = \frac{1}{2}\left(\frac{\partial u_\alpha}{\partial x_\beta} + \frac{\partial u_\beta}{\partial x_\alpha}\right), \quad \alpha, \beta \ = \ 1, 2, 3 \tag{C.1}$$

Hook discovered that the strain ϵ is proportional to the stress σ:

$$\epsilon = S\sigma \quad or \quad \sigma = C\epsilon \tag{C.2}$$

These relation between the stress and the strain could be generalized for an anisotropic solid, in the terms of tensors:

$$\epsilon_{\alpha\beta} = \sum_{\gamma\delta} S_{\alpha\beta\gamma\delta}\sigma_{\gamma\delta} \tag{C.3}$$

or

$$\sigma_{\alpha\beta} = \sum_{\gamma\delta} C_{\alpha\beta\gamma\delta}\epsilon_{\gamma\delta} \tag{C.4}$$

The fourth-rank tensor $C_{\alpha\beta\gamma\delta}$ has $3x3x3x3 = 81$ components, but thanks to cubic symmetry (of simple cubic, fcc and bcc lattices) there are only *three independent components* of the elasticity tensor $C_{\alpha\beta\gamma\delta}$.

It is customary to reduce the number of subscripts by means of an abbreviated notation for pairs of coordinate directions, as follows [67]:
in the full tensor notation the indices of $C_{\alpha\beta\gamma\delta}$: 11, 22 , 33, 23 32, 13 31, 12 21,
while in the abbreviated notation they are $C_{\alpha\beta}$: 1, 2, 3, 4, 5, 6
and there is a correspondence:

$$11 \rightarrow 1 => C_{1111} \equiv C_{11} \tag{C.5}$$

$$22 \rightarrow 2 => C_{1122} \equiv C_{12} \tag{C.6}$$

$$33 \rightarrow 3 => C_{1133} \equiv C_{13} \tag{C.7}$$

$$23, 32 \rightarrow 4 => C_{3223} = C_{2323} \equiv C_{44} \tag{C.8}$$

$$13, 31 \rightarrow 5 => C_{3113} = C_{1313} \equiv C_{55} \tag{C.9}$$

$$21, 12 \rightarrow 6 => C_{1221} = C_{2121} \equiv C_{66} \tag{C.10}$$

In these notation the three independent components of the elasticity tensor [67]:

$$C_{11},\ C_{12},\ C_{44}$$

Of course, one can use linear combinations of these elastic constants, to express other physical quantities such as: bulk modulus: $B = C_{11} + \frac{1}{2}C_{12}$ and shear modulus: $C' = \frac{1}{2}(C_{11} - C_{12})$ The free energy of a distortion can be expressed in powers of the strain tensor components. Assuming that the distortion of a crystal lattice too small to break the four-fold cubic symmetry and neglecting high order terms we can write the free energy as:

$$U_{el} = \sum_{\alpha\beta} C_{\alpha\beta}\epsilon_{\alpha\beta} = \frac{1}{2}B(2\epsilon_{11} + \epsilon_{23})^2 + \frac{1}{3}C'(\epsilon_{33} - \epsilon_{11})^2 + \frac{1}{2}C_{44}(\epsilon_{11}^2 + \epsilon_{13}^2) \qquad \text{(C.11)}$$

On the base of elasticity theory, Born [68] derived the general conditions for stability of a crystal lattice:

$$B > 0,\ C' > 0 \ and \ C_{44} > 0 \qquad \text{(C.12)}$$

and showed that the melting temperature could be found from the condition $C_{44}(T_m) = 0$. Therefore, calculation of these elastic coefficients is very important in studying the melting transition. The elastic constants C_{11}, C_{12}, C_{44} are calculated by means of fluctuation formulas obtained by Ray and Rahman [57,58]. It was shown that the elastic coefficients $C_{\alpha\beta\gamma\delta}$ are related to fluctuations of the stress tensor

$$< \sigma_{\alpha\gamma}\sigma_{\beta\rho} > - < \sigma_{\alpha\rho} >< \sigma_{\beta\gamma} > \qquad \text{(C.13)}$$

and the ensemble average of the Born term $B_{\alpha\beta\gamma\rho}$. The elastic coefficients are calculated in the following way:

$$\begin{aligned} C_{\alpha\beta\gamma\rho} &= (-V/k_bT)\,(< \sigma_{\alpha\gamma}\sigma_{\beta\rho} > - < \sigma_{\alpha\rho} >< \sigma_{\beta\gamma} >) \\ &+ (2Nk_bT/V)\,(< \delta_{\alpha\gamma}\delta_{\beta\rho} > + < \delta_{\alpha\rho}\delta_{\beta\gamma} >) + < B_{\alpha\beta\gamma\rho} > \end{aligned} \qquad \text{(C.14)}$$

where $\sigma_{\alpha\beta}$ is the stress tensor, $\delta_{\alpha\beta}$ is Kroneker delta, and $<>$ is ensemble averaging, N, V, T are the volume, the number of atoms and the temperature, respectively.

The Born term is defined as:

$$B_{\alpha\beta\gamma\delta} = \frac{1}{2V} \sum_{i,j:\ i\neq j}^{N_{neigh}} \left(\frac{\partial^2 U_{pot}}{\partial r_{ij}^2} - \frac{1}{r_{ij}} \frac{\partial U_{pot}}{\partial r_{ij}} \right) \frac{r_{ij\alpha} r_{ij\beta} r_{ij\gamma} r_{ij\delta}}{r_{ij}^2} \qquad \text{(C.15)}$$

here $\alpha\beta\gamma\delta = 1, 2, 3$ are Cartesian indices and $i, j = 1, 2, .., N_{neigh}$ numerate the neighbor atoms, and $r_{ij} = |\vec{r}_i - \vec{r}_j|$ is distance between the atoms i and j. Potential energy is represented by the Finnis - Sinclair potential:

$$U_{pot} = \frac{1}{2}\sum_{i,j:\ i\neq j}^{N} \Phi(r_{ij}) + \sum_{i=1}^{N} F(\rho_i) \tag{C.16}$$

and therefore the Born term is given by the formula:

$$B_{\alpha\beta\gamma\delta} = (B1)_{\alpha\beta\gamma\delta} + (B2)_{\alpha\beta\gamma\delta} + (B3)_{\alpha\beta\gamma\delta} \tag{C.17}$$

where:

$$(B1)_{\alpha\beta\gamma\delta} = \frac{1}{2V}\sum_{i,j:\ i\neq j}^{N} \left(\frac{\partial^2\Phi}{\partial r_{ij}^2} - \frac{1}{r_{ij}}\frac{\partial\Phi}{\partial r_{ij}}\right)\frac{r_{ij\alpha}r_{ij\beta}r_{ij\gamma}r_{ij\delta}}{r_{ij}^2} \tag{C.18}$$

$$(B2)_{\alpha\beta\gamma\delta} = \frac{1}{2V}\sum_{i,j:\ i\neq j}^{N} F'[\rho_i]\left(\frac{\partial^2\rho}{\partial r_{ij}^2} - \frac{1}{r_{ij}}\frac{\partial\rho}{\partial r_{ij}}\right)\frac{r_{ij\alpha}r_{ij\beta}r_{ij\gamma}r_{ij\delta}}{r_{ij}^2} \tag{C.19}$$

here

$$\rho_i = \sum_{j}^{N_{neigh}} \rho(r_{ij}) \tag{C.20}$$

and finally:

$$(B3)_{\alpha\beta\gamma\delta} = \sum_{i}^{N} F''[\rho_i] g_{i\alpha\beta} g_{i\gamma\delta} \tag{C.21}$$

while the function $g_{i\alpha\beta}$ defined as:

$$g_{i\alpha\beta} = \frac{1}{2}\sum_{j:\ j\neq i}^{N_{neigh}} \frac{\rho'(r_{ij}) r_{ij\alpha} r_{ij\beta}}{r_{ij}} \tag{C.22}$$

The main advantage of this method is the fast convergence of the stress fluctuation term to its equilibrium value.

In order to calculate the shear elastic modulus using the above mentioned formula two steps are required. The first step is to find the zero-stress reference matrix for the computational cell $H^0_{\alpha\beta}$ and the second step is to run the **NVT** MD simulation to calculate the elastic coefficients, using stress tensor fluctuations and average value of the $< B_{\alpha\beta\gamma\rho} >$ term.

Appendix D

Isothermal-isotension ensemble

An extended ensemble, which was implemented in this project, is the *isothermal-isotension ensemble* with constant number of atoms (**NtT**). We study the behavior of vanadium containing self-interstitials and vacancies, at temperatures close to the melting point. In this case, one expects changes of the volume and shape of the sample due to the thermal expansion and point defects. Therefore, in order to describe the system accordingly we have to implement an algorithm, which allows for fluctuation of the shape and volume of the computational cell, and at the same time controls the pressure of the system.

Parinello and Rahman in 1980 [66] invented a new method, in which the shape and volume of the computational cell are dynamical variables. This technique is very helpful in study of the phase transitions. The statistical ensemble designed by Parrinello and Rahman, combined with Nose-Hoover thermostat is identified as an isothermal-isotension (**NtT**) ensemble.

The realization of this method is in the following way. First of all, we introduce scaled coordinates $\{s^i_\alpha\}$ in addition to the usual ones $\{r^i_\alpha\}$:

$$s^i_\alpha = \sum_{\beta=1}^{N=3} H_{\alpha\beta} r^i_\beta \tag{D.1}$$

where the Greek indices $\alpha, \beta, ...$ are the coordinate indices $\{\alpha = 1, 2, 3 \; or \; \alpha = x, y, z\}$, the Latin indices i $\{ \; i = 1, N\}$ count the atoms, and $H_{\alpha\beta}$ is a transformation matrix.

The volume of the computational box is given by:

$$V = det(H_{\alpha\beta}) \tag{D.2}$$

One can introduce a metric tensor $G_{\alpha\beta}$ by using the $H_{\alpha\beta}$ matrix:

$$G_{\alpha\beta} = \sum_{\gamma=1}^{N=3} H_{\gamma\alpha} H_{\gamma\beta} \tag{D.3}$$

or

$$G = H^T H \tag{D.4}$$

where T is stands for transpose. A distance between any two atoms are i and j is simply given by:

$$r^2_{ij} = \sum_\alpha \left(r^i_\alpha - r^j_\alpha\right)^2 = \sum_{\alpha,\beta} \left(s^i_\alpha - s^j_\alpha\right) G_{\alpha\beta} \left(s^i_\beta - s^j_\beta\right) \tag{D.5}$$

where α and β are the coordinate indices.

The Lagrangian of the system has to be extended from 3N coordinate degrees of freedom to 3N+9, including the new 9 degrees of freedom of the real matrix $H_{\alpha\beta}$:

$$L = T - U \longrightarrow L' = T + T_H - U - U_H \tag{D.6}$$

where the extra 'kinetic' energy T_H and the corresponding 'potential' V_H energy are added to the original Lagrangian to account for the new decrees of freedom.

The 'kinetic' energy term is given by:

$$T_H = \frac{1}{2} Q \sum_{\alpha=1}^{3} \sum_{\beta=1}^{3} \dot{H}_{\alpha\beta} \dot{H}_{\alpha\beta} \tag{D.7}$$

where Q is a parameter, its physical meaning will be explained latter.

The 'potential' U_H energy term is:

$$U_H = PV \tag{D.8}$$

here V is volume of the system $V = det(H_{\alpha\beta})$ and P its pressure, which can be obtained according to the *virial theorem*:

$$P = Nk_bT + \frac{1}{3} < \sum_{i,j} \vec{r}_{ij}\vec{F}_{ij} > \tag{D.9}$$

where $\vec{F}_{ij}$ is the force acting on atom i due to the atom j, and $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ is the vector which connects the atoms i and j, and $< ... >$ stands for ensemble averaging.

The modified Lagrangian is written in the terms of new variables:

$$L' = \frac{1}{2}\sum_{i=1}^{3}\sum_{\alpha,\beta}^{3} m\dot{s}^i_\alpha G_{\alpha\beta}\dot{s}^i_\beta - \frac{1}{2}\sum_{i,j:\ i\neq j}^{N} U(r_{ij}) + \frac{1}{2}Q\sum_{\alpha=1}^{3}\sum_{\beta=1}^{3}\dot{H}_{\alpha\beta}\dot{H}_{\alpha\beta} - PV \tag{D.10}$$

here, for the sake of simplicity we omit for a while the terms corresponding to the *Noose-Hoover* thermostat. The equations of motion are obtained in the usual way:

$$\frac{\partial L'}{\partial s^i_\alpha} - \frac{d}{dt}\frac{\partial L'}{\partial \dot{s}^i_\alpha} = 0 \tag{D.11}$$

$$\frac{\partial L'}{\partial h^i_\alpha} - \frac{d}{dt}\frac{\partial L'}{\partial \dot{h}^i_\alpha} = 0 \tag{D.12}$$

Therefore we obtain:

$$m\ddot{s}^i_\alpha = \sum_\beta (H^{-1})_{\alpha\beta} f^i_\beta - \sum_{\beta,\gamma} m(G^{-1})_{\alpha\beta}(\dot{G})_{\beta\gamma}\dot{s}^i_\gamma \tag{D.13}$$

$$Q\ddot{H}_{\alpha\beta} = V\sum_\gamma(\sigma_{\alpha\gamma} - P\delta_{\alpha\gamma})(H^{-1})\beta\gamma \tag{D.14}$$

where $f^i_\beta = \sum_{j:\ j\neq i} F^{ij}_\beta$ and $m_1 = m_2 = ... = m_i \equiv m$,

the $\sigma_{\alpha\gamma}$ is the stress tensor which describes anisotropy of the solid.

The stress tensor $\sigma_{\alpha\beta}$ is defined in the following way [67]:

$$\sigma_{\alpha\beta} = \frac{1}{V}\left(\sum_{i=1}^{N}\frac{v_i^\alpha v_j^\beta}{m} + \frac{1}{2}\sum_{i,j:\ i\neq j}^{N}\frac{|\vec{F}_{ij}|r_{ij}^\alpha r_{ij}^\beta}{|\vec{r}_{ij}|}\right) \tag{D.15}$$

here v_i^α is αth component of the atom i velocity,

and r_i^α is αth component of the vector connecting atoms i and j.

The hydrostatic pressure components are the diagonal elements of the stress tensor:

$$p_x =< \sigma_{xx} >; \quad p_y =< \sigma_{yy} >; \quad p_z =< \sigma_{zz} > \tag{D.16}$$

The stress tensor can be rewritten in the terms of the scaled variables $\{s_\alpha^i\}$:

$$\sigma_{\alpha\beta} = \frac{1}{V}\left(\sum_{i=1}^{N} m(\sum_\gamma \dot{s}_\gamma^i H_{\gamma\alpha})(\sum_\delta H_{\beta\delta}\dot{s}_\delta^i) + \frac{1}{2}\sum_{i,j:\ i\neq j}^{N} H_{\alpha\gamma}(s_\delta^i - s_\delta^j)F_{ij,\beta}\right) \tag{D.17}$$

here $F_{ij,\beta}$ is the βth component of the force acting on the atom i due to the atom j.

The system is driven by *dynamic imbalance* between the applied external stress and the internally generated stress. The Q parameter corresponds to the *relaxation time* of recovery from the imbalance between the external and the internal stress. It can be shown that the value of Q does not influence the ensemble average [67].

The equations of motion generated by the Lagrangian can be effectively solved numerically. One has take into account that the Parinello-Rahman model is represented by a system of *coupled ordinary differential equations* for scaled coordinates $\{s_\alpha^i\}$ and for the $\{H_{\alpha\beta}\}$ elements of the H matrix. These equations have to be solved *simultaneously.*

This can be done using the predictor-corrector (**PG**) method, which includes the following steps:

1.) New scaled positions $\{s_\alpha^i(t+\delta t)\}$ and velocities $\{\dot{s}_\alpha^i(t+\delta t)\}$ are predicted on the basis of the old ones, including the accelerations $\{s^i_\alpha(t), \dot{s}^i_\alpha(t), \ddot{s}^i_\alpha(t), \ddot{s}^i_\alpha(t-\delta t), \ddot{s}^i_\alpha(t-2\delta t)\}$:

$$s^i_\alpha(t+\delta t) = s^i_\alpha(t) + \dot{s}^i_\alpha(t)\delta t + (\delta t)^2 \sum_{k=1}^{3} a_k^{PG} \ddot{s}^i_\alpha(t+(1-k)\delta t) \tag{D.18}$$

$$\dot{s}^i_\alpha(t+\delta t)\delta t = s^i_\alpha(t+\delta t) - s^i_\alpha(t) + \dot{s}^i_\alpha(t)\delta t + (\delta t)^2 \sum_{k=1}^{3} b_k^{PG} \ddot{s}^i_\alpha(t+(1-k)\delta t) \tag{D.19}$$

here $\{a_k^{PG},\ b_k^{PG}\}$ are the (**PG**) prediction coefficients.

2.) The force acting on the atom i is calculated using the predicted coordinates $\{s^i_\alpha(t+\delta t)\}$:

$$\vec{F}_i = -\vec{\nabla}_i U_{pot}(r(t+\delta t)), \quad r^i_\alpha = \sum_{\beta=1}^{3} H_{\alpha\beta} s^i_\beta \tag{D.20}$$

3.) The temperature of the system and the stress tensor are evaluated at $t+\delta t$:

$$T = \frac{2}{3} \sum_{i=1}^{N} \sum_{\alpha=1}^{3} m(\dot{r}^i_\alpha)^2 \tag{D.21}$$

$$\sigma_{\alpha\beta} = \frac{1}{V} \left(\sum_{i=1}^{N} m(\sum_\gamma \dot{s}^i_\gamma H_{\gamma\alpha})(\sum_\delta H_{\beta\delta}\dot{s}^i_\delta) + \frac{1}{2} \sum_{i,j:\ i\neq j}^{N} H_{\alpha\gamma}(s^i_\delta - s^j_\delta) F_{ij,\beta} \right) \tag{D.22}$$

4.) The values of the elements of the matrices $H_{\alpha\beta}, \dot{H}_{\alpha\beta}$ are predicted:

$$H_{\alpha\beta}(t+\delta t) = H_{\alpha\beta}(t) + \dot{H}_{\alpha\beta}(t)\delta t + (\delta t)^2 \sum_{k=1}^{3} a_k^{PG} \ddot{H}_{\alpha\beta}(t+(1-k)\delta t) \tag{D.23}$$

$$\dot{H}_{\alpha\beta}(t+\delta t)\delta t = H_{\alpha\beta}(t+\delta t) - H_{\alpha\beta}(t) + \dot{H}_{\alpha\beta}(t)\delta t + (\delta t)^2 \sum_{k=1}^{3} b_k^{PG} \ddot{H}_{\alpha\beta}(t+(1-k)\delta t) \tag{D.24}$$

5.) The values of the matrix $\ddot{H}_{\alpha\beta}$ is evaluated at $t+\delta t$ using the predicted values of the matrix $H_{\alpha\beta}$ according to:

$$\ddot{H}_{\alpha\beta} = \frac{V}{Q} \sum_{\gamma=1}^{3} (\sigma_{\alpha\gamma} - P\delta_{\alpha\gamma})(H^{-1})\beta\gamma \tag{D.25}$$

6.) The acceleration of of the atom i is calculated:

$$\ddot{s}^i_\alpha(t+\delta t) = \sum_\beta (H^{-1})_{\alpha\beta} f^i_\beta - \sum_{\beta,\gamma} m(G^{-1})_{\alpha\beta}(\dot{G})_{\beta\gamma}\dot{s}^i_\gamma \tag{D.26}$$

if we take into account the *Noose-Hoover* thermostat:

$$\ddot{s}^i_\alpha(t+\delta t) = \sum_\beta (H^{-1})_{\alpha\beta} f^i_\beta - \sum_{\beta,\gamma} m(G^{-1})_{\alpha\beta}(\dot{G})_{\beta\gamma}\dot{s}^i_\gamma - \frac{1}{M_s}\psi(t+\delta t)\dot{s}^i_\alpha \tag{D.27}$$

$$\psi(t+\delta t) = \psi(t) + \frac{\delta t}{M_s N k_b T}\left[\sum_{i=1}^{N}\frac{k_b 2m(\dot{r}_i)^2}{3} - Nk_bT\right]\dot{s}^i_\alpha, \quad \psi(0)=0 \tag{D.28}$$

7.) Now the values of $H_{\alpha\beta}$ and $\dot{H}_{\alpha\beta}$ are corrected:

$$H_{\alpha\beta}(t+\delta t) = H_{\alpha\beta}(t) + \dot{H}_{\alpha\beta}(t)\delta t + (\delta t)^2\sum_{k=1}^{3} c^{PG}_k \ddot{H}_{\alpha\beta}(t+(2-k)\delta t) \tag{D.29}$$

$$\dot{H}_{\alpha\beta}(t+\delta t)\delta t = H_{\alpha\beta}(t+\delta t) - H_{\alpha\beta}(t) + \dot{H}_{\alpha\beta}(t)\delta t + (\delta t)^2\sum_{k=1}^{3} d^{PG}_k \ddot{H}_{\alpha\beta}(t+(2-k)\delta t) \tag{D.30}$$

here $\{c^{PG}_k,\ d^{PG}_k\}$ are the (**PG**) correction coefficients.

8.) Finally, the cycle is completed by correction of positions $\{s^i_\alpha(t+\delta t)\}$ and velocities $\{\dot{s}^i_\alpha(t+\delta t)\}$ of all atoms:

$$s^i_\alpha(t+\delta t) = s^i_\alpha(t) + \dot{s}^i_\alpha(t)\delta t + (\delta t)^2\sum_{k=1}^{3} c^{PG}_k \ddot{s}^i_\alpha(t+(2-k)\delta t) \tag{D.31}$$

$$\dot{s}^i_\alpha(t+\delta t)\delta t = s^i_\alpha(t+\delta t) - s^i_\alpha(t) + \dot{s}^i_\alpha(t)\delta t + (\delta t)^2\sum_{k=1}^{3} d^{PG}_k \ddot{s}^i_\alpha(t+(2-k)\delta t) \tag{D.32}$$

9.) Go to the step **1**.

The structural phase transitions can be studied using this algorithm. We used this algorithm to find the volume and the shape of the sample under the zero external pressure, close to the melting point $\mathbf{T_m}$.

Appendix E

Computer programs

Readers interested in downloading the software described in this MS Thesis should contact the home page of this Thesis at :
http://phycomp.technion.ac.il/∼phsorkin/vanadium.html
and
http://phycomp.technion.ac.il/∼phsorkin/index.html

The MD program (for surface premelting simulations) accepts an input file "initdata.h" where the size of the computational cell and all the relevant parameters (which are not changed frequently) are assigned.

```
    //************** The input control file: "initdata.h"************//
    #ifndef INITDATA_H
    #define INITDATA_H
    #define surface 0 // 0–> [001], 1–> [011], 2–>[111]


    # if surface != 1
#define size_x 10 // number of x-size atoms
#define size_y 10 // number of y-size atoms
#define Atoms 2700 // Number of atoms =size_x*size_y*size_z
#define FROZEN_ATOMS 300 // NUMBER_OF_FROZEN_LAYERS*size_x*size_y
# else //[011]
#define size_x 7 // number of size atoms
#define size_y 7 // number of size atoms
#define Atoms 2646 // Number of atoms =2*size_x*size_y*size_z
#define FROZEN_ATOMS 294 // 2*NUMBER_OF_FROZEN_LAYERS*size_x*size_y
  #endif


    #define size_z 27 // number of size atoms
    #define INIT_LAYERS size_z
    #define NUMBER_OF_FROZEN_LAYERS 3
    #define NUMBER_OF_LAYERS INIT_LAYERS-NUMBER_OF_FROZEN_LAYERS+1
    #define a00 3.0399 /* size of initail cell at T=0 K*/
    #define alpha_t 0.0000086 /*Thermal Expansion Coefficient*/
    #define dt 0.002 /*Time step*/
    #define N_neigh_step 100 /*Number of neighbour steps*/
    #define step_EPT_meas 30000
    #define step_backup 200000
    //Pressure & Temperature Control
    #define press 0.0
    // Density Profiles
    #define DELTA 20
    #define RESOLUTION DELTA*(INIT_LAYERS+5)
    //Chen_density
    # define Z_RES 500
    // Pair_Correlation Data
    # define PAIR_STEP 200
    // Order Parameter
    # define NUMBER_OF_DIRECTIONS 3
    // Diffusion Date
    # define TMAX 3000 /* number of measurements*/
    # define T0max 300 /* max number of t0 time origins*/
    # define NUMBER_OF_CALLS 10
/* time interval between set of a new t0 time origin*/

    #endif
```

An example of the input file "aaa_info.txt" where the parameters which are most frequently changed (type of surface, temperature, number of measurements, etc) are assigned. This file is generated automatically at the first run.

```
    ***************"aaa_info.txt" *****************
2
1800
1000
100
100
30000
100
100.0
6.000000
2700
100
3
24
[001]
*************** Main Parameters *****************
    Regime: 0–> start, 1 –> equilibration, 2–> measurements
    Temperature
    Number of steps in order to approach to the equilibrium
    Number of steps between two succesive measurements
    Number of measurements
    Initial step
    Number of the aviz pictures
    Percent of the accomplished job
```

```
 NVE/NVT parameter
 Number of the atoms
 Number of the atoms in a layer
 Number of frozen layers
 Number of free layers
 The surface type
 *************** " func_cp.h " *****************
```

The " func_cp.h " file describes all the functions are used by the main "md.c" program.

```
# ifndef NPTFUNC_H
# define NPTFUNC_H
void add_info();
double scale_lattice();
void aviz_layer(char *fn);
void check_initial_data();
void layer_colors(void);
void comments(char *fn);
long get_inform(char *fn);
void save_inform(char *fn);
void backup(char *fn, int mode);
void save_configuration(int mode);
 void equilibration();
 void measurements(void);
 void measurements_initialize_n_backup(int mode);
 void measurements_make_n_output(int mode);
//  Trajectories
void init_trajectories(int mode);
 void trajectories();
// Layer distribution
 void energy_temp_pressure();
 void atom_distribution_in_layers (int mode);
 void atom_distribution_in_layers_rw (char *fn, int mode);
//Diffusion
void diffusion_rw(char *fn,int mode);
void find_diffusion_times(void);
void diffusion(int mode);
void pbc_rw(char *fn,int mode);
// Pair Correlation Functions
void pair_correlation(int mode);
void pair_correlation_rw(char *fn, int mode );
// Inter Layer Distances
void inter_layer_dist(int mode);
void inter_layer_dist_rw(char *fn,int mode);
// Order Profile Functions
 void order_parameters(int mode);
 void order_parameters_rw(char *fn, int mode );
// Temperature Profile Functions
 void temperature_profile (int mode);
 void temperature_profile_rw(char *fn, int mode);
// Density Profile Functions
 void density_profile(int mode);
 void density_profile_rw(char *fn, int mode);
 void chen_density_profile(int mode);
 void chen_density_profile_rw(char *fn, int mode);
//Initalization
void init_r_init(void);
void init(void);
void  aviz(char *fn);
//Setup
void bcc_011(void) ;
void bcc_001(void);
void velocity_init(void);
double lattice_parameter(double x);
void    setup_111();
void    create_001();
//Input Output
void read_coord_from_file(char *);
void read_vel_from_file(char*);
void save_to_file(long,int);
// Basis
void build_neigh_list(void);
void predictor(void);
void corrector(void);
void compute_f(void );
void eval_PTO(void);
// Density part of potential
double density(double pr_dis);
double density_dot(double pr_dis);
double density_dot_dot(double pr_dis);
double density_cut(double r);
double density_dot_cut(double r);
// Phi potential
double phi_pot(double pr_dis);
double phi_pot_dot(double pr_dis);
double phi_pot_dot_dot(double pr_dis);
double phi_pot_cut(double r);
double phi_pot_dot_cut(double r);
double phi_pot_dot_ren(double x);
double phi_pot_dot_dot_ren(double x);
//F_potential
double F_pot(double);
double F_pot_dot(double);
double F_pot_dot_dot(double x);
double  max3_plus(double R1,double R2,double R3,double M);
#endif
```

*************** "potential.h" *****************

```
// All the parameters of the modified FS potential for vanadium are collected in this file
# ifndef  POTENTIAL_H
# define  POTENTIAL_H
```

```
    #define cutoff 3.8              /*Cutoff of potential*/
    #define neicutoff 14.440000   /*Square of the nearest neighbour distance */
    # define FS_d 3.692767
    # define r_cut 3.8
    # define c1 -0.8816318
    # define c2 1.4907756
    # define c3 -0.3976370
    # define A -2.010637
    # define FND 2.63185120210091
    # define B 23.0
    # define alp 0.5
    # define b0 2.632
    #endif
     *************** "constants.h" *****************
```

The various constants (π,k_B) are defined here

```
# ifndef CONSTANT_H
    # define CONSTANT_H
    #define pi 3.14159265358979323846264338327950
    #define kb 0.0000862 /* Boltzman constant */
    #endif
     *************** "md.c" *****************
```

The main program "md.c" which calls other functions.

```
    # include <stdio.h>
    # include <stdlib.h>
    # include <math.h>
    # include "func_cp.h"
    # include "potential.h"
    # include "initdata.h"
    # include "constants.h"
    int Color[Atoms ];
    double r[Atoms][3];
    double v[Atoms][3];
    double f[Atoms][3];
    int nei[Atoms][Atoms];
    // Predictor-Corrector
    double f1[Atoms][3];
    double f2[Atoms][3];
    double r1[Atoms][3];
    double v1[Atoms][3];
    //General properties
    double a0,Q,temp,ksi=0;
    double temp_current,press_current;
    double pot_E,kin_E,virsum;
    double lx,ly,lz,Width;
    double pten[3][3];
    int regime;
    //Layer structure
     int atoms_distribution[NUMBER_OF_LAYERS];
     long    counter_of_atoms_distribution;
     long    step0,step,num_eq_steps,num_of_meas;
     int     interval,Limit_aviz_pict,number_aviz_pict=0,step_Aviz;
```

```
 int      layers[NUMBER_OF_LAYERS][Atoms ];
// Pair Correlation Function
double     Profile[NUMBER_OF_LAYERS][PAIR_STEP],width_rdf,
normFac,half_box;
long       pair_corr_count;
// Order parameter profile
double     S[NUMBER_OF_LAYERS][NUMBER_OF_DIRECTIONS][2];
// Chen Density profile
int        z_profile[RESOLUTION];
long       count_density_profile ;
double dens_ch[Z_RES],temp_ch[Z_RES], chen_counter;
 //Diffusion
 long       diff_counter,tmax,t0max;
 int        ntime[TMAX],time0[T0max],t0=0;
 double     r0[Atoms][3][T0max];
 double     r2f[TMAX][3][NUMBER_OF_LAYERS];
 int        pbc[Atoms][2];
//Temperature profile
double dis_bet_lay[NUMBER_OF_LAYERS], counter_dist;
double temp_profile[NUMBER_OF_LAYERS], counter_temp;
int  main()
{
int i,j,dd;
FILE* outd;
char fn[50];
check_initial_data();
get_inform("aaa_info.txt");
// Take thermal expansion into account
a0=lattice_parameter(temp);
if(surface==0)//==== Structure b.c.c. [001] ==
{
bcc_001() ;
Width=a0/2;
lx=a0*size_x;            // a0  Angstrems  lattice constants
ly=a0*size_y;            // lx, ly length of the box  Angstrems  lattice constants
 } //=================
if(surface==1)
{
Width=a0/sqrt(2.);                  // Distance between layers in ordered crystal
lx=sqrt(2.)*a0*size_x;          // a0  Angstrems  lattice constants
ly=a0*size_y;                       // lx, ly length of the box  Angstrems  lattice constants
bcc_011() ;
}
//================
 if(surface==2)
{
Width=a0/(2*sqrt(3.));
lx=sqrt(2.)*a0*size_x;               // a0  Angstrems  lattice constants
ly=sqrt(3./2.)*a0*size_y;            // lx, ly length of the box  Angstrems  lattice constants
read_coord_from_file("r00.log");
scale_lattice();
}
//=========================//
lz=Width*(INIT_LAYERS-NUMBER_OF_FROZEN_LAYERS);
layer_colors();
aviz("van_0.xyz");
add_info();
//====================//
 step=step0;
 init();
if(regime){  read_coord_from_file("r.log");  read_vel_from_file("v.log");}
if(regime < 2)
{
if(regime==0)
{
velocity_init();
aviz("van_0.xyz");
regime=1;
}
else   fprintf(stderr,"\n Continue approaching to the equilibrium");
 //Continue simulation without the structure measurements
equilibration();
step0=step;
regime=2;
}

if(regime==2)
{
fprintf(stderr,"\n Start measurements");
measurements();
}
fprintf(stderr,"\n\t ********* The Happy End ***************\n");
return(0);

}  *************** "initialization.c" *****************
```

Set to zero all the parameters of the system

```
# include <stdio.h>
      # include <stdlib.h>
      # include "initdata.h"
      void init(void)
      {
      extern double f1[Atoms][3];
      extern double f2[Atoms][3];
      extern double r1[Atoms][3];
      extern double v1[Atoms][3];
      extern int  z_profile[RESOLUTION];
      int i,j;
            for(i=0;i<Atoms;i++)
                 for(j=0;j<3;j++)
      r1[i][j]=v1[i][j]=f1[i][j]=f2[i][j]=0.;
      for(i=0;i<RESOLUTION;i++)
                    z_profile[i]=0;
      }
      ////////////////////////////////////////////////////////////
      void pbc_rw(char *fn,int mode)
      {
      extern  int pbc[Atoms][2];
      extern double r[Atoms][3];
      FILE *fd;
      int i;
      if(mode==0)
      {
      ////////////////////////////////////////
      if((fd=fopen(fn,"r"))!=NULL)
      {
      for (i=FROZEN_ATOMS;i<Atoms;i++)
      fscanf(fd," \n %d %d ",&pbc[i][0],&pbc[i][1]);
      fclose(fd);
      }
      else
      {
      fprintf(stderr,"\n ! pbc initialization  (:) ");
      system("rm -rf diffusion.back  atom*.txt ");
      if((fd=fopen("r0.log","w"))!=NULL)
         {
           for (i=0;i<Atoms;i++)
                        {
                        fprintf(fd,"\n %lf  %lf  %lf", r[i][0],r[i][1],r[i][2]);
                        pbc[i][0]=pbc[i][1]=0;//Check & keep p.b.c. for each atom
                        }
           fclose(fd);
        }
      }
      ///////////////////////////////////////////
      }
      if(mode==1)
      {
      ////////////////////////////////////////
       if((fd=fopen(fn,"w"))!=NULL)
      {
       for (i=FROZEN_ATOMS;i<Atoms;i++)
       fprintf(fd," \n %d %d ",pbc[i][0],pbc[i][1]);
        fclose(fd);
      }
      }
      }
       *************** "equilibration.c" *****************
```

Equilibration of the system

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include "func_cp.h"
# include ”potential.h”
# include ”initdata.h”
# include ”constants.h”
void equilibration()
{
 extern  int number_aviz_pict,Limit_aviz_pict,step_Aviz;
 extern  long step,num_eq_steps;
 extern  double temp_current,press_current,kin_E,pot_E;
 int i,j,eq_steps;
 char fn[50];
 eq_steps=(int)1.0*num_eq_steps/N_neigh_step;
 build_neigh_list();
 compute_f();
//Launching  calculations
for(i=0;i<eq_steps;i++){
for(j=0;j<N_neigh_step;j++)
{
step++;
predictor();
compute_f();
eval_PTO();
corrector();
}
build_neigh_list();
fprintf(stderr,”\n step=%ld temp=%g pressure=%g
total_E=%g ”,step,temp_current,press_current,(kin_E+pot_E));
if(step % step_EPT_meas==0) energy_temp_pressure();
if(step % step_Aviz==0)
{
if(number_aviz_pict++ < Limit_aviz_pict)
 {  sprintf(fn,”van_%ld.xyz”,step);
       aviz(fn);
 }
}
if(step % step_backup==0)backup(”aaa_info.txt”,1);
} // The Main Cycle: over i
backup(”aaa_info.txt”,1);
}
```

*************** “measurements.c” *****************

Measurement of the various physical properties

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include ”func_cp.h”
# include ”potential.h”
# include ”initdata.h”
# include ”constants.h”
void measurements()
{
 extern long  step,num_of_meas,step_Aviz;
 extern int      interval,Limit_aviz_pict,number_aviz_pict;
 extern double temp_current,press_current;
 extern double pot_E,kin_E;
  long   number_of_steps;
  int  i, j, write_intermediate_results;
  char fn[50];
 write_intermediate_results=(int)1.0*num_of_meas*interval/4.0;
 number_of_steps=(long)1.*num_of_meas*interval/N_neigh_step;
measurements_initialize_n_backup(0); // Initialize  or  read some functions:
//  Starting calculations
 build_neigh_list();
 compute_f();

for(i=0;i<number_of_steps;i++){
//====================================//
for(j=0;j<N_neigh_step;j++)
{
step++;
predictor();
compute_f();
eval_PTO();
corrector();
   }
build_neigh_list();
fprintf(stderr,"\n step=%ld temp=%g pressure=%g
total_E=%g ",step,temp_current,press_current,(kin_E+pot_E));
  if(step % interval==0)
{
  if(step % write_intermediate_results) measurements_make_n_output(1);
  else measurements_make_n_output(2) ;
 }
 if(step % step_Aviz==0)
{
if(number_aviz_pict++ < Limit_aviz_pict)
 {
        sprintf(fn,"van_%ld.xyz",step);
 aviz(fn);  aviz_layer(fn);
 }
}
if(step % step_backup==0)  measurements_initialize_n_backup(1);
//========================//
}        // Main cycle
 // Final measurements:
 measurements_make_n_output(2) ;
  // Final backup:
 measurements_initialize_n_backup(1);
}
 ///////////////////////
 void measurements_initialize_n_backup(int mode)
{
   backup("aaa_info.txt",mode);
```

```
    density_profile_rw("dens_prof.back",mode);
     chen_density_profile_rw("chen_dens_prof.back",mode);
     atom_distribution_in_layers_rw ("at_dist_prof.back",mode);
    temperature_profile_rw("temp_prof.back",mode);
     order_parameters_rw("order_prof.back",mode);
    inter_layer_dist_rw("inter_layer_dist.back",mode);
     pair_correlation_rw("pair_func.back",mode);
    diffusion_rw("diffusion.back",mode);
    pbc_rw("pbc.back",mode);
    }
  //////////////////////////////
   void measurements_make_n_output(int mode)
  {
   energy_temp_pressure();
   chen_density_profile(mode);
   atom_distribution_in_layers(mode);
   temperature_profile(mode);
   order_parameters(mode);
   inter_layer_dist(mode);
   pair_correlation(mode);
   diffusion(mode);
   }
   *************** "bcc_setup.c" *****************
```

How to construct the various low index faces of a bcc crystal and initialize the velocities

```
# include <stdio.h>
  # include <math.h>
  #include <stdlib.h>
  #include <time.h>
  # include "initdata.h"
  # include "constants.h"
  # include "potential.h"

  void velocity_init(void){
  extern double v[Atoms][3],temp;
  int j,k;
  double sum[3];// Initial Temperature
  double x1,x2,x3,y1,y2,y3,Sf;
  for(j=0;j<FROZEN_ATOMS;j++)
  { v[j][0]=0.; v[j][1]=0.; v[j][2]=0.;  }
    for (j=0;j<3;j++)
        sum[j]=0.;
     srand( (unsigned)time( NULL ) );
         /* Seed the random-number generator with current
time so that * the numbers will be different every time we run    */
    /* Generate gaussian distribution */
         for (j=FROZEN_ATOMS;j<Atoms;j++)
         {
      x1=((double)rand())/RAND_MAX;
      x2=((double)rand())/RAND_MAX;
      x3=((double)rand())/RAND_MAX;
```

```
         /*The Box Muller method */
    y1=sqrt(-log(x1))*cos(2*pi*x2);
    y2=sqrt(-log(x2))*cos(2*pi*x1);
    y3=sqrt(-log(x3))*cos(2*pi*x1);
   v[j][0]=y1;
   v[j][1]=y2;
   v[j][2]=y3;
   for (k=0;k<3;k++)sum[k]+=v[j][k];
        }
  /* Center of mass velocity */
         for (k=0;k<3;k++)
  sum[k]/=(Atoms-FROZEN_ATOMS);
  for (j=FROZEN_ATOMS;j<Atoms;j++)
          for (k=0;k<3;k++)
      v[j][k]-=sum[k];
  Sf=0.;
  for (j=FROZEN_ATOMS;j<Atoms;j++)
  Sf+=v[j][0]*v[j][0]+v[j][1]*v[j][1]+v[j][2]*v[j][2];
        /* Let's rescale the velocities */
  Sf=sqrt(3*kb*temp*(Atoms-FROZEN_ATOMS)/Sf);
  for (j=FROZEN_ATOMS;j<Atoms;j++)
          for (k=0;k<3;k++)
          v[j][k]*=Sf;
  /*Now check it and be sure, that T appropriate and Vcm=0 */
    Sf=0;
    for (j=0;j<3;j++)
    sum[j]=0.;
      for (j=FROZEN_ATOMS;j<Atoms;j++){
      Sf+=v[j][0]*v[j][0]+v[j][1]*v[j][1]+v[j][2]*v[j][2];
      for (k=0;k<3;k++){ sum[k]=sum[k]+v[j][k];}
      }
        fprintf(stdout,"\n Center_of_Mass
\n Vx=%lf Vy=%lf  Vz=%lf ",sum[0],sum[1],sum[2]);
      fprintf(stdout,"\n temp=%lf ", Sf/(3*(Atoms-FROZEN_ATOMS)*kb));
   // for (j=0;j<Atoms;j++)
   //fprintf(stderr,"\n j=%ld %lf  %lf  %lf",j,v[j][0],v[j][1],v[j][2]);
}
 ///////////////////////////////////////////////
void bcc_001(void){
extern double r[Atoms][3];
extern double a0;
int i,j,k,pa;
double a02;//Initial coordinates
FILE *fd;

a02=a0/2;
pa=0;
for (k=0;k<INIT_LAYERS;k++){
        for (j=0;j< size_x;j++){
      for(i=0;i<size_y;i++){
              r[pa][0]=a0*i;
```

```
                r[pa][1]=a0*j;
                r[pa][2]=a02*k;
                if(k%2!=0){
                r[pa][0]+=a02;
                r[pa][1]+=a02;}
                        pa++;
                }
          }
    }
    if((fd=fopen("r0.log","w"))!=NULL)
    {
    for (j=0;j<Atoms;j++)
    fprintf(fd,"\n %lf  %lf  %lf",
r[j][0],r[j][1],r[j][2]);
    fclose(fd);
     }
    else
    {fprintf(stderr,"\n I can't read file:\"
r0.log \"");exit(0);}
     }
    ///////////////////////////
    void bcc_011(void)
    {
    extern double r[Atoms][3];
    extern double a0;
    int i,j,k,n,index;
    FILE *fd;
    double E1[3][1],E2[3][1],E3[3][1],E4[3][1];
    // The Unit  vectors of the cell
    E1[0][0]=a0*sqrt(2);E1[0][1]=E1[0][2]=0;
//E1=[sqrt(2);0;0];
    E2[0][0]=E2[0][1]=a0/sqrt(2);E2[0][2]=0.;
//E2=[1./sqrt(2);1/sqrt(2);0];
    E3[0][0]=E3[0][1]=0.;E3[0][2]=a0;
//E3=[0;0;1.0];
    E4[0][0]=a0*sqrt(2)/2;E4[0][1]=0.;E4[0][2]=a0*0.5;
//E4=[sqrt(2)/2;0;1/2];
    index=0;
    for (k=0;k<size_y;k++){
    for (j=0;j<size_z;j++){
    for (i=0;i<size_x;i++){
    n=i+j*size_x+k*size_x*size_y;
    r[index][2]=E1[1][0]*i+j*E2[1][0];
    r[index][1]=E3[2][0]*k;
    r[index][0]=E1[0][0]*i+E2[0][0];
     // Set the center atoms for odd  lines
    if(j%2)  r[index][0]+=E2[0][0];
    index++;
    }//i
    }//j
    }//k
```

```
/////////// ADD CENTRAL ATOM ///////////////
for (k=0;k<size_y;k++){
for (j=0;j<size_z;j++){
for (i=0;i<size_x;i++){
r[index][2]=E4[1][0]*i+j*E2[1][0];
r[index][1]=E3[2][0]*k+E4[2][0];
r[index][0]=E1[0][0]*i+2*E4[0][0];
if(j%2) r[index][0]+=E4[0][0] ;
 index++;
n++;
}//i
}//j
}//k
////////////////////////////
   if((fd=fopen("r0.log","w"))!=NULL)
{
for (j=0;j<Atoms;j++)
fprintf(fd,"\n %lf  %lf  %lf",
r[j][0],r[j][1],r[j][2]);
fclose(fd);
 }
else  {fprintf(stderr,"\n I can't read file:\"
r0.log \"");exit(0);}  }
//////////////////////////////////////////
void layer_colors(void)
{
extern double r[Atoms][3],Width,a0;
extern int Color[Atoms ]; double live[Atoms-FROZEN_ATOMS ][3], frozen[FROZEN_ATOMS ][3];
int i,j,cold=0,hot=0;
double h,barrier;
barrier=Width*(NUMBER_OF_FROZEN_LAYERS-1)+0.001;
 for ( i=0;i<Atoms;i++)
{
     h=r[i][2]-barrier;
     if(h<=0)
{
frozen[cold][0]=r[i][0];frozen[cold][1]=r[i][1]; frozen[cold][2]=r[i][2];cold++;
//fprintf(stderr,"\n cold= %d Atoms=%d h=%g ", cold,Atoms,h);
}
 else
{
live[hot][0]=r[i][0];live[hot][1]=r[i][1]; live[hot][2]=r[i][2];
hot++;
//fprintf(stderr,"\n hot= %d i=%d", hot,i);
}
}
// Chek post
if (cold-FROZEN_ATOMS)
{
fprintf(stderr, "That strange somehow !!!! >> cold  != FROZEN_ATOMS");
 fprintf(stderr, "  \n \t cold =%d",cold);
 fprintf(stderr, " \n \t FROZEN_ATOMS=%d",FROZEN_ATOMS);
  exit(0);
}
if (hot+FROZEN_ATOMS-Atoms)
{fprintf(stderr, "That strange somehow !!!! >>hot != FROZEN_ATOMS-Atoms "); exit(0);}

//Rearrange the atoms:
for (i=0;i<cold;i++)
{ r[i][0]=frozen[i][0]; r[i][1]=frozen[i][1];r[i][2]=frozen[i][2];}
for (i=0;i<hot;i++)
{  j=i+FROZEN_ATOMS;r[j][0]=live[i][0]; r[j][1]=live[i][1];r[j][2]=live[i][2];}

//Redefine Colors again:
 for ( i=0;i<Atoms;i++)
{
    h=r[i][2]-Width/2.0;
    if(h<0) Color[i] =0;
    else     Color[i] =(int) h/Width+1;
 }
 //fprintf(stderr, "\n \n \t Bye !!!! ");
 }
 ////////////////////////////////////////
void check_initial_data()
{
switch(surface)
{
case 0:    break;
case 1:    break;
case 2:    break;
default:
printf("This is an error, the parameter could be  surface [0,3] only ");exit(0);
break;
}
///////////////////////////////////////////
 if(surface==1){
  // Let's check the initial conditions:
if(  2*size_x*size_y*size_z-Atoms)
{
fprintf(stderr, "\n Attention : 2*size_x*size_y*size_z-Atoms
!=0 , \n Check the \" initdata.h
\" file ");
fprintf(stderr, "\n  Write  \"#define
Atoms  %d \" ",2*size_x*size_y*size_z);

exit(0);}
if(  2*size_x*size_y*NUMBER_OF_FROZEN_LAYERS-FROZEN_ATOMS  ) {
fprintf(stderr, "\n Attention : 2*size_x*size_y*NUMBER_OF_FROZEN_LAYERS-FROZEN_ATOMS
!= 0 , \n Check the \" initdata.h
\" file ");
```

```
 fprintf(stderr, "\n Write \"
#define FROZEN_ATOMS %d \" ",

2*size_x*size_y*NUMBER_OF_FROZEN_LAYERS);
exit(0);
}
}
////////////////////////
        if(surface==2 || surface==0 ){
if( size_x*size_y*size_z-Atoms)
{
fprintf(stderr, "\n Attention : size_x*size_y*size_z-Atoms
!=0 , \n Check the \" initdata.h
\" file ");
fprintf(stderr, "\n Write \"#define
Atoms %d \" ",size_x*size_y*size_z);

exit(0);}

if( size_x*size_y*NUMBER_OF_FROZEN_LAYERS-FROZEN_ATOMS )
{
fprintf(stderr, "\n Attention :
size_x*size_y*NUMBER_OF_FROZEN_LAYERS-FROZEN_ATOMS
!= 0 , \n Check the \" initdata.h
\" file ");

 fprintf(stderr, "\n Write \"
#define FROZEN_ATOMS %d \" ",
size_x*size_y*NUMBER_OF_FROZEN_LAYERS);
exit(0);
}
}
}
//////////////////////////////
double lattice_parameter(double x)
{
int i;
double z;
// The Bulk simulations:
double lattice_set[10]=
{3.0482,3.0626,3.0706,3.073,3.089,3.092,3.095,3.096,3.096,3.18};
double temp_set[10]={400,1200, 1600,1700,2300,2400,2450,2480,2480,2500};
  z= a00*exp(alpha_t*x);
  if(x < temp_set[0]){return(z);}
  if(x > temp_set[9]){return(z);}
  if(x == temp_set[0]){return(lattice_set[0]);}
  i=1;
  while(x>temp_set[i])i++;
  // The simplest linear interpolation:
  z=(lattice_set[i]*(x-temp_set[i-1])+lattice_set[i-1]*(temp_set[i]-x))/
  (temp_set[i]-temp_set[i-1]);
  return(z);
 }
////////////////////////// double scale_lattice() {
extern double a0;
extern double r[Atoms][3];
FILE *fd;
int i;
for (i=0;i<Atoms;i++)
{r[i][0]*=a0;r[i][1]*=a0;r[i][2]*=a0;}
if((fd=fopen("r0.log","w"))!=NULL)
{
for( i=0;i<Atoms;i++)
fprintf(fd," \n
%lf %lf %lf",r[i][0],r[i][1],r[i][2]);
fclose(fd);
}
}  ***************"build_neigh_list.c.c" *****************
```

Neighbor-list method - keep the list of the nearest neighbors

```
# include "constants.h"
# include "initdata.h"
# include "potential.h"
# include <math.h>
void build_neigh_list()
{
extern int nei[Atoms][Atoms];
extern double r[Atoms][3];
extern double lx,ly;
 int i,j,k;
  double dis[3]={0.,0.,0.};
  double pr_dis;
  for(i=0;i<Atoms;i++)
  for(j=0;j<Atoms;j++)
   nei[i][j]=0;
  for(i=0;i<Atoms;i++){
  for(j=i+1;j<Atoms;j++){
  pr_dis=0.;
 // x&y periodic boundary conditions
   dis[0]=r[i][0]-r[j][0];
   if(fabs(dis[0])> lx/2.0)
  {
   if(dis[0] > 0.)dis[0]-=lx;
   else dis[0]+=lx;
   }
   dis[1]=r[i][1]-r[j][1];
   if(fabs(dis[1])> ly/2.0)
   {
   if(dis[1]>0.)dis[1]-=ly;
   else dis[1]+=ly;
   }
  dis[2]=r[i][2]-r[j][2];
  pr_dis=sqrt(dis[0]*dis[0]+dis[1]*dis[1]+dis[2]*dis[2]);
//....................................................
  if(pr_dis<neicutoff)
  {
   nei[i][0]++;
    nei[i][nei[i][0]]=j;
  }
     } // for(i=0;i<Atoms;i++)
     }  // for(j=i+1;j<Atoms;j++)
  }
 ***************"compute_f.c" *****************
```

Calculation of force and potential energy per atom (with the modified FS potential) # include <math.h>

```
#include "initdata.h"
# include "func_cp.h"
//Modification of FS by Rebonato and et.al
#define Reb_FND    2.6319
#define Reb_K      3.3
#pragma_CRI inline density,density_dot,density_dot_dot,phi_pot,phi_pot_dot,
phi_pot_dot_dot,F_pot,F_pot_dot,F_pot_dot_dot
/* vectorization of compute_f,build_neigh_list , parallelization of compute_f */
////////////////////////////////////////////////
void compute_f()
{
extern double r[Atoms][3],f[Atoms][3];
 extern int regime,nei[Atoms][Atoms];
 extern double pot_E,virsum,lx,ly;
 extern double op;
int i,j,k,ll,mm;
double phi,dphi;
double dphi_ren,ddphi_ren;
double pr_dis,fr,fr1,fr2;
double rho,drho,ddrho;
double ui,dui,duj,ddui;
double dis[3],n[Atoms],Atoms];
/*Vectorization: A new part !!!! */
double nj[Atoms],f_tmp[Atoms][3],ni,fi0,fi1,fi2,pot_E_tmp,virsum_tmp;
pot_E=0.;
virsum=0.;
  for(i=0;i<Atoms;i++)
  for(ll=0;ll<3;ll++) f[i][ll]=0.0;
 for(i=0;i<Atoms;i++){
   n[i]=0.0;
 }
#pragma _CRI parallel shared(g11,g22,g23,n) defaults
 for(j=0;j<Atoms;j++) {
   nj[j]=0.0;
 }
/* The first loop: the background n(i) calculation */
#pragma _CRI taskloop
 for( i=0;i<Atoms;i++){
   ni=0.0;
#pragma _CRI ivdep
```

```
   for( k=1;k<=nei[i][0];k++){
     j=nei[i][k];
     dis[0]=r[i][0]-r[j][0];
     if (dis[0] > lx/2.0 ) dis[0]-=lx;
     if (dis[0] < -lx/2.0) dis[0]+=lx;
     dis[1]=r[i][1]-r[j][1];
     if (dis[1] > ly/2.0 ) dis[1]-=ly;
     if (dis[1] < -ly/2.0) dis[1]+=ly;
     dis[2]=r[i][2]-r[j][2];
    pr_dis=dis[0]*dis[0]+dis[1]*dis[1]+dis[2]*dis[2] ;
    pr_dis=sqrt(pr_dis);
     if(pr_dis<cutoff)
       {
         rho=density(pr_dis);
         //cj moved to the if regime block !  drho=density_dot(pr_dis);
         //cj n[i]+=rho;
         ni+=rho;
         //cj  n[j]+=rho;
         nj[j]+=rho;
       } // cj end of cutoff
   } //cj end of k loop
   // n[i]+=ni;
   n[i]=ni;
 } //cj end of i loop
#pragma _CRI guard
 for(j=0;j<Atoms;j++) {
   n[j]+=nj[j];
   if(regime==3){
     g11[j]+=g11_tmp[j];
     g22[j]+=g22_tmp[j];
     g23[j]+=g23_tmp[j];
   }
 }
#pragma _CRI endguard
#pragma _CRI endparallel
//THE SECOND LOOP
#pragma _CRI parallel shared(g11,g22,g23,n) defaults
 pot_E_tmp=0.0;
 virsum_tmp=0.0;
 for(j=0;j<Atoms;j++)
 for(ll=0;ll<3;ll++) f_tmp[j][ll]=0.0;
#pragma _CRI taskloop
 for( i=0;i<Atoms;i++){
   fi0=fi1=fi2=0.0;

   ui=F_pot(n[i]);
   dui=F_pot_dot(n[i]);
   ddui=F_pot_dot_dot(n[i]);
   pot_E_tmp+=ui;
#pragma _CRI ivdep
   for( k=1;k<=nei[i][0];k++){
        j=nei[i][k];

        dis[0]=r[i][0]-r[j][0];
        if (dis[0] > lx/2.0 ) dis[0]-=lx;
        if (dis[0] < -lx/2.0) dis[0]+=lx;

        dis[1]=r[i][1]-r[j][1];
        if (dis[1] > ly/2.0 ) dis[1]-=ly;
        if (dis[1] < -ly/2.0) dis[1]+=ly;
        dis[2]=r[i][2]-r[j][2];
        pr_dis=dis[0]*dis[0]+dis[1]*dis[1]+dis[2]*dis[2] ;
        pr_dis=sqrt(pr_dis);

        if(pr_dis < cutoff)
          {
            phi=phi_pot(pr_dis);
            dphi=phi_pot_dot(pr_dis);
            drho=density_dot(pr_dis);
            if(regime==3)
      {
          dphi_ren=phi_pot_dot_ren(pr_dis);
              ddphi_ren=phi_pot_dot_dot_ren(pr_dis);
         ddrho=density_dot_dot(pr_dis);
            }
            pot_E_tmp+=phi;
            duj=F_pot_dot(n[j]);
            fr1=-dphi;
            fr2=-drho*(dui+duj);
            fr=fr1+fr2;
            virsum_tmp+=fr*pr_dis;
            for (ll=0;ll<3;ll++)
              {
                //cj fi0,fi1,fi2 will be used instead of
f[i][ll]+=fr*dis[ll]/pr_dis;
                f_tmp[j][ll]-=fr*dis[ll]/pr_dis;
              }
            fi0+=fr*dis[0]/pr_dis;
            fi1+=fr*dis[1]/pr_dis;
            fi2+=fr*dis[2]/pr_dis;
          }   //cutoff loop
       } //nei loop
      /*cj add fi0,fi1,fi2 to the force field */
       f[i][0]=fi0;
       f[i][1]=fi1;
       f[i][2]=fi2;
     } //i Atoms loop
    #pragma _CRI guard
     for(j=0;j<Atoms;j++)
     for(ll=0;ll<3;ll++) f[j][ll]+=f_tmp[j][ll];
       pot_E+=pot_E_tmp;
     virsum+=virsum_tmp;
```

```
#pragma _CRI endguard
#pragma _CRI endparallel
}
////////////////////////////////////////
double F_pot(double x)
{
return(A*sqrt(x));
}
//////////////////////////////////////////
double F_pot_dot(double x)
{
if(fabs(x)<0.0000000000001)
x=0.0000000000001;
return(0.5*A/sqrt(x));
}
//////////////////////////////////////
double phi_pot(double x){
double y;

if(x>r_cut) return(0);
y=(x-r_cut)*(x-r_cut)*(c1+c2*x+c3*x*x);
// FS+ Rebonato et al
if(x<Reb_FND)y+=Reb_K*(Reb_FND-x)*(Reb_FND-x)*(Reb_FND-x);
return(y);
}
////////////////////////////////////////////// double phi_pot_dot(double x){
double y;
if(x>r_cut)return(0);
y=2*(x-r_cut)*(c1+c2*x+c3*x*x)+(x-r_cut)*(x-r_cut)*(c2+2*c3*x);
// FS+ Rebonato et al
if(x<Reb_FND)y+=-3*Reb_K*(Reb_FND-x)*(Reb_FND-x);
 return(y);
}
///////////////////////////////////////
double density(double pr_dis)
{
double rho;
if(pr_dis>FS_d)
return(0);
else
rho=(pr_dis-FS_d)*(pr_dis-FS_d);
return(rho);}
///////////////////////////////////////
double density_dot(double pr_dis)
{
double rho;
if(pr_dis>FS_d)
return(0);
else
rho=2*(pr_dis-FS_d);
return(rho);
```

```
}
 *************** “predictor_corrector.c ” *****************
```

Use the predictor corrector to solve the equations of motion # include ”initdata.h”

```
# include <stdio.h>
void corrector()
{
extern double r[Atoms][3],v[Atoms][3],f[Atoms][3];
extern double f1[Atoms][3],f2[Atoms][3],r1[Atoms][3],v1[Atoms][3];
extern double lx,ly;
extern int pbc[Atoms][2],regime;
double cr[]={3.0,10.,-1.};
double cv[]={7.,6.,-1.};
double div=24.;

int i;
for( i=FROZEN_ATOMS;i<Atoms;i++){
r[i][0]=r1[i][0]+v1[i][0]*dt+(dt*dt/div)*(cr[0]*f[i][0]+cr[1]*f1[i][0]+
cr[2]*f2[i][0]);
r[i][1]=r1[i][1]+v1[i][1]*dt+(dt*dt/div)*(cr[0]*f[i][1]+cr[1]*f1[i][1]+
cr[2]*f2[i][1]);
r[i][2]=r1[i][2]+v1[i][2]*dt+(dt*dt/div)*(cr[0]*f[i][2]+cr[1]*f1[i][2]+
cr[2]*f2[i][2]);

v[i][0]=(r[i][0]-r1[i][0])/dt+(dt/div)*(cv[0]*f[i][0]+cv[1]*f1[i][0]+
cv[2]*f2[i][0]);
v[i][1]=(r[i][1]-r1[i][1])/dt+(dt/div)*(cv[0]*f[i][1]+cv[1]*f1[i][1]+
cv[2]*f2[i][1]);
v[i][2]=(r[i][2]-r1[i][2])/dt+(dt/div)*(cv[0]*f[i][2]+cv[1]*f1[i][2]+
cv[2]*f2[i][2]);

// Periodic Boundary Conditions for xy coordinates:
if(regime>1)
{
if(r[i][0]>lx){r[i][0]-=lx;pbc[i][0]++;}
if(r[i][1]>ly){r[i][1]-=ly;pbc[i][1]++;}

if(r[i][0]<0.){r[i][0]+=lx;pbc[i][0]=pbc[i][0]-1;}
if(r[i][1]<0.){r[i][1]+=ly;pbc[i][0]=pbc[i][0]-1; }
}
else
 {
if(r[i][0]>lx){r[i][0]-=lx;}
if(r[i][1]>ly){r[i][1]-=ly;}

if(r[i][0]<0.){r[i][0]+=lx;}
if(r[i][1]<0.){r[i][1]+=ly;}
}
                    }
}
//==================================//
void predictor()
```

```
{
extern double r[Atoms][3],v[Atoms][3],f[Atoms][3];
extern double f1[Atoms][3],f2[Atoms][3],r1[Atoms][3],v1[Atoms][3];
double cr[]={19.,-10.,3.};
double cv[]={27.,-22.,7.};
double div=24.;

int i,k;
for( i=FROZEN_ATOMS;i<Atoms;i++){
for( k=0;k<3;k++){

r1[i][k]=r[i][k];
v1[i][k]=v[i][k];
r[i][k]+=v[i][k]*dt+(dt*dt/div)*(cr[0]*f[i][k]+cr[1]*f1[i][k]+
cr[2]*f2[i][k]);
v[i][k]=(r[i][k]-r1[i][k])/dt+(dt/div)*(cv[0]*f[i][k]+cv[1]*f1[i][k]+
cv[2]*f2[i][k]);

f2[i][k]=f1[i][k];
f1[i][k]=f[i][k];
 }
                    }
}
```

*************** "evap_pto.c " *****************

Evaluate the current pressure and temperature. Use the Noose-Hoover thermostat.

```
# include <math.h>
# include "initdata.h"
# include "constants.h"
void eval_PTO()
{
extern double ksi,lx,ly,lz,Q;
extern double f[Atoms][3],v[Atoms][3],r[Atoms][3];
extern double kin_E;
extern double temp,temp_current,press_current,virsum;
  int i,k;
  double vvsum,vol;
  vol=lx*ly*lz;
  vvsum=0.0;
 for(i=FROZEN_ATOMS;i<Atoms;i++)
vvsum+=(v[i][0]*v[i][0]+v[i][1]*v[i][1]+v[i][2]*v[i][2]);
  kin_E=1.0*vvsum/2.;
  temp_current=vvsum/(3.0*kb*(Atoms-FROZEN_ATOMS));
  press_current=(temp_current*(Atoms-FROZEN_ATOMS)*kb+virsum/3.)/vol;
      ksi+=dt*Q*((temp_current/temp)-1.0);
       for(i=FROZEN_ATOMS;i<Atoms;i++)
       for(k=0;k<3;k++)
       f[i][k]=f[i][k]-Q*ksi*v[i][k];
}
```

*************** "layers.c " *****************

Evaluate the atom distribution (instant and average) between the layers. Calculate distance between the neighboring layers.

```
# include "initdata.h"
# include "func_cp.h"
# include <math.h>
# include <stdio.h>
# include <stdlib.h>
void inter_layer_dist(int mode)
{
extern double r[Atoms][3],dis_bet_lay[NUMBER_OF_LAYERS-1],counter_dist;
extern int layers[NUMBER_OF_LAYERS][Atoms];
int i,k,p;
double z_prev, z_new;
 FILE *fd;
if(mode==0)
{
  //Initialization:
  fprintf(stderr,"\n Inter Layer Distances initialization (:)");
  counter_dist=0.;
        for (i=0;i<NUMBER_OF_LAYERS-1;i++)
  dis_bet_lay[i]=0.;
}
//Start calculations
 z_new=z_prev=0.;
for (i=0;i<NUMBER_OF_LAYERS;i++)
{
        z_new=0;
                for (p=1;p<=layers[i][0];p++)
        {
                k=layers[i][p];
                z_new+=r[k][2];
        }
                if(layers[i][0]>0)
                {
z_new/=layers[i][0]; // Coordinate of c.m.c. of the layer
if(i>0) dis_bet_lay[i-1]+=z_new-z_prev;
z_prev=z_new;
                }
}
counter_dist++; // Number of the time measurements
if(mode==2)
{ //Result Output
        if((fd=fopen("dis_bet_lay.txt","w"))!=NULL)
  {
  for (i=0;i<(NUMBER_OF_LAYERS-1);i++)
  {
        if(counter_dist>0)
  fprintf(fd,"%d %lf \n",i+1,dis_bet_lay[i]/counter_dist);
  }
  fclose(fd);
  }
}
}
```

```
////////////////////////////////////
void inter_layer_dist_rw(char *fn,int mode)
{
extern double dis_bet_lay[NUMBER_OF_LAYERS-1],counter_dist;
int i;
FILE *fd;
if(mode==0)
{
////
if((fd=fopen(fn,"r"))!=NULL)
{
fscanf(fd,"\n%lf",&counter_dist);
for (i=0;i<NUMBER_OF_LAYERS-1;i++)
fscanf(fd,"\n%lf", &dis_bet_lay[i]);
fclose(fd);
}
else  inter_layer_dist(0);
////
}
if(mode==1)
{
////////////
if((fd=fopen(fn,"w"))!=NULL)
{
fprintf(fd,"\n %lf",counter_dist);
for (i=0;i<NUMBER_OF_LAYERS-1;i++)
fprintf(fd,"\n%lf", dis_bet_lay[i]);
fclose(fd);
}
///////////
}
}
//============================================//
void atom_distribution_in_layers (int mode)
{
extern double r[Atoms][3],Width;
extern int  layers[NUMBER_OF_LAYERS][Atoms];
//[][]-Number of a layer, number of Atoms [0][],[][i>0] indexes
extern double a0;
extern  int     atoms_distribution[NUMBER_OF_LAYERS];
extern  long  counter_of_atoms_distribution,step;
double   Z0;
double  dz;
int i,j,index;
FILE * fd;
char fn[80];
if(mode==0)
{
 fprintf(stderr,"\n Atoms distribution in the layers initialization (:) ");
counter_of_atoms_distribution=0;
for (i=0;i<NUMBER_OF_LAYERS;i++)
atoms_distribution[i] =0;
}
for (i=0;i<NUMBER_OF_LAYERS;i++)
for (j=0;j<Atoms;j++)
layers[i][j]=0;
// Collect the Statistics
 Z0=NUMBER_OF_FROZEN_LAYERS *Width-Width/2;
 for (i=FROZEN_ATOMS;i<Atoms;i++)
 {
dz=r[i][2]-Z0;
if(dz<0)
{
if(r[i][2]<(NUMBER_OF_FROZEN_LAYERS -1)*Width )
{
 fprintf(stderr,"\n layers.c:  Very Negative Base Layer
i=%d Z0=%lf r(i,3)=%lf  diff=%lf ",i,Z0,r[i][2],r[i][2]-Z0);
 exit(0);
}
dz=-dz;
}
//////////
  if(fmod(dz,Width)==0)
 {
 index=(int)(dz/Width)-1;
 if(index<NUMBER_OF_LAYERS){layers[index][0]+=1; layers[index][layers[index][0]]=i;}
 else {fprintf(stderr,"\n  layers.c :Super adlayer %d",index);exit(0);}
 }
else
{index=(int)(dz/Width);
 if(index<NUMBER_OF_LAYERS)
{
layers[index][0]+=1;
layers[index][layers[index][0]]=i;
}
 else {
//Adlayer
index=NUMBER_OF_LAYERS-1;
layers[index][0]+=1;
layers[index][layers[index][0]]=i;
 }
 }
}
 for (i=0;i<NUMBER_OF_LAYERS;i++)
{
    atoms_distribution[i] +=layers[i][0];
    sprintf(fn,"layer_%d.txt",i);
    if((fd=fopen(fn,"a+"))!=NULL){
        fprintf(fd,"\n%ld %d", step,layers[i][0]);
        fclose(fd); }
}
    counter_of_atoms_distribution++;
```

```
}
//==========================================================//
 void atom_distribution_in_layers_rw (char *fn, int mode)
{
extern  int atoms_distribution[NUMBER_OF_LAYERS];
extern  long    counter_of_atoms_distribution;
int i;
FILE * fd;
 if(mode==0)
{
  if((fd=fopen(fn,"w"))!=NULL)
{
 ///////////////
  fprintf(fd,"\n%ld",counter_of_atoms_distribution);
  for (i=0;i<NUMBER_OF_LAYERS;i++)
  fprintf(fd,"\n%d",atoms_distribution[i] );
  fclose(fd);
/////////////////
 }
 }
 if(mode==1)
{
if((fd=fopen(fn,"r"))!=NULL)
{
  fscanf(fd,"\n%ld",&counter_of_atoms_distribution);
 for (i=0;i<NUMBER_OF_LAYERS;i++)
  fscanf(fd,"\n%d", &atoms_distribution[i] );
      fclose(fd);
    }
else   atom_distribution_in_layers (0);
 }
}
 *************** "density_profile.c" *****************
```

Evaluate the atomic density along the z-direction. Use the Chen et al. method to facilitate the obtained data.

```
# include <stdio.h>
# include <math.h>
# include <stdlib.h>
# include "func_cp.h"
# include "potential.h"
# include "initdata.h"
# include "constants.h"
void chen_density_profile(int mode)
{
extern double r[Atoms][3],Width;
extern double dens_ch[Z_RES],chen_counter;
double z_in,z_fin,delta,z,f,sigma_ch;
int i,j;
FILE *fd;
z_in=NUMBER_OF_FROZEN_LAYERS*Width;
z_fin=z_in+(NUMBER_OF_LAYERS+4)*Width;
delta=(z_fin-z_in)/Z_RES;
```

```
sigma_ch=0.1*Width;
if(mode==0)
{
//Initialization:
    fprintf(stderr,"\n Chen density profile initialization  (:) ");
        chen_counter=0;
        for (i=0;i<Z_RES;i++)
    dens_ch[i]=0.;
}
// Let's start calculations
for (i=0;i<Z_RES;i++)
{
        z=z_in+i*delta;
for (j=FROZEN_ATOMS;j<Atoms;j++)
{
f=(z-r[j][2])*(z-r[j][2])/(2*sigma_ch*sigma_ch);
dens_ch[i]+=exp(-f)/(sqrt(2*pi)*sigma_ch);
}
}
chen_counter++; // Number of the measurements
// Let's print the results
if(mode==2)
{
  if((fd=fopen("chen_density.txt","w"))!=NULL)
  {
        for (i=0;i<Z_RES;i++)
  fprintf(fd,"%lf  %lf \n",z_in+i*delta,dens_ch[i]/chen_counter);
  fclose(fd);
  }
}
}
/////////////////////////////
void chen_density_profile_rw(char *fn, int mode)
{
extern double dens_ch[Z_RES],chen_counter;
int i;
FILE *fd;
if(mode==1)
{
// BackUp
 if((fd=fopen(fn,"w"))!=NULL)
{
fprintf(fd,"%lf",chen_counter);
for (i=0;i<Z_RES;i++)   fprintf(fd,"%lf ",dens_ch[i]);
fclose(fd);
}
}
if(mode==0)
{
//Read
if((fd=fopen(fn,"r"))!=NULL)
{
fscanf(fd,"%lf",&chen_counter);
for (i=0;i<Z_RES;i++) fscanf(fd,"%lf ",&dens_ch[i]);
fclose(fd);
}
else    chen_density_profile(0);
}
}
 *************** "temp_prof.c " *****************
```

Find the temperature profile along the z-directions.

```
# include "initdata.h"
# include "constants.h"
# include "func_cp.h"
# include "potential.h"
# include <math.h>
# include <stdio.h>
void temperature_profile (int mode)
{
extern double v[Atoms][3],temp_profile[NUMBER_OF_LAYERS],counter_temp;
extern int layers[NUMBER_OF_LAYERS][Atoms];
int i,j,k;
double s;
FILE *fd;
if(mode==0)
{ //Initialization:
   fprintf(stderr,"\n Temperature profile initialization  (:) ");
   counter_temp=0.;
        for (i=0;i<NUMBER_OF_LAYERS;i++)
    temp_profile[i]=0.;
}
//Start calculations
 for (i=0;i<NUMBER_OF_LAYERS;i++)
{
         s=0.;
                for (j=1;j<=layers[i][0];j++)
        {
                k=layers[i][j];
            s+=(v[k][0]*v[k][0]+v[k][1]*v[k][1]+v[k][2]*v[k][2]);
        }
if(layers[i][0]>10)
{
        s/=kb*3*layers[i][0];
        temp_profile[i]+=s;
}
}
counter_temp++; // Number of the time measurements
if(mode==2)
{ //Result Output
  if((fd=fopen("temp_profile.txt","w"))!=NULL)
  {
  for (i=0;i<NUMBER_OF_LAYERS;i++)
 if(counter_temp>0) fprintf(fd,"%d  %lf  \n",i+1,temp_profile[i]/counter_temp);
 //  for (i=0;i<NUMBER_OF_LAYERS;i++)
 //   fprintf(stderr," \n %lf  %lf ",counter_temp,temp_profile[i]/counter_temp);
   fclose(fd);
  }
}
}
//======================//
void temperature_profile_rw(char *fn,int mode)
 {
extern double temp_profile[NUMBER_OF_LAYERS],counter_temp;
int i;
FILE *fd;
  if(mode==0)
{
if((fd=fopen(fn,"r"))!=NULL)
{
     for (i=0;i<NUMBER_OF_LAYERS;i++)
         fscanf(fd," \n %lf ",&temp_profile[i]);
         fscanf(fd," \n %lf ",&counter_temp);
         fclose(fd);
  }
else   temperature_profile( 0 ) ;    // initialization
}
  if(mode==1)
{
if((fd=fopen(fn,"w"))!=NULL)
{      for (i=0;i<NUMBER_OF_LAYERS;i++)
         fprintf(fd," \n %lf ",temp_profile[i]);
         fprintf(fd," \n %lf ",counter_temp);
         fclose(fd);
  }
}
}
 *************** "order_parameter.c " *****************
```

Calculate the structure order parameters along the x,y, and z-directions.

```
# include "initdata.h"
# include "constants.h"
# include "func_cp.h"
# include "potential.h"
# include <math.h>
# include <stdio.h>
void order_parameters(int mode)
{
extern double r[Atoms][3];
extern int layers[NUMBER_OF_LAYERS][Atoms];
// The order parameter is defined in each layer and  in various directions
extern double S[NUMBER_OF_LAYERS][NUMBER_OF_DIRECTIONS][2] ;
extern double a0;

double k[NUMBER_OF_DIRECTIONS][3];
double sx,sy,sz,dx,dy,S_c,S_s;
int index,p,i,j,Ns,parity;
double or1,or2,or3;
FILE *fd;
char fn[50];
//   Directions  [001]
if(surface==0)
{
k[0][0]=4*pi/a0; k[0][1]=0.; k[0][2]=0.;
k[1][0]=0.; k[1][1]=4*pi/a0; k[1][2]=0.;
 k[2][0]=0.;  k[2][1]=0.;  k[2][2]=4*pi/a0;
 }
 if(surface==1)
{
k[0][0]=2*sqrt(2)*pi/a0; k[0][1]=0.; k[0][2]=0;
k[1][0]=0; k[1][1]=4*pi/a0; k[1][2]=0.;
 k[2][0]=0;  k[2][1]=0;  k[2][2]=2*sqrt(2)*pi/a0;
 }
 if(surface==2)
{
k[0][0]=4*pi/(a0*sqrt(2.)); k[0][1]=0.; k[0][2]=0.;
k[1][0]=0; k[1][1]=4*pi/(2.*a0*sqrt(3./2.)); k[1][2]=0.;
 k[2][0]=0;  k[2][1]=0;  k[2][2]=2*pi/(a0/(2*sqrt(3.)));
 }
if(mode ==0)
{
 // Initiallization of the arrays
 fprintf(stderr,"\n Order parameter profile initialization  (:) ");
 for (i=0;i<NUMBER_OF_LAYERS;i++)
 for (j=0;j<NUMBER_OF_DIRECTIONS;j++)
 { S[i][j][0]=0.; S[i][j][1]=0.; }
}
for (p=0;p<3;p++)//3D space
{//p
for (i=0;i<NUMBER_OF_LAYERS;i++)
{//Initialization of the arrays
Ns=0;
S_c=S_s=0.0;
for (j=1;j<=layers[i][0];j++)
{
index=layers[i][j];
//Take all the particles
sz=k[p][0]*r[index][0]+k[p][1]*r[index][1]+k[p][2]*r[index][2];
S_c+=cos(sz);
S_s+=sin(sz);
Ns++;
}  // end of the j cycle
if(Ns)sx=(S_c * S_c+S_s * S_s)/ (Ns*Ns);
if(layers[i][0]>20)
{
 S[i][p][0]+=sx;  S[i][p][1]++;
}
} // end of the i Layer
} // 3D space
if(mode==2)
{
if((fd=fopen("order_prof.txt","w"))!=NULL){
for (i=0;i<NUMBER_OF_LAYERS;i++)
{
or1=or2=or3=0.;
if(S[i][0][1]>0)or1=S[i][0][0]/S[i][0][1];
if(S[i][1][1]>0)or2=S[i][1][0]/S[i][1][1];
if(S[i][2][1]>0)or3=S[i][2][0]/S[i][2][1];
fprintf(fd,"\n %d %lf %lf %lf",i+1,or1,or2,or3);
// Start calculate from 1
}
fclose(fd);
}
else{fprintf(stderr,"\n Can't open the %s file ",fn);exit(0);}
}//if(mode==2)
}
//=====================================================//
void order_parameters_rw(char *fn, int mode )
{
extern double S[NUMBER_OF_LAYERS][NUMBER_OF_DIRECTIONS][2] ;
int i;
FILE *fd;
if(mode==0)
{
/////////////////////////////////////
if((fd=fopen(fn,"r"))!=NULL)
{
for (i=0;i<NUMBER_OF_LAYERS;i++)
fscanf(fd," \n %lf %lf %lf %lf %lf %lf",
&S[i][0][0],&S[i][1][0],&S[i][2][0],&S[i][0][1],&S[i][1][1],&S[i][2][1]);
fclose(fd);
}
else order_parameters(0);
/////////////////////////////////////
}
if(mode==1)
{
/////////////////////////////////////
if((fd=fopen(fn,"w"))!=NULL)
{
 for (i=0;i<NUMBER_OF_LAYERS;i++)
 fprintf(fd,"\n %lf %lf %lf %lf %lf  %lf",
S[i][0][0],S[i][1][0],S[i][2][0],S[i][0][1],S[i][1][1],S[i][2][1]);
 fclose(fd);
}
/////////////////////////////////////
}
```

```
}
 *************** “ pair_correlation.c ” *****************
```

Evaluate the atomic 2D radial distribution function

```
# include ”initdata.h”
# include ”constants.h”
# include ”func_cp.h”
# include ”potential.h”
# include <math.h>
# include <stdio.h>
# include <stdlib.h>
void pair_correlation(int mode)
{
extern double r[Atoms][3],Profile[NUMBER_OF_LAYERS][PAIR_STEP];
extern double lx,ly,width_rdf, normFac,half_box;
extern int layers[NUMBER_OF_LAYERS][Atoms];
extern long pair_corr_count;
extern double a0;
int i, j, k,  index;
int kk,jj;
double dist,s;
double dis[2];
char fn[30];
FILE * fd;
if(mode ==0)
{
if (lx>ly) half_box=ly/2;
else  half_box=lx/2;
width_rdf=half_box/PAIR_STEP;
normFac=lx*ly/(2*pi*width_rdf*width_rdf);
 // Initiallization of the arrays
fprintf(stderr,”\n Pair Correlation  initialization (:) ”);
for (i=0;i<NUMBER_OF_LAYERS;i++)
for (j=0;j<PAIR_STEP;j++)
        Profile[i][j]=0.;
        pair_corr_count=0;
 }
// Designing the G(r)  function
for (i=0;i<NUMBER_OF_LAYERS;i++)
{
 for (kk=1;kk<layers[i][0];kk++)  {  //  layers[i][0] is instant number of atoms in i layer
 for (jj=kk+1;jj<=layers[i][0];jj++) {

//Periodic Boundary Conditions for the xy (parallel to the surafce) components
   dist=0.;
   k=layers[i][kk];
   j=layers[i][jj];
  dis[0]=r[j][0]-r[k][0];
   if(fabs(dis[0])> lx/2.0)
{
   if(dis[0]>0.)dis[0]-=lx;
   else dis[0]+=lx;
 }

  dis[1]=r[j][1]-r[k][1];
   if(fabs(dis[1])> ly/2.0)
{
   if(dis[1]>0.)dis[1]-=ly;
   else dis[1]+=ly;
 }
// End of the Periodic Boundary Conditions for the xy (paralle to the surafce) components
  dist=dis[0]*dis[0]+dis[1]*dis[1];
  dist=sqrt(dist);

// Check what you still inside the box  (just to be on the safe side)
   index=(int)(dist/width_rdf);
if ( index < PAIR_STEP ) {
if(layers[i][0]>0){
  Profile[i][index]+=2./(layers[i][0]*layers[i][0]); // Rij & Rji contributions
 }
 }  // j j
} //  kk
} //  i:  NUMBER_OF_LAYERS
pair_corr_count++;
} //==================

if(mode==2){
  /*Final Output for each layer*/
  for (i=0;i<NUMBER_OF_LAYERS;i++)
{
  sprintf(fn,"pair_func_%d.txt",i);

   if((fd=fopen(fn,"w"))!=NULL)
  {
          for(j=0;j<PAIR_STEP;j++)
          {
  if(pair_corr_count>0){
  fprintf(fd,"%lf %lf \n",(j+0.5)*width_rdf, normFac*Profile[i][j]/((j+0.5)*pair_corr_count));
          }
                 else
                 {fprintf(fd,"%lf %lf \n",j*width_rdf,0.); }
          }
        fclose(fd);    }
 }   // end for
 }
}
//================================//
void pair_correlation_rw(char *fn, int mode )
{
extern double Profile[NUMBER_OF_LAYERS][PAIR_STEP];
extern long  pair_corr_count;
int i,j;
FILE *fd;
```

```
if(mode==0)
{
if((fd=fopen(fn,"r"))!=NULL)
{
for (i=0;i<NUMBER_OF_LAYERS;i++)
for (j=0;j<PAIR_STEP;j++)
fscanf(fd," \n %lf ",&Profile[i][j]);
fscanf(fd," \n %ld ",&pair_corr_count);
fclose(fd);
}
 else pair_correlation(0);
}
if(mode==1)
{
if((fd=fopen(fn,"w"))!=NULL)
{
  for (i=0;i<NUMBER_OF_LAYERS;i++)
  for (j=0;j<PAIR_STEP;j++)
  fprintf(fd," \n %lf ",Profile[i][j]);
  fprintf(fd," \n %ld ",pair_corr_count);
  fclose(fd);
 }
}
}
 *************** "diffusion.c " *****************
```

Calculate in-plane the out-plane diffusion coefficients

diff_counter	**– number of diffusion(1) calls**
NUMBER_OF_CALLS	**– number of calls when a new t0 is taken**
interval	**– number of steps betwee two succesive calls**
T0max	**– maximal number of time origins in the "initdata.h"**
TMAX	**– maximal number of sampling in the "initdata.h"**
t0max	**– maximal number of time origins**
tmax	**– maximal number of sampling**
t0	**– number of time origins**

```
# include "initdata.h"
# include "constants.h"
# include "func_cp.h"
# include "potential.h"
# include <math.h>
# include <stdio.h>
void diffusion(int mode)
{
extern double r[Atoms][3],lx,ly;
extern int layers[NUMBER_OF_LAYERS][Atoms];
extern long num_of_meas,diff_counter,t0max,tmax;
extern  int    ntime[TMAX],pbc[Atoms][2],time0[T0max],t0,interval;
extern  double r0[Atoms][3][T0max];
extern  double r2f[TMAX][3][NUMBER_OF_LAYERS];
long delta,tt0;
int i,j,k,n,p;
```

```
double dtime, norm;
FILE *fr;
char fn[30];
if(mode ==0)
{
fprintf(stderr,"\n Diffusion initialization  (:) ");
tmax=t0+num_of_meas;
if (tmax > TMAX)
{
if((fr=fopen("README","w"))!=NULL)
{
fprintf(fr,"\n tmax=%ld > TMAX=%ld",tmax,TMAX);
fprintf(fr,"\n Only the shortest time TMAX=%ld is taken ",TMAX);
fclose(fr);
tmax=TMAX;
}
}
t0max=(long)1.*tmax/NUMBER_OF_CALLS;
if (t0max > T0max) t0max = T0max;
    for(i=t0;i<t0max;i++)
    time0[i]=0;//The current time origin
            for(i=t0;i<tmax;i++)    {
     ntime[i]=0; // delta table
    for(j=0;j<NUMBER_OF_LAYERS;j++) {
              for(k=0;k<3;k++)
r2f[i][k][j]=0.;          }
   }
  for (i=FROZEN_ATOMS;i<Atoms;i++)
  for (k=t0;k<t0max;k++)
  for (j=0;j<3;j++)
  r0[i][j][k]=0.;//Starting Points
} // End of Start Initialization
// Start of Sampling
if(diff_counter%NUMBER_OF_CALLS==0)
{ //Let's  take a new time origin
tt0=t0-(int)(t0/t0max)*t0max;//Check if number of time origins less than tmax, if not remove the first meas.
++t0; //Number of time origins are taken
time0[tt0]=diff_counter;     // Store the time at t=0;
for (i=FROZEN_ATOMS;i<Atoms;i++)
{
//For Future Parallelization
r0[i][0][tt0]=r[i][0]+pbc[i][0]*lx; //Starting Points
r0[i][1][tt0]=r[i][1]+pbc[i][1]*ly; //Starting Points
r0[i][2][tt0]=r[i][2];           //Starting Points
} //   for(i=FROZEN_ATOMS;i<Atoms;i++)
} //  if(diff_counter%NUMBER_OF_CALLS==0)
 n=(t0<t0max)?t0:t0max;
 for(k=0;k<n;k++) //Update the r2f now
{
     delta=diff_counter-time0[k]; //Actual time minus time origin
    if(delta<tmax) // Watch out the matrix boundaries
```

```
{
ntime[delta]++;
//  norm=1.0/(Atoms-FROZEN_ATOMS);// Divide on the number of the particles in the layer
  for (j=0;j<NUMBER_OF_LAYERS;j++)
        {
        for (p=1;p<=layers[j][0];p++)
        {
         i=layers[j][p];
   if(layers[j][0]) norm=1.0/layers[j][0];// Divide on the number of the particles in the layer
   else  norm=0.0;
r2f[delta][0][j]+=(r[i][0]+pbc[i][0]*lx-r0[i][0][k])*(r[i][0]+pbc[i][0]*lx
- r0[i][0][k])*norm;
r2f[delta][1][j]+=(r[i][1]+pbc[i][1]*ly-r0[i][1][k])*(r[i][1]+pbc[i][1]*ly
- r0 [i][1][k])*norm;
r2f[delta][2][j]+=(r[i][2]-r0[i][2][k])*(r[i][2]-r0[i][2][k])*norm;
 } //  for (p=1;p<=layers[j][0];p++)
 } // for (j=0;j<NUMBER_OF_LAYERS;j++)
   }// if(delta<TMAX)
 } // for(k=0;k<n;k++)
diff_counter++;  // Number of calls
                // End of the Sampling
if(mode==2)
{   // Output of  the Results
         dtime=2*0.721*dt*interval;
   for (k=0;k<NUMBER_OF_LAYERS;k++){
         sprintf(fn,"r2f%d.txt",k);
   if((fr=fopen(fn,"w"))!=NULL){///——-
          for(i=0;i<tmax;i++)
          {
        if(ntime[i]>0)
                {
norm=1./ntime[i];
fprintf(fr,"%lf %lf  %lf  %lf \n",
dtime*(i+0.5),r2f[i][0][k]*norm,r2f[i][1][k]*norm,r2f[i][2][k]*norm);
                } //(ntime[i]>0)
        }  //  for(i=0;i<TMAX;i++)
 fclose(fr);    }
  else{fprintf(stderr,"\n\n\n Can't open: r2f%d.txt",k);}
      }
}//End of  Output of the Results }
//====================//
void diffusion_rw(char *fn,int mode)
{
extern  long   diff_counter,t0max,tmax;
extern  int    ntime[TMAX],time0[T0max],t0;
extern  double r0[Atoms][3][T0max];
extern  double r2f[TMAX][3][NUMBER_OF_LAYERS];
FILE *fd;
int i,j;
if (mode==0)
{
 if((fd=fopen(fn,"r"))!=NULL)
{
 fscanf(fd," %ld %ld %ld ",&diff_counter,&t0max,&tmax);
 fscanf(fd," \n %d ",&t0);
 for(i=0;i<tmax;i++)
 fscanf(fd," \n %d ",&ntime[i]);
 for(i=0;i<t0max;i++)
 fscanf(fd," \n %d ",&time0[i]);
 for(i=0;i<tmax;i++)
 for(j=0;j<NUMBER_OF_LAYERS;j++)
 fscanf(fd," \n %lf %lf %lf ",&r2f[i][0][j],&r2f[i][1][j],&r2f[i][2][j]);
 for (i=FROZEN_ATOMS;i<Atoms;i++)
 for (j=0;j<t0max;j++)
 fscanf(fd," \n %lf %lf %lf  ",&r0[i][0][j],&r0[i][1][j],&r0[i][2][j]);
 fclose(fd);
   }
 else {diffusion(0);system("rm -f pbc.back;");}
}
if (mode==1)
{
 if((fd=fopen(fn,"w"))!=NULL)
{
 fprintf(fd," %ld %ld %ld ",diff_counter,t0max,tmax);
 fprintf(fd," \n %d ",t0);

 for(i=0;i<tmax;i++)
 fprintf(fd," \n %d ",ntime[i]);
 for(i=0;i<t0max;i++)
 fprintf(fd," \n %d ",time0[i]);
 for(i=0;i<tmax;i++)
 for(j=0;j<NUMBER_OF_LAYERS;j++)
 fprintf(fd," \n %lf %lf %lf ",r2f[i][0][j],r2f[i][1][j],r2f[i][2][j]);
 for (i=FROZEN_ATOMS;i<Atoms;i++)
 for (j=0;j<t0max;j++)
 fprintf(fd," \n %lf %lf %lf  ",r0[i][0][j],r0[i][1][j],r0[i][2][j]);
 fclose(fd);
       } }
}
 *************** "read_save.c " *****************
```

The various input-output functions: Aviz, mixing, read_vel_from_file(char *fn), read_coord_from_file(char *fn) save_to_file(long no,int d), save_inf(long step,long counter,float stam_real), get_inf(void) pbc_recover(void), pbc_init(void)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
# include "initdata.h"
# include "constants.h"
# include "func_cp.h"
 void aviz(char *fn)
{
extern double r[Atoms][3];
extern int Color[Atoms ];
FILE * fd;
int j,i;

char *color[]=
{"a1","a2","a3","a4","a5",
"a6","a7","a8","a9","b1",
"b2","b3","b4","b5","b6",
"b7","b8","b9","c1","c2",
"c3","c4","c5","c6","c7",
"c8","c9","d1","d2","d3"} ;
if(surface==0) {
if((fd=fopen(fn,"w"))!=NULL)
 {
fprintf(fd," %d ", Atoms);
fprintf(fd,"\n ### Surface simulations ");
for (j=0;j<FROZEN_ATOMS;j++)
fprintf(fd,"\n Cu %lf  %lf  %lf ", r[j][0],r[j][1],r[j][2]);
for(i=0;i<INIT_LAYERS-NUMBER_OF_FROZEN_LAYERS;i++)
for (j=FROZEN_ATOMS+i*size_x*size_y;j< FROZEN_ATOMS+(i+1)*size_x*size_y; j++)
fprintf(fd,"\n %s %lf  %lf  %lf ",color[i],r[j][0],r[j][1],r[j][2]);
fclose(fd);
}
 else  { fprintf(stderr,"I can't open the file %s !!!",fn); exit(0);}
}  if(surface==1)
{
 if((fd=fopen(fn,"w"))!=NULL)
 {
fprintf(fd," %d ", Atoms);
fprintf(fd,"\n ### Surface simulations [011] ");
for (j=0;j<Atoms;j++)
if (  Color[j]< NUMBER_OF_FROZEN_LAYERS)fprintf(fd,"\n Cu %lf  %lf  %lf ", r[j][0],r[j][1],r[j][2]);
else  fprintf(fd,"\n %s %lf  %lf  %lf ",color[Color[j]],r[j][0],r[j][1],r[j][2]);
fclose(fd);
}
else  { fprintf(stderr,"I can't open the file %s !!!",fn); exit(0);}
}
if(surface==2) {
 if((fd=fopen(fn,"w"))!=NULL)
 {
fprintf(fd," %d ", Atoms);
fprintf(fd,"\n ### Surface simulations [111] ");
for (j=0;j<Atoms;j++)
if (  Color[j]< NUMBER_OF_FROZEN_LAYERS)fprintf(fd,"\n Fe %lf  %lf  %lf ", r[j][0],r[j][1],r[j][2]);
else  fprintf(fd,"\n %s %lf  %lf  %lf ",color[Color[j]],r[j][0],r[j][1],r[j][2]);
fclose(fd);
}
else  { fprintf(stderr,"I can't open the file %s !!!",fn); exit(0);}
}
   }
//================================//
void read_vel_from_file(char *fn)
```

```
{
extern double v[Atoms][3];
int i;
FILE *fp;
   fprintf(stderr,"\n %s",fn);
    if((fp=fopen(fn,"r"))!=NULL)
 {
  for(i=0;i<Atoms;i++)
  fscanf(fp,"%lf %lf %lf \n",&v[i][0],&v[i][1],&v[i][2]);
  //printf("N=%d %g %g %g \n",i,v[i][0],v[i][1],v[i][2]);
  fclose(fp);
    }
        else
        {fprintf(stderr,"\n I can't read file: %s",fn);exit(0);}
 }
//======================================//
void read_coord_from_file(char *fn)
{
extern double r[Atoms][3];
 int i;
 FILE *fd;
    fprintf(stderr,"\n %s",fn);
 if((fd=fopen(fn,"r"))!=NULL){
   for(i=0;i<Atoms;i++)
   fscanf(fd,"%lf %lf %lf \n",&r[i][0],&r[i][1],&r[i][2]);
   //fprintf(stderr,"N=%d %g %g %g \n",i,r[i][0],r[i][1],r[i][2]);
    fclose(fd);              }
        else
        {fprintf(stderr,"\n I can't read file: %s",fn);exit(0);}
}

//==========================//
void energy_temp_pressure()
{
extern long step;
extern double pot_E,kin_E,temp;
extern double temp_current,press_current;
 FILE *fd;
if((fd=fopen("infoEPT.txt","a+"))!=NULL)
{
fprintf(fd,"\n %ld   %lf  %lf   %lf   %lf",step,pot_E,kin_E,temp_current,press_current);
fclose(fd);
}
}
//==========================================//
void save_configuration(int mode)
{
extern double r[Atoms][3],v[Atoms][3];
FILE *fd;
int i;
if(mode)
```

```
{
if((fd=fopen("r.log","w"))!=NULL)
{
for( i=0;i<Atoms;i++)
fprintf(fd," \n %lf %lf %lf",r[i][0],r[i][1],r[i][2]);
fclose(fd);
}
if((fd=fopen("v.log","w"))!=NULL)
{
for( i=0;i<Atoms;i++)
fprintf(fd," \n %lf %lf %lf",v[i][0],v[i][1],v[i][2]);
fclose(fd);
}
}
}
//===================================//
void save_inform(char *fn)
{
  extern int regime,interval,Limit_aviz_pict;
  extern long step,step0,num_eq_steps,num_of_meas;
  extern double temp,Q;
 FILE *outd;
  float percent;
 // Calculate percentage of the accomplished job
   if(regime<2)
   {percent=100.0*(step-step0)/num_eq_steps;}
   else
   {percent=100.0*(step-step0)/(num_of_meas*interval);}
      if((outd=fopen(fn,"w"))!=NULL)
        {
   fprintf(outd,"\n %d",regime);
   fprintf(outd,"\n %4.0f",temp);
   fprintf(outd,"\n %ld",num_eq_steps);
   fprintf(outd,"\n %d",interval);
   fprintf(outd,"\n %ld",num_of_meas);
   fprintf(outd,"\n %ld",step);
   fprintf(outd,"\n %d",Limit_aviz_pict);
   fprintf(outd,"\n %3.1f",percent);
   fprintf(outd,"\n %f",Q);
   fprintf(outd,"\n %d",Atoms);
   fprintf(outd,"\n %d",(int)size_x*size_y);
         fprintf(outd,"\n %d",NUMBER_OF_FROZEN_LAYERS);
   fprintf(outd,"\n %d",INIT_LAYERS-NUMBER_OF_FROZEN_LAYERS);
   switch (surface)
  {
  case 0:
  fprintf(outd,"\n [001]");
  break;
  case 1:
  fprintf(outd,"\n [011]");
  break;
```

```
    case 2:
   fprintf(outd,"\n [111]");
   break;
   default:
   printf("Too Strange, too much different surface=%d \n", surface);
   exit(0);
   break;
   }
    fclose(outd);
    comments(fn);
      }
    else
     {
     fprintf(stderr,"\n Can't open the file:\"%s\"",fn);
     exit(0);
     }
}
//===========================//
long get_inform(char *fn)
{
  // Get all necessary information

  extern int regime,interval,Limit_aviz_pict,step_Aviz;
  extern long step,step0,num_eq_steps,num_of_meas;
  extern double temp,Q;
        float percent;
        FILE *outd;
     if((outd=fopen(fn,"r"))!=NULL)
         {
   fscanf(outd,"\n %d",&regime);
   fscanf(outd,"\n %lf",&temp);
   fscanf(outd,"\n %ld",&num_eq_steps);
   fscanf(outd,"\n %d",&interval);
   fscanf(outd,"\n %ld",&num_of_meas);
   fscanf(outd,"\n %ld",&step0);
   fscanf(outd,"\n %d",&Limit_aviz_pict);
   fscanf(outd,"\n %g",&percent);
   fscanf(outd,"\n %lf",&Q);
   fclose(outd);
   if (regime) fprintf(stderr,"\n Regime: %d -> measurements",regime);
   else fprintf(stderr,"\n Regime: %d -> equilibration",regime);
   fprintf(stderr,"\n\n Temperature %4.0f K",temp);
   fprintf(stderr,"\n  Number of steps in approaching to the equilibrium %ld",num_eq_steps);
   fprintf(stderr,"\n  Number of steps in the measurement regime %d \n ",interval*num_of_meas);
   fprintf(stderr,"\n\t  Number of steps between two measurements %d",interval);
   fprintf(stderr,"\n\t  Number of measurements %ld",num_of_meas);
   fprintf(stderr,"\n \t The last step was = %ld \n ",step0);
   fprintf(stderr,"\n \t The task have been complited on = %3.0f percent",percent);
fprintf(stderr,"\n \n  \t   Layers + addlayer = %d + %d",NUMBER_OF_LAYERS-1,1);
fprintf(stderr,"\n     \t   Frozen Atoms =%d",FROZEN_ATOMS);
fprintf(stderr,"\n     \t   Frozen Layers =  %d",NUMBER_OF_FROZEN_LAYERS );
fprintf(stderr,"\n \n \t   Total number of steps =%d",num_eq_steps+num_of_meas*interval);
fprintf(stderr,"\n \t   Real time  = %lf psec",(num_eq_steps+num_of_meas*interval)*dt*0.721);

 step_Aviz=(int)(1.0*num_eq_steps+num_of_meas*interval)/Limit_aviz_pict;
         }
     else
   {
   fprintf(stderr,"\n Can't open the file:\"%s\" , I try to guess ",fn);
   if((outd=fopen(fn,"w"))!=NULL)
    {
   fprintf(outd,"\n %d",0);
   fprintf(outd,"\n %4.0f",1800.0);
   fprintf(outd,"\n %d",1000);
   fprintf(outd,"\n %d",100);
   fprintf(outd,"\n %d",100);
   fprintf(outd,"\n %d",0);
   fprintf(outd,"\n %d",10);
   fprintf(outd,"\n %3.1f",0.0);
   fprintf(outd,"\n %lf",6.0);
   fprintf(outd,"\n %d",Atoms);
   fprintf(outd,"\n %d",(int)size_x*size_y);
   fprintf(outd,"\n %d",NUMBER_OF_FROZEN_LAYERS);
   fprintf(outd,"\n %d",INIT_LAYERS-NUMBER_OF_FROZEN_LAYERS);
   switch (surface)
  {
 case 0:
  fprintf(outd,"\n [001]");
  break;
  case 1:
  fprintf(outd,"\n [011]");
  break;
   case 2:
  fprintf(outd,"\n [111]");
  break;
  default:
  printf("Too Strange, too much different surface=%d \n", surface);
  exit(0);
  break;
  }
   fclose(outd);
   }
   comments(fn);
   system("more -d aaa_info.txt");
   exit(0);
    }

}
//==============================//
 void comments(char *fn)
{
  FILE *outd;
  if((outd=fopen(fn,"a"))!=NULL)
  {
    fprintf(outd,"\n *************** Main Parameters ***************** ");
    fprintf(outd,"\n Regime: 0-> start, 1 -> equilibration, 2-> measurements");
    fprintf(outd,"\n Temperature");
    fprintf(outd,"\n Number of steps in order to approach to the equilibrium ");
    fprintf(outd,"\n Number of steps between two succesive measurements ");
    fprintf(outd,"\n Number of measurements ");
    fprintf(outd,"\n Initial step ");
    fprintf(outd,"\n Number of the aviz pictures");
    fprintf(outd,"\n Percent of the accomplished job");
    fprintf(outd,"\n NVE/NVT parameter");
    fprintf(outd,"\n Number of the atoms ");
    fprintf(outd,"\n Number of the atoms in a layer ");
    fprintf(outd,"\n Number of frozen layers");
    fprintf(outd,"\n Number of free layers");
    fprintf(outd,"\n The surface type: ");
    fprintf(outd,"\n *************** Main Parameters ***************** ");
    fclose(outd);
}
   else
{ fprintf(stderr,"\n Can't open the file:\"%s\"",fn); }
}
//====================================//
void backup(char *fn,int mode)
{
extern long step;
if(mode)
{
save_configuration(mode) ;
save_inform(fn);
}
}
//====================================//
void add_info()
{
extern double  lx,ly,lz,a0;
fprintf(stderr,"\n \n \t Lattice constants: a0(0)=%g  a0(T)=%g",a00,a0);
fprintf(stderr,"\n \t    Lattice box:  [X Y Z] =%g x %g x %g ",lx,ly,lz);
}
//====================================//
void aviz_layer(char *fn)
{
extern double r[Atoms][3];
extern   layers[NUMBER_OF_LAYERS][Atoms];
FILE * fd;
int  j,i,k;
char *color[]=
{"a1","a2","a3","a4","a5",
"a6","a7","a8","a9","b1",
"b2","b3","b4","b5","b6",
"b7","b8","b9","c1","c2",
"c3","c4","c5","c6","c7",
"c8","c9","d1","d2","d3"} ;
if((fd=fopen(fn,"w"))!=NULL)
{ fprintf(fd," %d ", Atoms);
switch (surface)
{
case 0:
fprintf(fd,"\n ### Surface simulations (001) ");
break;
case 1:
fprintf(fd,"\n ### Surface simulations (011) ");
break;
default :
 fprintf(fd,"\n ### Surface simulations (111) ");
break;
}
for (j=0;j<FROZEN_ATOMS;j++)
fprintf(fd,"\n Cu %lf  %lf  %lf ", r[j][0],r[j][1],r[j][2]);
for(i=0;i<NUMBER_OF_LAYERS;i++)
for (j=1;j<=layers[i][0] ; j++)
{
k= layers[i][j];
fprintf(fd,"\n %s %lf  %lf  %lf ",color[i],r[k][0],r[k][1],r[k][2]);
}
fclose(fd);
}
else  { fprintf(stderr,"I can't open the file %s !!!",fn); exit(0);}
}
 *************** "Makefile " *****************
```

The Makefile for complitation and creation an executable file vanadium

```
INCS=initdata.h func_cp.h constants.h potential.h
CC=gcc -O3
LIBS=-lm
OBJS= measurements.o equilibration.o layers.o temp_prof.o bcc_setup.o compute_f.o density_profile.o evap_pto.o initialization.o
md.o pair_correlation.o predictor_corrector.o   read_save.o   build_neigh_list.o  order_parameter.o   diffusion.o
vanadium: $(OBJS) $(INCS)
        $(CC) $(OBJS) -o vanadium   $(LIBS)
order_parameter.o: order_parameter.c  $(INCS)
               $(CC) -c  order_parameter.c
pair_correlation.o: pair_correlation.c  $(INCS)
               $(CC) -c  pair_correlation.c
bcc_setup.o: bcc_setup.c  $(INCS)
               $(CC) -c  bcc_setup.c
build_neigh_list.o: build_neigh_list.c  potential.h   $(INCS)
               $(CC) -c  build_neigh_list.c
density_profile.o: density_profile.c $(INCS)
               $(CC) -c  density_profile.c
compute_f.o: compute_f.c $(INCS)
               $(CC) -c  compute_f.c
evap_pto.o: evap_pto.c $(INCS)
```

```
            $(CC) -c  evap_pto.c
initialization.o: initialization.c $(INCS)
            $(CC) -c  initialization.c
predictor_corrector.o:  predictor_corrector.c $(INCS)
            $(CC) -c  predictor_corrector.c
read_save.o: read_save.c $(INCS)
            $(CC) -c  read_save.c
md.o: md.c $(INCS)
            $(CC) -c  md.c
diffusion.o: diffusion.c $(INCS)
            $(CC) -c  diffusion.c
temp_prof.o:  temp_prof.c $(INCS)
            $(CC) -c   temp_prof.c
layers.o: layers.c $(INCS)
            $(CC) -c   layers.c
equilibration.o: equilibration.c $(INCS)
            $(CC) -c   equilibration.c
measurements.o: measurements.c $(INCS)
            $(CC) -c   measurements.c
//************** The m-script file "setup_surf_geom.m"************//
a0=1.0;
 %Type of the cell:  0-> [001]; 1-> [011]; 2-> [111];
keys=1;
keys=input('Type of the cell:  0-> [001]; 1-> [011]; 2-> [111]');
% Draw (1) or not to draw(0) the super cell points
draw_super_cell=0;
% Draw (1) or not to draw(0) the xy,xz,yz projections
draw_sections=0;
if(keys==0)
%%%%%%%%%%%%% [001] %%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=eye(3); B=inv(A);
Nx=5; Ny=5; Nz=5;
Lx=a0*Nx; Ly=a0*Ny; Lz=a0*Nz;
%%%%%%%%%%%%% [001] %%%%%%%%%%%%%%%%%%%%%%%%%%%%
end

if(keys==1)
%%%%%%%%%%%%% [011] %%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=[-1./sqrt(2)  1./sqrt(2)  0
   0            0          1.0
   1./sqrt(2)  1./sqrt(2)  0];
Nx=5; Ny=5; Nz=1;
B=inv(A);
Lx=sqrt(2)*a0*Nx; Ly=a0*Ny; Lz=sqrt(1/2)*a0*Nz;
%%%%%%%%%%%%% [001] %%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
if(keys==2)
%%%%%%%%%%%%% [111] %%%%%%%%%%%%%%%%%%%%%%%%%%%%
A=[-1./sqrt(2.)  1./sqrt(2.)  0
   -1./sqrt(6.) -1./sqrt(6.) sqrt(2./3.)
   1./sqrt(3.)   1./sqrt(3.) 1./sqrt(3.0)];
```

```
B=inv(A);
Nx=6; Ny=6; Nz=18;
Nx=input('Input the number of Nx=Ny atoms  ');
Ny=Nx;
Nz=input('Input the number of Nz atoms  ');
Lx=(a0*sqrt(2))*Nx; Ly=(a0*sqrt(3/2))*Ny; Lz=a0/(2*sqrt(3))*Nz;
%%%%%%%%%%%% [111] %%%%%%%%%%%%%%%%%%%%%%%%%%%
end
% Define dimension of the new surface:
Q=[0  0  0
  Lx 0  0
  0  Ly 0
  0  0  Lz
  Lx Ly 0
  0  Ly Lz
  Lx 0  Lz
  Lx Ly Lz];

  % Change the frame and find: the Limits

  Xmax=-1000000;Xmin=1000000;    Ymax=-1000000;Ymin=1000000;    Zmax=-1000000;Zmin=1000000;
  QQ=[];
  for i=1:8
    x=Q(i,1);y=Q(i,2);z=Q(i,3);
    xx=B(1,1)*x+B(1,2)*y+B(1,3)*z;
    yy=B(2,1)*x+B(2,2)*y+B(2,3)*z;
    zz=B(3,1)*x+B(3,2)*y+B(3,3)*z;
    if(xx>Xmax)Xmax=xx;end
    if(yy>Ymax)Ymax=yy;end
    if(zz>Zmax)Zmax=zz;end
    if(xx<Xmin)Xmin=xx;end
    if(yy<Ymin)Ymin=yy;end
    if(zz<Zmin)Zmin=zz;end

    QQ=[QQ;[xx,yy,zz]];
 end
 %Now, let's builda b.c.c crystal
 ix=floor((Xmax-Xmin)/a0+0.5);
 iy=floor((Ymax-Ymin)/a0+0.5);
 iz=2*floor((Zmax-Zmin)/a0+0.5);
r=[];
pa=1;
for i=0:ix
for j=0:iy
for  k=0:iz+2
r(pa,1)=Xmin+i*a0;
r(pa,2)=Ymin+j*a0;
r(pa,3)=Zmin+k*a0/2;
if(rem(k,2)==0)
  r(pa,1)=r(pa,1)+a0/2;
  r(pa,2)=r(pa,2)+a0/2;
end
pa=pa+1;
end
end
end
%Let's cut of the sample now
s=0;
R=[];RR=[];
 epss=-0.00001;
 for i=1:pa-1
     x=r(i,1);y=r(i,2);z=r(i,3);
     xx=A(1,1)*x+A(1,2)*y+A(1,3)*z;
      yy=A(2,1)*x+A(2,2)*y+A(2,3)*z;
      zz=A(3,1)*x+A(3,2)*y+A(3,3)*z;
      %check the limits Lx,Ly,Lz
      if(xx >= epss  & xx <Lx+epss)
      if(yy >= epss  & yy <Ly+epss)
      if( zz>= epss  & zz <Lz+epss)
            RR=[RR;[r(i,1),r(i,2),r(i,3)]];
            R=[R;[xx,yy,zz]];
            s=s+1;
         end
      end
   end end
 % Save the configurations:
 fd=fopen('r00.log','w');
 for i=1:s
 fprintf(fd,'%4.8f  %4.8f  %4.8f  \n',R(i,1),R(i,2),R(i,3));
 end
 fclose(fd);
```

The MD program (for Parinello-Rahman Bulk simulations) accepts an input file "initdata.h" where the size of the computational cell and all the relevant parameters (which are not changed frequently) are assigned.

```
#ifndef INITDATA_H
#define INITDATA_H


#define a0       3.0399            /* lattice parameter: spacing for Va at T=300K */
#define size     18                 /* its a double size of the computational cell */


#define Interstitials       6
#define Vacancies        0
#define Atoms       1464       /* Atoms:  size*size*size/4+Interstitials  - Vacancies*/
#define NDIM              3
```

```
#define N_neigh_step      100
#define  press  0.0         /* setpoint pressure*/

#define Q    0.01
#define W    9.0
#define dt   0.001

#define step_backup      100000
#define save_meas_step   200
#define sizeHistRdf       200       /*Grid for rdf functions*/

/*contants*/
#define pi   3.14159265358979
#define kb   0.0000862
# define M 16383                        /* Size of a random numbers array */
#endif
#ifndef  FUNCTIONS_H
#define  FUNCTIONS_H

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

  #define SignR(x,y) (((y) >= 0) ? (x) : (- (x)))
  #define Sqr(x) ((x) * (x))

  typedef double mat_mt[3][3];
  typedef double mat_c[7][7];

  typedef double ** matrix;
  typedef double * vector;
  typedef double  real;
  typedef int ** matrix_int;
  typedef int * vector_int;

typedef struct
 {
  real av,av2;
  real sig,mean;
  long step;

 } meas_list;

typedef struct
 {
   int   mode, Num_aviz,aviz_frames, Interval, Num_meas;
   long  Num_eql_step,Num_meas_step,Step;
   int    seed_num, atoms;
   double   temp,Press;
 } input_list;

typedef struct
 {
  long countRdf;
  real   deltaRdf;
 } rdf_list;


double Det(mat_mt a);


void SaveOrdFFT(vector order, real delta,input_list *data);
void CalcOrderFFT(input_list *data);
void EvalOrd(input_list *data);
void SaveOrdDis(vector order, input_list *data);
/*Miscellaneous*/
void printR(char *fn, real num);

/*Rdf functions*/
void  EvalRdf (vector histRdf, rdf_list *rdf, int mode);
real Min3R (real x1, real x2, real x3);
real Max3R (real x1, real x2, real x3);
void PrintRdf (char *fn, vector histRdf, rdf_list *rdf);
int RestoreRdfBackUp(char *fn, vector histRdf, rdf_list *pt );
void BackUpRdf(char *fn, vector histRdf, rdf_list *pt );

/*Basic functions*/
void SingleStep();
void SaveStatics(char *fn , meas_list *m );
void Measure(int mode, meas_list *m,  input_list *input);

void SetInitParam(input_list *input);
void InitSimulation(int code, input_list *data );

void randinit(long seed);
void Velocity_Init(input_list *p);
void init(int code,input_list *data );
void CalcOrder();
void read_input_list(  char *fn, input_list *data);


    /*Matrix Allocation: allocation.c*/
  matrix AllocMatR( int size1, int size2);
  void FreeMatR(matrix a, int size1);
  matrix_int AllocMatI (int size1, int size2) ;
  void FreeMatI(matrix_int a, int size1);
  vector AllocVecR (int size1);
  void FreeVecR(vector v);
  vector_int AllocVecI (int size1);
  void FreeVecI(vector_int v);
  long *AllocVecL (int size1) ;


void SaveData( char *fn, long step, real variable );

/* Backup functions*/
void SaveRV();
void ReadRV();
void SaveMatH(char *fn, mat_mt H);
void ReadMatH(char *fn, mat_mt H);
void ReadParam(char *fn);
void SaveParam( char *fn );
void set_inform(char *fn, input_list *p);
void BackUp(input_list *input);
void RestoreBackUp(input_list *input);






void save_for_nvt(void);
double determ(double x00,double x01,double x10,double x11);
void locate_interstitials(void);
void setup_interstitials(void);

real CalcTemp();
int comp_nums(const double *num1, const double  *num2);
void display_nums(double  *array, int count);
int find_num(int i, int j,int k);


void setup_vacancies(void);

 void getch(char *fn);
//Mon Nov 19 12:44:18
 void write_rv(void);

//Defect setup
void setup_vacancies_ordered(void);
void setup_interstit_ordered(void);
void setup_vacancies_random(void);
void setup_interstit_random(void);
void interstit_setup(void);

//Fri Oct 12 23:25:58 2001
void velocity_init(input_list *p);
void save_to_file(int mode);
void save_EPTVOh(void);
double my_abs(double x);

void compute_f();
void cry_setup();
void defects();

void save_h();
void read_h();
void scale_vel();
void read_from_file();
void build_nei_list();
void predictor();
void corrector();
void evalPT();
void leapfrog();
void scaled_frame();
void real_frame();
void init_var();
void predictorP();
void correctorP();
void rahman();
void parinello();
void init();
void accumulate_h();
void comp_c();

/*matrix functions*/
void print_mat();
void mat_mul();
void mat_sca_mul();
void mat_add();
void transpose();
double det();
void invert();
void mat_vec_mul();
void inv();
void  Aviz(char *fn);
void init_h(mat_mt a);

// Density
double density(double pr_dis);
double density_dot(double pr_dis);
double density_dot_dot(double pr_dis);
// Phi potential
double phi_pot(double pr_dis);
double phi_pot_dot(double pr_dis);
double phi_pot_dot_dot(double pr_dis);
//F_potential
double F_pot(double);
double F_pot_dot(double);
double F_pot_dot_dot(double x);

#endif
#include "initdata.h"
```

```
#include "functions.h"


/* Ziff part of random generator*/
long   ra[M+1], nd;


int  MODE;

double r[Atoms][3];
double v[Atoms][3];
double f[Atoms][3];
double f1[Atoms][3];
double f2[Atoms][3];
double r1[Atoms][3];
double v1[Atoms][3];


int nei[Atoms][300];
double n[Atoms];


double temp;
double s_temp,s_press;
double potE,kinE,virsum;
double op;
mat_mt pten;
mat_mt h,hdot,hddot;
mat_mt h_zero;

double h1vec[9],hd1vec[9],hdd1vec[9],hdd2vec[9];
double ksi;
double vol,l;


int main()
{
  long i,j,Step0,meas_step=0;
  char fn[30];
  int meas_param=4;/*vol,press,temp,ord*/


  input_list input;
  meas_list *meas;
  rdf_list rdf;
  vector  histRdf;

 histRdf=AllocVecR(sizeHistRdf);
 meas=(meas_list *)malloc(meas_param*sizeof(meas_list));

 /* Get the parameters */
```

```
read_input_list("input.list", &input);

/*Set the parameters*/
SetInitParam(&input);
InitSimulation(MODE, &input);


 build_nei_list();
 compute_f();
 rahman();


 Step0=input.Step;
  while( MODE < 3 )
 {

 for(j=0;j<N_neigh_step;j++)
      SingleStep(h,hdot,hddot);


 build_nei_list();
 input.Step+=N_neigh_step;

 if(input.Step % step_backup == 0)
                              {
                              BackUp(&input);
                              BackUpRdf("rdf.back",histRdf,&rdf);
                              }
 if(MODE==1)
   { /*Equilibration*/
      if( input.Step-Step0 > input.Num_eql_step)
   {MODE=2;input.mode=MODE; Step0=input.Step;}
   }
 else
   {  /*Measurements*/

   if( input.Step-Step0 > input.Num_meas_step)MODE=3; /*exit the loop*/
   if(input.Step%input.Interval==0 )
   {
   if(meas_step==0)
                                                      {/*Initialize*/
               Measure(0, meas,&input);  /*volume, temp, press, ord*/
               EvalRdf (histRdf, &rdf, 0);   /*radial dist. func.*/
              }
   else if(meas_step%save_meas_step==0 )
               { /*Statitstics Output*/

fprintf(stderr,"\n energy=%lf volume=%lf pressure=%lf",(potE+kinE)/Atoms,det(h)/Atoms,s_press);
fprintf(stderr,"\n step=%d order=%lf temp=%lf",input.Step,op,s_temp);

                Measure(2, meas,&input);
                EvalRdf (histRdf, &rdf, 2);
                CalcOrderFFT(&input);
                }
    else
               { /*Just measure*/
                Measure(1, meas,&input);
                EvalRdf (histRdf, &rdf, 1);
                EvalOrd(&input);   /*order dist.*/

               }
     ++meas_step;
    }/*measurements are allowed*/

     }

if(input.Step % input.aviz_frames ==0)
{
              sprintf(fn,"van_%ld.xyz",input.Step);
              real_frame(r,Atoms);
              Aviz(fn);
              scaled_frame(r,Atoms);
}


 }/*end of the main loop*/






/*Statitstics*/
CalcOrderFFT(&input);
Measure(2, meas,&input);
EvalRdf (histRdf, &rdf, 2);
BackUp(&input);
BackUpRdf("rdf.back",histRdf,&rdf);
FreeVecR(histRdf);
fprintf(stderr,"\n Happy End ");
return(0);
}




/***************************************************************************
                       basis.c  -  description
                          ——————-
    begin                : Sat Jan 11 2003
    copyright             : (C) 2003 by slava
```

```
   email              : phsorkin@technion.ac.il
 *************************************************************************/
#include "initdata.h"
#include "functions.h"

void SingleStep(h,hdot,hddot)
{
   predictor();
        compute_f();
        evalPT();
        predictorP(h,hdot,hddot);
   rahman(h);
        parinello();
        correctorP(h,hdot,hddot);
        corrector();
}

/**********************************/
 void BackUp(input_list *input)
{
  extern  double r[Atoms][3],v[Atoms][3];
  extern  mat_mt h,hdot;
  extern double ksi;

  real_frame(r,Atoms);
  real_frame(v,Atoms);

  SaveRV();
  SaveMatH("hr.log",h);
  SaveMatH("hv.log",hdot);
  SaveParam("param.log");

  scaled_frame(r,Atoms);
  scaled_frame(v,Atoms);
  set_inform("input.list", input);
}

 /**********************************/
 void RestoreBackUp(input_list *input)
 {
  extern  double r[Atoms][3],v[Atoms][3];
  extern  mat_mt h,hdot;
  extern double ksi;

  int i;

  ReadMatH("hr.log",h);
  ReadMatH("hv.log",hdot);
  ReadRV();
  scaled_frame(r,Atoms);
  scaled_frame(v,Atoms);
  ReadParam("param.log");

  /* check temp
  fprintf(stderr,"\n temp=%lf ksi=%lf",CalcTemp(),ksi);
  */




 }
 /**********************************/
 void SetInitParam(input_list *input)
 {
 extern double temp,l;
 extern int MODE;

 temp=input->temp;
 l=a0*size/2.0;
 MODE=input->mode;
 input->atoms=Atoms;
 }

 /**********************************/
  void Measure(int mode, meas_list *m,  input_list *input)
  {

   extern double op,s_temp,s_press,vol;



   int i;
   double volume,x,x2;

   volume= vol/Atoms;

   switch( mode )
     {
       case 0 : fprintf(stderr, "Initialization" );

               for(i=0;i<4;i++)
               {
               m[i].step=0;
               m[i].av=0.;
               m[i].av2=0.;
               m[i].sig=0.;
               }

               break;

       case 2 :
               for(i=0;i<4;i++)
               {
               if(m[i].step >0)
               {
                x= m[i].av/m[i].step;
                x2= m[i].av2/m[i].step;
                m[i].sig= sqrt(fabs(x2-Sqr(x)));
                m[i].mean=x;
               }
               SaveStatics("report.txt",m);
               }
               break;


       default :

          /*— volume—*/
               m[0].step++;
               m[0].av+=volume;
               m[0].av2+=volume*volume;
               SaveData( "volume.txt", input->Step,volume);

               /*— temp—*/
               m[1].step++;
               m[1].av+=s_temp;
               m[1].av2+=s_temp*s_temp;
               SaveData( "temperature.txt", input->Step,s_temp);

               /*— press—*/
               m[2].step++;
               m[2].av+=s_press;
               m[2].av2+=s_press*s_press;
               SaveData( "pressure.txt", input->Step,s_press);

               /*— order—*/
               CalcOrder();
               m[3].step++;
               m[3].av+=op;
               m[3].av2+=op*op;
               SaveData( "order.txt", input->Step,op);

               /*— matrix—*/
               SaveHij(input);

                 break;
       }



  }
```

```
/***************************************************************************
                    alocation.c  -  description
                      ————————-
   Dynamic Memory  Allocation
   begin               : Tue Dec 17 2002
 ***************************************************************************/
# include "initdata.h"
#include "functions.h"

matrix AllocMatR( int size1, int size2)
{
int i;
matrix a;
a=malloc( size1*sizeof(vector));

for (i=0;i<size1;i++)
a[i]=malloc(size2*sizeof(real));

return(a);
}
/*==========================================*/
void FreeMatR( matrix a, int size1)
{
int i;
for (i=0;i<size1;i++)
{
//printR("i=",1.*i);
free(a[i]);
}

free(a);
}

/*==========================================*/
matrix_int AllocMatI( int size1, int size2)
{
int i;
matrix_int a;
a=calloc( size1, sizeof(vector_int));

for (i=0;i<size1;i++)
a[i]=calloc(size2,sizeof(int));

return(a);
}
/*==========================================*/
void FreeMatI( matrix_int a, int size1)
{
int i;
for (i=0;i<size1;i++)
free(a[i]);

free(a);
}
/*==========================================*/

  vector AllocVecR (int size1)
  {
   vector v;
   v = (vector) malloc (size1 * sizeof (real));
   return (v);
  }
  void FreeVecR(vector v)
  { free(v); }

/*==========================================*/
   vector_int AllocVecI (int size1) {
   vector_int v;
    v = (vector_int) malloc (size1 * sizeof (int));
   return (v);
  }
    void FreeVecI(vector_int v)
  { free(v); }
 /*==========================================*/
 long *AllocVecL (int size1) {
    long *v;
    v = (long *) malloc (size1 * sizeof (long));
    return (v);
  }
/*==========================================*/

/*==============================================*/
void PrintMatI(int size1, int size2, matrix_int beta, char *fn)
{ int i,j;
 for (i=0 ;i<size1;i++)
 for (j=0 ;j<size2;j++)
 fprintf(stderr,"\n %s[%d][%d]=[%d] ",fn,i,j,beta[i][j]);
}
/*==============================================*/
void AddMatR(int size1, int size2, matrix a,matrix b, matrix c)
{ int i,j;
 for (i=0 ;i<size1;i++)
 for (j=0 ;j<size2;j++)
 c[i][j]=a[i][j]+b[i][j];
}



#include "initdata.h"
#include "functions.h"
```

```
/*   Functions:
 void read_input_list(  char *fn, input_list *data)   Line 4
 void set_inform(char *fn, input_list *p)              Line 179
*/



/*===== read the input list   ===========*/
void read_input_list(  char *fn, input_list *data)
{
       char buffer[BUFSIZ];
       char * ptr;
       FILE * in;

       printf(”Reading an input file \” %s \”: \n \n ”,fn);

       if ((in = fopen(fn, ”r”))) {

/*============== Read 1st line===== */
       fgets( buffer, BUFSIZ, in );

              ptr = buffer;
              while ( (*ptr) != ’:’)
                     ptr++;
         ptr++;

         data->mode= atoi( ptr );
              printf(”mode %d\n”, data->mode);

/*========= Read 2nd line ========*/

   fgets( buffer, BUFSIZ, in );

              ptr = buffer;
              while ( (*ptr) != ’:’)
                     ptr++;

       ptr++;

              data->temp = atof( ptr );
              printf(” temp %f\n”, data->temp);

              /*===== Read 3hd line========= */
              fgets( buffer, BUFSIZ, in );

              ptr = buffer;
              while ( (*ptr) != ’:’)
                     ptr++;

     ptr++;

         data-> Num_eql_step= atol( ptr );
```

```
            printf(" num equil step %ld\n", data-> Num_eql_step);

/* ========Read 4th line========= */
            fgets( buffer, BUFSIZ, in );

            ptr = buffer;
            while ( (*ptr) != ':')
                  ptr++;

  ptr++;

      data-> Num_meas= atoi( ptr );
            printf(" number meas %d\n",data-> Num_meas );

 /* Read 5th line */
            fgets( buffer, BUFSIZ, in );

            ptr = buffer;
            while ( (*ptr) != ':')
                  ptr++;

  ptr++;

      data-> Interval= atoi( ptr );
            printf(" interval  %d\n",data-> Interval );
      data->Num_meas_step =  data->  Interval * data-> Num_meas;

 /* Read 6th line */
            fgets( buffer, BUFSIZ, in );

            ptr = buffer;
            while ( (*ptr) != ':')
                  ptr++;

  ptr++;

      data-> Num_aviz= atoi( ptr );
            printf(" AViz frames %d\n",data-> Num_aviz );
 data->aviz_frames= (int)1.0*data->Num_meas_step/data->Num_aviz;
   /* Read 7th line */
            fgets( buffer, BUFSIZ, in );

            ptr = buffer;
            while ( (*ptr) != ':')
                  ptr++;

  ptr++;

      data-> Step= atol( ptr );
            printf(" initial step %ld\n",data-> Step);
   /* Read 11th line */
             fgets( buffer, BUFSIZ, in );

             ptr = buffer;
             while ( (*ptr) != ':')
                    ptr++;
             ptr++;

        data-> seed_num= atoi( ptr );




             fclose(in);
       }
       else {
             printf("Error: file \"%s\" does not exist -> exiting.\n", fn);
fprintf(stderr,"\n \t I tried to guess the initial parameters");
fprintf(stderr,"\n \t Check them and start again ");

        if((in=fopen(fn,"w"))!=NULL)
        {
      fprintf(in," mode        :%d    ",0);
      fprintf(in,"\n temperature: %d   ",400);
      fprintf(in,"\n num_eq_step: %d   ",3000);
      fprintf(in,"\n num_of_meas: %d   ",100);
      fprintf(in,"\n meas_interv: %d ",1000);
      fprintf(in,"\n num_AViz_fr: %d ",10);
      fprintf(in,"\n step0       :%d   ",0);
      fprintf(in,"\n seed <20   : %d   ",1);
      fclose(in);
      exit(0);
     }
       }
}

//=========================================//


void set_inform(char *fn, input_list *p)
{
   FILE *outd;
        if((outd=fopen(fn,"w"))!=NULL)
        {
    fprintf(outd," mode        : %d  ", p->mode);
    fprintf(outd,"\n temperature: %lf  ", p->temp);
    fprintf(outd,"\n num_eq_step: %ld ", p->Num_eql_step);
    fprintf(outd,"\n num_of_meas: %d ", p->Num_meas);
    fprintf(outd,"\n meas_interv: %d ", p->Interval);
    fprintf(outd,"\n num_AViz_fr: %d ", p->Num_aviz);
    fprintf(outd," \n step0       : %ld ", p->Step);
```

```
     fprintf(outd," \n seed<20     : %d \n ", p->seed_num);
     fclose(outd);
    system("date  >inp1;cat input.list inp1 >inp2;mv inp2 input.list ;rm inp1; ");
     }
     else {printf(" \n\n Can not open !!!!!!!!");exit(0);}
}
#include "initdata.h"
#include "functions.h"
#define NewRandomInteger (ra[(nd++)&M] = ra[(nd-471)&M]^ra[(nd-1586)&M]^ra[(nd-6988)&M]^ra[(nd-9689)&M])
#define NewRandomFloat (NewRandomInteger/2147483648.0)

/*——————————————————————*/
void cry_setup()
{
extern  double r[Atoms][3];

int i,j,k,pa;
double a02;//Initial coordinates
FILE *fid;

a02=a0/2;
pa=0;

for (k=0;k< size;k++){
        for (j=0;j< (size/2);j++){
                for(i=0;i<(size/2);i++){
               r[pa][0]=a0*i;
               r[pa][1]=a0*j;
               r[pa][2]=a02*k;

               if(k%2!=0){
               r[pa][0]+=a02;
               r[pa][1]+=a02;}

                           pa++;
                }
        }
}
}

/*—————————————————————*/
void velocity_init(input_list *p){
extern  double v[Atoms][3],temp;
extern long    ra[M+1], nd;
int j,k;
double sum[3];
double x1,x2,x3,y1,y2,y3,Sf;

long seeds[]={ 606842583, 485982468,891298966, 762096833,
    556467665, 118503643, 821407164,444703364, 19543234,
    39193703,  921812970, 73820724, 17626614,  405706210,
    935469691,  816904431, 410270205, 893649533, 157891307,
    3568130 ,57689780, 1099068};

fprintf(stderr,"\n temp=%g",temp);
 randinit(seeds[p->seed_num]);

 for (j=0;j<3;j++)sum[j]=0.;



  /*srand( (unsigned)time( NULL ) );  */
      /* Seed the random-number generator with current time so that
  * the numbers will be different every time we run. */



  /* Generate gaussian distribution */
      for (j=0;j<Atoms;j++)
      {

       /*  RAND_MAX is the Max integer random number
    x1=((double)rand())/RAND_MAX;
    x2=((double)rand())/RAND_MAX;
    x3=((double)rand())/RAND_MAX; */

    x1=NewRandomFloat;
    x2=NewRandomFloat;
    x3=NewRandomFloat;

         /*The Box Muller method*/
   y1=sqrt(-log(x1))*cos(2*pi*x2);
   y2=sqrt(-log(x2))*cos(2*pi*x1);
   y3=sqrt(-log(x3))*cos(2*pi*x1);




  v[j][0]=y1;
  v[j][1]=y2;
  v[j][2]=y3;

  for (k=0;k<3;k++)sum[k]+=v[j][k];

      }



 /* Center of mass velocity */
        for (k=0;k<3;k++) sum[k]/=(Atoms);




 for (j=0;j<Atoms;j++)
         for (k=0;k<3;k++)
     v[j][k]-=sum[k];
  Sf=0.;
  for (j=0;j<Atoms;j++)
  Sf+=v[j][0]*v[j][0]+v[j][1]*v[j][1]+v[j][2]*v[j][2];



        /* Let's rescale the velocities */
  Sf=sqrt(3*kb*temp*(Atoms)/Sf);

  for (j=0;j<Atoms;j++)
          for (k=0;k<3;k++)
                 v[j][k]*=Sf;




  /*Now check it and be sure, that T appropriate and Vcm=0 */
    Sf=0;

    for (j=0;j<3;j++)sum[j]=0.;

for (j=0;j<Atoms;j++)
{
      Sf+=v[j][0]*v[j][0]+v[j][1]*v[j][1]+v[j][2]*v[j][2];
      for (k=0;k<3;k++){ sum[k]=sum[k]+v[j][k];}
}



          fprintf(stdout,"\n Center_of_Mass Velocity \n Vx=%lf Vy=%lf  Vz=%lf ",sum[0],sum[1],sum[2]);
    fprintf(stdout,"\n temp=%lf ", Sf/(3*Atoms*kb));

}



/**********************************************/
void randinit(long seed)
{          /* Initialization of a random massive  */
        double a, ee = -1 + 1/2147483648.0;
        long i;
        extern long nd, ra[M+1];

        a = seed/2147483648.0;
        for (nd = 0; nd <= M; nd++)
        {
             a *= 16807;
             a += ee * (long)(a);
             if (a >= 1) a += ee;
             ra[nd] = (long) (2147483648.0 * a);
        }
        nd = M;
        for(i = 0; i<1000000; i++)
         NewRandomInteger;
}
/***************************************************************************
                    defect_setup.c  -  description
                       ——————————-
   begin              : Mon Oct 8 2001
   copyright           : (C) 2001 by
   email              : phsorkin@phsorkin.technion
 ***************************************************************************/
#define Rcr   0.1*a0
#define Rcr2  Rcr*Rcr
#define bsize size/2
#define bsize2 bsize*bsize
#define PAtoms  bsize*bsize*size

#include "initdata.h"
#include "functions.h"



/*****************************************************************************/
void setup_interstitials(void)
{
extern double r[Atoms][3],l;

double a02,a04,R[PAtoms][4];
int particle,point[5][2];
int bstep,rest;
int i,j,k,pa,m;

FILE *fd;

if(Interstitials>5*size)
{
fprintf(stderr,"\n You've exceeded the Maximal Number of Interstitials =%d",5*size);
exit(0);
}

for (k=0;k<PAtoms;k++)R[k][3]=2.;

bstep=(int)(bsize/4);

point[0][0]=(int)(bsize/2.0)-bstep;
point[0][1]=(int)(bsize/2.0)-bstep;

point[1][0]=(int)(bsize/2.0)+bstep;
point[1][1]=(int)(bsize/2.0)-bstep;

point[2][0]=(int)(bsize/2.0)+bstep;
point[2][1]=(int)(bsize/2.0)+bstep;
```

```
point[3][0]=(int)(bsize/2.0)-bstep;
point[3][1]=(int)(bsize/2.0)+bstep;

point[4][0]=(int)(bsize/2.0);
point[4][1]=(int)(bsize/2.0);

// Setup the perfect b.c.c lattice
a02=a0/2;
a04=a0/6;

pa=0;

for (k=0;k<size;k++){
    for (j=0;j< bsize;j++){
        for(i=0;i<bsize;i++){

        R[pa][0]=a0*i;
        R[pa][1]=a0*j;
        R[pa][2]=a02*k;

        if(k%2!=0){
        R[pa][0]+=a02;
        R[pa][1]+=a02;
                                                        }

              pa++;
        }
    }
}



pa=(int)1.0*Interstitials/size;
rest=Interstitials-pa*size;



if(pa >= 1)
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[0][0];j=point[0][1];}
else   { i=point[2][0];j=point[2][1];}

/* Here is a vacancy*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
//if(k<size-1){particle=find_num(i,j,k+1);if(R[particle][3]){R[particle][3]=1.;}}
//if(k>0){particle=find_num(i,j,k-1);if(R[particle][3]){R[particle][3]=1.;}}

}

}

 if(pa >= 2)
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[2][0];j=point[2][1];}
else   { i=point[0][0];j=point[0][1];}

/* Here is a Interstitial*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}



}

}
if(pa >= 3)
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[1][0];j=point[1][1];}
else   { i=point[3][0];j=point[3][1];}

/* Here is a Interstitial*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
}

}
if(pa == 4)
{
for (k=0;k<size;k++)
```

```
{
if(k%2){ i=point[3][0];j=point[3][1];}
else   { i=point[1][0];j=point[1][1];}


/* Here is a Interstitial*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
}

}

if(Interstitials>size){

pa=(int)size/rest;

for(m=0;m<rest;m++)
{
k=m*pa;
i=point[4][0];
j=point[4][1];

particle=find_num(i,j,k);R[particle][3]=0;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}

}
} //if(Interstitials<size)

/**********************************************************************/
/**********************************************************************/
if(Interstitials<size)
{

pa=(int)size/rest;
//printf("\n pa=%d",pa);

for(m=0;m<rest;m++)
{
k=m*pa;
```

```
//printf("\n k=%d \n\t",k);

if(m%2){ i=point[3][0];j=point[3][1];}
else   { i=point[1][0];j=point[1][1];}

/* Here is a Interstitial*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
}

} //if(Interstitials<size)

/**********************************************************************/
/**********************************************************************/

if(fd=fopen("conf_init.xyz","w"))
{

/**********************************************************************/
/**********************************************************************/

 fprintf(fd," %d \n",Atoms);
 fprintf(fd,"##Cijkl simulations with Defects\n");

pa=(int)PAtoms;
for(i=0;i<PAtoms;i++)
{
r[i][0]=R[i][0];  r[i][1]=R[i][1]; r[i][2]=R[i][2];

if(R[i][3])
{
if(R[i][3]==1.0)

{fprintf(fd,"Al %lf  %lf  %lf \n", r[i][0],r[i][1],r[i][2]);}//NN neighbors
else
{fprintf(fd,"Va %lf  %lf  %lf \n", r[i][0],r[i][1],r[i][2]);}//Others

} //  if(R[i][3])

else
{ /*Setup the interstitials*/

r[pa][0]=R[i][0]+a02/2;
r[pa][1]=R[i][1]+a02/2;
r[pa][2]=R[i][2]+a04/2;
fprintf(fd,"Fe %lf  %lf  %lf \n", r[i][0],r[i][1],r[i][2]);
fprintf(fd,"Cu %lf  %lf  %lf \n", r[pa][0],r[pa][1],r[pa][2]);
pa++;
}//Setup the interstitials


} //for cycle
 fclose(fd);
}
else
  {
        fprintf(stderr,"I can't open the file conf_init.xyz ");
  exit(0);
  }
//system("aviz  conf_init.xyz");



  }


 /*************************/


void setup_interstit_random(void)
{
extern double r[Atoms][ 3],l;
double a02;
int place[(size/2)][(size/2)][2*(size/2)];
int t1,t2,t3,index=0;
int i,j,k,pa;
a02=a0/2;
pa=0;
// Setup the perfect b.c.c lattice
for (k=0;k<size;k++){
        for (j=0;j< (size/2);j++){
                for(i=0;i<(size/2);i++){

               r[pa][0]=a0*i;
               r[pa][1]=a0*j;
               r[pa][2]=a02*k;

               if(k%2!=0){
               r[pa][0]+=a02;
               r[pa][1]+=a02;}

                           pa++;
                }
        }
```

```
}
//Add the selfinterstitials  randomly

for(t1=0;t1<(size/2);t1++)
for(t2=0;t2<(size/2);t2++)
for(t3=0;t3<2*(size/2);t3++)
 place[t1][t2][t3]=1;

    srand( (unsigned)time( NULL ) );
       /* Seed the random-number generator with current time so that
   * the numbers will be different every time we run.
   */
while(index<Interstitials)
{
t1=(int)((size/2-1)*((double)rand())/RAND_MAX+0.5);
t2=(int)((size/2-1)*((double)rand())/RAND_MAX+0.5);
t3=(int)((size/2-1)*((double)rand())/RAND_MAX+0.5);



//fprintf(stderr,"\n t1=%d t2=%d t3=%d ",t1,t2,t3);
if(place[t1][t2][t3])
{
place[t1][t2][t3]=0;
r[Atoms-Interstitials+index][0]=a0*t1+a02;
r[Atoms-Interstitials+index][1]=a0*t2+a02;
r[Atoms-Interstitials+index][2]=a0*t3;
index++;
}
}//while(index<Interstitials)

Aviz("Defect.xyz");
//system("aviz -ar Defect.xyz");



}

/***************************************************/
/***************************************************/
/***************************************************/

/* Setup Interstitials ordered: Another Version */
void interstit_setup(void)
{
extern double r[Atoms][ 3],l;
int i,j,k,m,index,N,pa;
int nx,ny,nz,odd=0;
double lx,ly,lz;
double x,y,z,xc,yc,zc,dist2,ex,ey,ez,a02;

a02=a0/2;
pa=0;
```

```
// Setup the perfect b.c.c lattice
for (k=0;k<size;k++){
        for (j=0;j< size/2;j++){
                for(i=0;i<size/2;i++){

               r[pa][0]=a0*i;
               r[pa][1]=a0*j;
               r[pa][2]=a02*k;

               if(k%2!=0){
               r[pa][0]+=a02;
               r[pa][1]+=a02;}

                           pa++;
                }
        }
}

//Let's analyse the  Interstitals structure
if(Interstitials%2)
{
odd=1;
// Set the coordinates  in the center of the box
x=y=z=l/2;
// Check if there is no overlapping


for(i=0;i<Atoms-Interstitials;i++)
{
dist2=(x-r[i][0])*(x-r[i][0])+(y-r[i][1])*(y-r[i][1])
+(z-r[i][2])*(z-r[i][2]);
if(dist2<Rcr2)
{
dist2=sqrt(dist2);
if(dist2)
{
ex=(r[i][0]-x)/dist2;
ey=(r[i][1]-y)/dist2;
ez=(r[i][2]-z)/dist2;

x+=Rcr*ex;
y+=Rcr*ey;
z+=Rcr*ez;
}
else
{
x+=Rcr;
y+=Rcr;
z+=Rcr;
}
```

```
} // if(dist2<Rcr)
}// for(i=0;i<Atoms;i++)
r[Atoms-Interstitials][0]=x;
r[Atoms-Interstitials][1]=y;
r[Atoms-Interstitials][2]=z;
} //if(Interstitals%2)


//Now let's divide the box onto small boxes

nx=ny=nz=1;
N=Interstitials;
if(odd)N–;

while(1)
{
if(N%2==0){nx*=2;N/=2;}
else {        nx*=N;break;   }
if(N%2==0){   nz*=2;N/=2;  }
else {     nz*=N;break;   }
if(N%2==0) {     ny*=2;N/=2; }
else  {        ny*=N;break;  }
  }//while


index=0;
if(odd)index=1;//Take into account an odd atom

lx=l/nx;
ly=l/ny;
lz=l/nz;

xc=0.5*lx;
yc=0.5*ly;
zc=0.5*lz;

for(i=0;i<nz;i++){
for(j=0;j<nx;j++){
for(k=0;k<ny;k++){

x=xc+j*lx;
y=yc+k*ly;
z=zc+i*lz;

for(m=0;m<Atoms-Interstitials;m++)
{
dist2=(x-r[m][0])*(x-r[m][0])+(y-r[m][1])*(y-r[m][1])
+(z-r[m][2])*(z-r[m][2]);
if(dist2<Rcr2)
{
dist2=sqrt(dist2);
if(dist2)
{
ex=(r[m][0]-x)/dist2;
ey=(r[m][1]-y)/dist2;
ez=(r[m][2]-z)/dist2;

x+=Rcr*ex;
y+=Rcr*ey;
z+=Rcr*ez;
}
else
{
x+=Rcr;
y+=Rcr;
z+=Rcr;
}

} // if(dist2<Rcr)
}// for(i=0;i<Atoms;i++)

r[Atoms-Interstitials+index][0]=x;
r[Atoms-Interstitials+index][1]=y;
r[Atoms-Interstitials+index][2]=z;
index++;
}
}
}

 Aviz(”Defect.xyz”);
//system(”aviz Defect.xyz”);

}

/////////////////////////////////////////////////////////////////


void setup_interstit_ordered(void)
{
 extern double r[Atoms][ 3],l;
double a02;
int i,j,k,pa,m;
a02=a0/2;
pa=0;
// Setup the perfect b.c.c lattice
for (k=0;k<size;k++){
        for (j=0;j< (size/2);j++){
                for(i=0;i<(size/2);i++){

               r[pa][0]=a0*i;
               r[pa][1]=a0*j;
               r[pa][2]=a02*k;
```

```
              if(k%2!=0){
              r[pa][0]+=a02;
              r[pa][1]+=a02;}

                         pa++;
               }
        }
}
//Add the self interstitials  in order
m=(int)size*(size/2)*(size/2);
i=j=k=0;
for(pa=0;pa<Interstitials;pa++)
{
r[m+pa][0]=a0*i+a02;
r[m+pa][1]=a0*j+a02;
r[m+pa][2]=a0*k;

if(k<(size/2))k++;
else if(i<(size/2)){k=0;i++;}
else if(j<(size/2)){k=0;i=0;j++;}
else
{
fprintf(stderr,"N=%d Too many defects !!! ",Interstitials);
exit(0);
}

}

Aviz("Defect.xyz");
//system("aviz Defect.xyz");

}

////////////////////////////////////////////////////////////////////
//           VACANCIES                                            //
////////////////////////////////////////////////////////////////////

void setup_vacancies_random(void)
{
extern double r[Atoms][ 3],l;
double R[2*(size/2)*(size/2)*(size/2)][3];
int place[2*(size/2)*(size/2)*(size/2)];
int t,index=0,N;
int i,j,k,pa;
double a02;//Initial coordinates



a02=a0/2;
pa=0;
```

```
for (k=0;k<2*(size/2);k++){
        for (j=0;j< (size/2);j++){
                for(i=0;i<(size/2);i++){

  // fprintf(stderr,"\n i=%d j= %d pa=%d ",i,j,pa);

               R[pa][0]=a0*i;
               R[pa][1]=a0*j;
               R[pa][2]=a02*k;

               if(k%2!=0){
               R[pa][0]+=a02;
               R[pa][1]+=a02;}

                           pa++;
                }
        }
}

N=(int)size*(size/2)*(size/2);
for(t=0;t<N;t++)
place[t]=1;

   srand( (unsigned)time( NULL ) );
        /* Seed the random-number generator with current time so that
   * the numbers will be different every time we run.
   */
index=0;
while(index<Vacancies)
{
t=(int)((N-1)*((double)rand())/RAND_MAX+0.5);


fprintf(stderr,"\n t=%d ",t);
if(place[t])
{
place[t]=0;
R[t][0]=1000000.0;
index++;
}

}//while(index<Interstitials)

index=0;
for(t=0;t<N;t++)
{

if(R[t][0]!=1000000.0)
{
r[index][0]=R[t][0];
r[index][1]=R[t][1];
r[index][2]=R[t][2];
index++;
}

}

Aviz("Defect.xyz");
//system("aviz Defect.xyz &");



}
////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
void setup_vacancies_ordered(void)
{
extern double r[Atoms][ 3],l;
double R[2*(size/2)*(size/2)*(size/2)][3];
double a02;
int i,j,k,pa,m,index;
a02=a0/2;
pa=0;

// Setup the perfect b.c.c lattice
for (k=0;k<2*(size/2);k++){
        for (j=0;j< (size/2);j++){
                for(i=0;i<(size/2);i++){

               R[pa][0]=a0*i;
               R[pa][1]=a0*j;
               R[pa][2]=a02*k;

               if(k%2!=0){
               R[pa][0]+=a02;
               R[pa][1]+=a02;}

                          pa++;
                }
        }
}
//Add the self interstitials  in order

i=j=k=0;
for(pa=0;pa<Vacancies;pa++)
{
m=i+j*(size/2)+k*(size/2)*(size/2);
m=(int)m;
fprintf(stderr,"\n m=%d",m);
R[m][0]=-100000000;
if(k<(size/2))k++;
```

```
else if(i<(size/2)){k=0;i++;}
else if(j<(size/2)){k=0;i=0;j++;}
else
{
fprintf(stderr,"N=%d Too many defects !!! ",Vacancies);
exit(0);
}

}

index=0;
for(m=0;m<2*(size/2)*(size/2)*(size/2);m++)
{

if(R[m][0]!=-100000000)
{
r[index][0]=R[m][0];
r[index][1]=R[m][1];
r[index][2]=R[m][2];
index++;
}

}
//for(m=Atoms-Interstitials;m<Atoms;m++)
//fprintf(stderr,"\n m=%d x%lf y%lf z%lf t%d ",m,r[m][0],r[m][1],r[m][2],index);

Aviz("Defect.xyz");
//system("aviz Defect.xyz");


}

/***************************************************************/
/***************************************************************/
/***************************************************************/

void setup_vacancies(void)
{
extern double r[Atoms][3],l;

double a02,a04,R[PAtoms][4];
int particle,point[5][2];
int bstep,rest;
int i,j,k,pa,m;

FILE *fd;

if(Vacancies>5*size)
{
fprintf(stderr,"\n You've exceeded the Maximal Number of Vacancies =%d",5*size);
exit(0);
```

```
}

for (k=0;k<PAtoms;k++)R[k][3]=2.;

bstep=(int)(bsize/4);

point[0][0]=(int)(bsize/2.0)-bstep;
point[0][1]=(int)(bsize/2.0)-bstep;

point[1][0]=(int)(bsize/2.0)+bstep;
point[1][1]=(int)(bsize/2.0)-bstep;

point[2][0]=(int)(bsize/2.0)+bstep;
point[2][1]=(int)(bsize/2.0)+bstep;

point[3][0]=(int)(bsize/2.0)-bstep;
point[3][1]=(int)(bsize/2.0)+bstep;

point[4][0]=(int)(bsize/2.0);
point[4][1]=(int)(bsize/2.0);

// Setup the perfect b.c.c lattice
a02=a0/2;
a04=a0/6;

pa=0;

for (k=0;k<size;k++){
        for (j=0;j< bsize;j++){
                for(i=0;i<bsize;i++){

               R[pa][0]=a0*i;
               R[pa][1]=a0*j;
               R[pa][2]=a02*k;

               if(k%2!=0){
               R[pa][0]+=a02;
               R[pa][1]+=a02;
                                                                      }

                          pa++;
                }
        }
}


pa=(int)1.0*Vacancies/size;
rest=Vacancies-pa*size;


if(pa >= 1)
```

```
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[0][0];j=point[0][1];}
else   { i=point[2][0];j=point[2][1];}

/* Here is a vacancy*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the vacancy neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}

//if(k<size-1){particle=find_num(i,j,k+1);if(R[particle][3]){R[particle][3]=1.;}}
//if(k>0){particle=find_num(i,j,k-1);if(R[particle][3]){R[particle][3]=1.;}}

}

}

 if(pa >= 2)
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[2][0];j=point[2][1];}
else   { i=point[0][0];j=point[0][1];}

/* Here is a vacancy*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the vacancy neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}



}

}
if(pa >= 3)
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[1][0];j=point[1][1];}
else   { i=point[3][0];j=point[3][1];}
```

```
/* Here is a vacancy*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the vacancy neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
}

}
if(pa == 4)
{
for (k=0;k<size;k++)
{
if(k%2){ i=point[3][0];j=point[3][1];}
else   { i=point[1][0];j=point[1][1];}


/* Here is a vacancy*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the vacancy neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
}

}

 if(Vacancies>size){
pa=(int)size/rest;

for(m=0;m<rest;m++)
{
k=m*pa;
i=point[4][0];
j=point[4][1];

particle=find_num(i,j,k);R[particle][3]=0;

/* Here are the vacancy neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
```

```
}
} //if(Vacancies<size)

 /************************************************************/
 /************************************************************/

if(Vacancies<size)
{

pa=(int)size/rest;


for(m=0;m<rest;m++)
{
k=m*pa;


if(m%2){ i=point[3][0];j=point[3][1];}
else   { i=point[1][0];j=point[1][1];}

/* Here is a Interstitial*/
particle=find_num(i,j,k);R[particle][3]=0.;

/* Here are the Interstitial neighbors and their colors*/
if(i<bsize-1){particle=find_num(i+1,j,k);if(R[particle][3]){R[particle][3]=1.;}}
if(i>0){particle=find_num(i-1,j,k);if(R[particle][3]){R[particle][3]=1.;}}

if(j<bsize-1){particle=find_num(i,j-1,k);if(R[particle][3]){R[particle][3]=1.;}}
if(j>0){particle=find_num(i,j+1,k);if(R[particle][3]){R[particle][3]=1.;}}
}

} //if(Vacancies<size)

/*********************************************************************/


if(fd=fopen("conf_init.xyz","w"))
{

 fprintf(fd," %d \n",Atoms);
 fprintf(fd,"##Cijkl simulations with Defects\n");


pa=0;
for(i=0;i<PAtoms;i++)
{

if(R[i][3])
{
if(R[i][3]==1.0)
{fprintf(fd,"Cu %lf  %lf  %lf \n", R[i][0],R[i][1],R[i][2]);}
else
{fprintf(fd,"Va %lf  %lf  %lf \n", R[i][0],R[i][1],R[i][2]);}

r[pa][0]=R[i][0];  r[pa][1]=R[i][1]; r[pa][2]=R[i][2];
pa++;
} //  if(R[i][3])

}
 fclose(fd);
}
else
  {
        fprintf(stderr,"I can't open the file conf_init.xyz ");
  exit(0);
  }
//system("aviz  conf_init.xyz");



  }
/*****************************************************************************/
/*****************************************************************************/

int find_num(int i, int j,int k)
{
return(i+j*bsize+k*bsize2);
}
#include <stdio.h>
#include "initdata.h"
#include "functions.h"


void InitSimulation(int code, input_list *data )
{

extern double r[Atoms][3],v[Atoms][3];
extern double ksi;
extern mat_mt h;
extern MODE;

int i,j;
ksi=0.0; /*thermostat*/

 init_var();

if(code==0) /* initialization */
{
/* remove the old *.txt files */
    system("rm *.txt *.back");
    MODE=1;
    data->mode=MODE;
```

```
   if(Interstitials)
 setup_interstitials();
 /*
 setup_interstit_random();
 setup_interstit_ordered();
 */
  else if(Vacancies)
  setup_vacancies();
 /*
 setup_vacancies_random();
 setup_vacancies_ordered();
 */
 else
 cry_setup();


     Aviz("init.xyz");
     init_h(h);

   /*convert to the scaled frame */
     scaled_frame(r,Atoms);
     velocity_init(data);
   /*convert to the scaled frame */
     scaled_frame(v,Atoms);
}



   else  /*Continue simulation */
   {
      RestoreBackUp(data);
   }
}

//////////////////////////////////////////////////////////////////////

void init_var()
{
 extern double f1[Atoms][3];
 extern double f2[Atoms][3];
 extern double r1[Atoms][3];
 extern double v1[Atoms][3];
 extern mat_mt hdot,hddot;
 extern double h1vec[9],hd1vec[9],hdd1vec[9],hdd2vec[9];

int i,k;
 for(i=0;i<Atoms;i++)
 for(k=0;k<3;k++)
 r1[i][k]=v1[i][k]=f1[i][k]=f2[i][k]=0.0;
```

```
    for(k=0;k<9;k++)
    h1vec[k]=hd1vec[k]=hdd1vec[k]=hdd2vec[k]=0.;

for(i=0;i<3;i++)
{
 hdot[i][0]=hdot[i][1]=hdot[i][2]=0.;
 hddot[i][0]=hddot[i][1]=hddot[i][2]=0.;
}



}
/************************
 * init_h
 *************************/
void init_h(a)
     mat_mt a;
{
int i;
for(i=0;i<3;i++)
a[i][0]=a[i][1]=a[i][2]=0.0;

a[2][2]=a[1][1]=a[0][0]=(size/2.0)*a0;

}

/*******************
 * init_h
 ********************/
#include <math.h>
#include "initdata.h"
#include "functions.h"

#define neicutoff 17.0
void  build_nei_list()
{

extern double l;
extern double r[Atoms][3];
extern mat_mt h;
extern  int nei[Atoms][300];



  int i,j;
  double rdis[3];
  double dis;
   mat_mt G,htr;

  transpose(h,htr);
  mat_mul(htr,h,G); /*build G*/
  for(i=0;i<Atoms;i++)  nei[i][0]=0;

  for(i=0;i<Atoms;i++)
    {
  for(j=i+1;j<Atoms;j++)
        {

  rdis[0]=r[i][0]-r[j][0];
   if(rdis[0]>0.5)rdis[0]-=1.;
   if(rdis[0]<-0.5)rdis[0]+=1.;



   rdis[1]=r[i][1]-r[j][1];
   if(rdis[1]>0.5)rdis[1]-=1.;
   if(rdis[1]<-0.5)rdis[1]+=1.;

   rdis[2]=r[i][2]-r[j][2];
   if(rdis[2]>0.5)rdis[2]-=1.;
   if(rdis[2]<-0.5)rdis[2]+=1.;


        dis=rdis[0]*G[0][0]*rdis[0] +rdis[0]*G[0][1]*rdis[1] +rdis[0]*G[0][2]*rdis[2];
    dis+=rdis[1]*G[1][0]*rdis[0] +rdis[1]*G[1][1]*rdis[1] +rdis[1]*G[1][2]*rdis[2];
    dis+=rdis[2]*G[2][0]*rdis[0] +rdis[2]*G[2][1]*rdis[1] +rdis[2]*G[2][2]*rdis[2];





          if(dis < neicutoff)
            {
              nei[i][0]++;
              nei[i][nei[i][0]]=j;

              /*              nei[j][0]++;
                             nei[j][nei[j][0]]=i;*/
            }
        }

    }
}

/*———————————————-*/
#include <math.h>

#include ”initdata.h”
#include ”functions.h”



/*potentail parameters for Va*/
#define cutoff     3.9                 /*Cutoff of potential*/
#define neicutoff 15.0000      /*Square of the nearest neighbour distance */
```

```
//Analitical Finnis-Sinclair  Potential
#define  FS_d 3.692767
# define r_cut 3.8
# define c1  -0.8816318
# define c2  1.4907756
# define c3  -0.3976370
# define A   -2.010637

// And its modification by Ackford and et.al.
# define FND  2.63185120210091
# define B    23.0
# define alp  0.5
# define b0   2.632

// And its modification by Rebonato and et.al
#define Reb_FND    2.6319    //2.45 ///
#define Reb_K      3.3

#pragma _CRI inline density,density_dot,density_dot_dot,phi_pot,phi_pot_dot,phi_pot_dot_dot,F_pot,F_pot_dot,F_pot_dot_dot

 void compute_f()
{

 extern double r[Atoms][3],f[Atoms][3];
 extern int nei[Atoms][300];
 extern double potE,kinE,virsum,l,vol;
 extern double op,n[Atoms];
 extern double pten[3][3], h[3][3];



  int i,j,k;
  int ll,mm;
  double dis,fr,fr1,fr2;
  double phi,dphi,rho,drho;
  double ui,dui,duj;
  double rdis[3],x,y,z;
  mat_mt htr,G;




double nj[Atoms],f_tmp[Atoms][3],ni,fi0,fi1,fi2,
       pt00,pt01,pt02,
       pt10,pt11,pt12,
       pt20,pt21,pt22,pot_E_tmp,virsum_tmp;



  transpose(h,htr);
 mat_mul(htr,h,G); /*build G*/
```

```
  for(i=0;i<3;i++)
  pten[i][0]=pten[i][1]=pten[i][2]=0.;

  for(i=0;i<Atoms;i++)
  n[i]=f[i][0]=f[i][1]=f[i][2]=0.0;

  potE=0.0;
  virsum=0.0;

  /*first loop calc ni*/
#pragma _CRI parallel shared(n,G) defaults
for(i=0;i<Atoms;i++) nj[i]=0.0;

#pragma _CRI taskloop
  for(i=0;i<Atoms;i++)
  {
  ni=0.0;
#pragma _CRI ivdep
  for(k=1;k<=nei[i][0];k++)
        {
          j=nei[i][k]; /*neigh atom*/


   rdis[0]=r[i][0]-r[j][0];
   if(rdis[0]>0.5)rdis[0]-=1.;
   if(rdis[0]<-0.5)rdis[0]+=1.;



   rdis[1]=r[i][1]-r[j][1];
   if(rdis[1]>0.5)rdis[1]-=1.;
   if(rdis[1]<-0.5)rdis[1]+=1.;

   rdis[2]=r[i][2]-r[j][2];
   if(rdis[2]>0.5)rdis[2]-=1.;
   if(rdis[2]<-0.5)rdis[2]+=1.;




    dis=rdis[0]*G[0][0]*rdis[0] +rdis[0]*G[0][1]*rdis[1] +rdis[0]*G[0][2]*rdis[2];
    dis+=rdis[1]*G[1][0]*rdis[0] +rdis[1]*G[1][1]*rdis[1] +rdis[1]*G[1][2]*rdis[2];
    dis+=rdis[2]*G[2][0]*rdis[0] +rdis[2]*G[2][1]*rdis[1] +rdis[2]*G[2][2]*rdis[2];



          dis=sqrt(dis);

          if(dis<cutoff)
            {
              rho=density(dis);
              ni+=rho;
              nj[j]+=rho;
            }  //if(dis<cutoff)
        } //for(k=1;k<=nei[i][0];k++)
 n[i]=ni;
    } //for(i=0;i<Atoms;i++)

#pragma _CRI guard
 for(j=0;j<Atoms;j++)
 n[j]+=nj[j];
#pragma _CRI endguard
#pragma _CRI endparallel
//////////////////////////////////////////////////////////////
#pragma _CRI parallel shared(n,G) defaults
 pot_E_tmp=0.0;
 virsum_tmp=0.0;
 pt00=pt01=pt02=0.0;
 pt10=pt11=pt12=0.0;
 pt20=pt21=pt22=0.0;

for(j=0;j<Atoms;j++)
f_tmp[j][0]=f_tmp[j][1]=f_tmp[j][2]=0.0;

#pragma _CRI taskloop
for(i=0;i<Atoms;i++)
  {
    fi0=fi1=fi2=0.0;
    ui=F_pot(n[i]);
    dui=F_pot_dot(n[i]);
    pot_E_tmp+=ui;

#pragma _CRI ivdep
  for(k=1;k<=nei[i][0];k++)
      {
        j=nei[i][k]; /*neigh atom*/

     rdis[0]=r[i][0]-r[j][0];
     if (rdis[0] > 0.5 ) rdis[0]-=1.;
     if (rdis[0] < -0.5) rdis[0]+=1.;

     rdis[1]=r[i][1]-r[j][1];
     if (rdis[1] > 0.5 ) rdis[1]-=1.;
     if (rdis[1] < -0.5) rdis[1]+=1.;

     rdis[2]=r[i][2]-r[j][2];
     if (rdis[2] > 0.5 ) rdis[2]-=1.;
     if (rdis[2] < -0.5) rdis[2]+=1.;



    dis=rdis[0]*G[0][0]*rdis[0] +rdis[0]*G[0][1]*rdis[1] +rdis[0]*G[0][2]*rdis[2];
    dis+=rdis[1]*G[1][0]*rdis[0] +rdis[1]*G[1][1]*rdis[1] +rdis[1]*G[1][2]*rdis[2];
    dis+=rdis[2]*G[2][0]*rdis[0] +rdis[2]*G[2][1]*rdis[1] +rdis[2]*G[2][2]*rdis[2];
    dis=sqrt(dis);



      if(dis < cutoff)
        {

          phi=phi_pot(dis);

          dphi=phi_pot_dot(dis);
          drho=density_dot(dis);

          duj=F_pot_dot(n[j]);

          pot_E_tmp+=phi;
          fr1=-dphi;
          fr2=-drho*(dui+duj);

          fr=fr1+fr2;

      virsum_tmp+=fr*dis;



      x=rdis[0];
      y=rdis[1];
      z=rdis[2];

      rdis[0] =h[0][0]*x + h[0][1]*y + h[0][2]*z;
      rdis[1] = h[1][0]*x + h[1][1]*y + h[1][2]*z;
      rdis[2] = h[2][0]*x + h[2][1]*y + h[2][2]*z;




  f_tmp[j][0]-=fr*rdis[0]/dis;
  f_tmp[j][1]-=fr*rdis[1]/dis;
  f_tmp[j][2]-=fr*rdis[2]/dis;


       fi0+=fr*rdis[0]/dis;
       fi1+=fr*rdis[1]/dis;
       fi2+=fr*rdis[2]/dis;

    pt00+=fr*rdis[0]*rdis[0]/dis;
       pt01+=fr*rdis[0]*rdis[1]/dis;
       pt02+=fr*rdis[0]*rdis[2]/dis;

    pt10+=fr*rdis[1]*rdis[0]/dis;
       pt11+=fr*rdis[1]*rdis[1]/dis;
       pt12+=fr*rdis[1]*rdis[2]/dis;
```

```
   pt20+=fr*rdis[2]*rdis[0]/dis;
         pt21+=fr*rdis[2]*rdis[1]/dis;
         pt22+=fr*rdis[2]*rdis[2]/dis;
          } //if(dis < cutoff)

  } // cycle: k

   f[i][0]=fi0;
   f[i][1]=fi1;
   f[i][2]=fi2;

  }// cycle: i
#pragma _CRI guard

 for(j=0;j<Atoms;j++)
{
f[j][0]+=f_tmp[j][0];
f[j][1]+=f_tmp[j][1];
f[j][2]+=f_tmp[j][2];
}



 pten[0][0]+=pt00;
 pten[0][1]+=pt01;
 pten[0][2]+=pt02;

 pten[1][0]+=pt10;
 pten[1][1]+=pt11;
 pten[1][2]+=pt12;

 pten[2][0]+=pt20;
 pten[2][1]+=pt21;
 pten[2][2]+=pt22;



 potE+=pot_E_tmp;
 virsum+=virsum_tmp;

#pragma _CRI endguard
#pragma _CRI endparallel

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
double F_pot(double x)
{
return(A*sqrt(x));
}
```

```
/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
double F_pot_dot(double x)
{
if(fabs(x)<0.0000000000001)
x=0.0000000000001;

return(0.5*A/sqrt(x));
}

/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////

double F_pot_dot_dot(double x)
{
if(fabs(x)<0.0000000000001)
x=0.0000000000001;

return(-0.25*A/(x*sqrt(x)));
}

/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
double phi_pot(double x){
double y;

if(x>r_cut) return(0);

y=(x-r_cut)*(x-r_cut)*(c1+c2*x+c3*x*x);
// FS+ Rebonato et al
if(x<Reb_FND)y+=Reb_K*(Reb_FND-x)*(Reb_FND-x)*(Reb_FND-x);



// FS+ Acland et al: Original and true version FND=10000000000
//if(x<FND)//{
 //y+=B*(b0-x)*(b0-x)*(b0-x)*exp(-alp*x);

return(y);

}
/////////////////////////////////////////////////////////////////////////////////////////
double phi_pot_dot(double x){
double y;
if(x>r_cut)return(0);


y=2*(x-r_cut)*(c1+c2*x+c3*x*x)+(x-r_cut)*(x-r_cut)*(c2+2*c3*x);
// FS+ Rebonato et al
if(x<Reb_FND)y+=-3*Reb_K*(Reb_FND-x)*(Reb_FND-x);

// FS+ Acland et al
//if(x<FND)
//y+=B*(b0-x)*(b0-x)*exp(-alp*x)*(-3-alp*(b0-x));


 return(y);

}
/////////////////////////////////////////////////////////////////////////////////////////
double phi_pot_dot_dot(double x){
double y;

if(x>r_cut)return(0);



y=2*(c1+c2*x+c3*x*x)+4*(x-r_cut)*(c2+2*c3*x)+(x-r_cut)*(x-r_cut)*2*c3;
// FS+ Rebonato et al
if(x<Reb_FND)y+=6*Reb_K*(Reb_FND-x);

// FS+ Acland et al
//if(x<FND)
//y+=B*(b0-x)*exp(-alp*x)*(6+6*alp*(b0-x)+alp*alp*(b0-x)*(b0-x));
return(y);

}
/////////////////////////////////////////////////////////////////////////////////////////
 ////////////////////////////////////////////////////////////////////////////////////////

double density(double pr_dis)
{
double rho;
if(pr_dis>FS_d)
return(0);
else
rho=(pr_dis-FS_d)*(pr_dis-FS_d);
return(rho);}

/////////////////////////////////////////////////////////////////////////////////////////
double density_dot(double pr_dis)
{
double rho;
if(pr_dis>FS_d)
return(0);
else
rho=2*(pr_dis-FS_d);
return(rho);
}
```

```
//////////////////////////////////////////////////////////////////////////////////////////
double density_dot_dot(double pr_dis)
{
double rho;
if(pr_dis>FS_d)
return(0);
else
rho=2.0;
return(rho);}

//////////////////////////////////////////////////////////////////////////////////////////


#include <stdlib.h>
#include <stdio.h>
#include <memory.h>


#include "initdata.h"
#include "functions.h"




/******************************************************************************
 * Rahman   Calculate the unit cell matrix accelerations                     *
 ******************************************************************************/
void rahman()

{
 extern double vol;
 extern  mat_mt pten,h,hddot ;
 extern double f[Atoms][3],r[Atoms][3],v[Atoms][3];

    mat_mt     //stress,              /* Stress tensor             */
               h_tr,                  /* Transpose of h            */
               h_tr_inv,              /* Inverse of transpose of h  */
               sigma;                 /* P & R sigma matrix        */
   int         i, j;                  /* Counters                  */



   vol=det(h);
   for(i = 0; i < 3; i++)
     for(j = 0; j < 3; j++)
       pten[i][j]  /= vol;

   for(i = 0; i < 3; i++)
     pten[i][i] -= press;       /* Subtract applied pressure from diagonal    */

   transpose(h, h_tr);          /* Calculate sigma = vol*h transpose inverse  */
```

```
  invert(h_tr, h_tr_inv);
  mat_sca_mul(vol, h_tr_inv, sigma);

  mat_mul(pten, sigma, hddot); /* Calculate unit cell accelerations  */
  mat_sca_mul(1.0/W, hddot, hddot);



  /* Zero unwanted degrees of freedom. Refson PhD Thesis (1986)
    hddot[2][0]=hddot[2][1]=hddot[1][0]=0.0;*/

}
/**************************************************************************/

void parinello()
{  extern  mat_mt pten,h,hdot ;
  extern double f[Atoms][3],r[Atoms][3],v[Atoms][3];
  mat_mt       h_tr,          /* Transpose of h                        */
               h_tr_dot,      /* Transpose of h_dot                    */
               h_tmp_1,       /* Store for intermediate terms          */
               h_tmp_2,       /* Store for intermediate terms          */
               G,             /* h_tr * h     (metric tensor)          */
               G_inv,         /* Inverse of G                          */
               G_dot,         /* Derivative of G                       */
               G_i_d;         /* G_inv * G_dot                         */

  int          i, imol;       /* Counters                              */
  double fc[Atoms][3];

  transpose(h,h_tr);
  mat_mul(h_tr,h,G);                        /* We now have the G matrix   */
  invert(G, G_inv);                         /* G (-1) done               */
  transpose(hdot, h_tr_dot);
  mat_mul(h_tr_dot, h, h_tmp_1);
  mat_mul(h_tr, hdot, h_tmp_2);
  mat_add(h_tmp_1, h_tmp_2, G_dot);         /* G dot now complete         */
  mat_mul(G_inv, G_dot, G_i_d);             /* G_inv * G_dot              */

  mat_vec_mul(G_i_d, v, fc, Atoms);   /* Calculate correction term  */

  scaled_frame(f,Atoms);  /* trans the forces to scaled frame */
  for(i = 0; i < 3; i++)                    /* Add correction term        */
    for(imol = 0; imol < Atoms; imol++)     /* to accelerations           */
      f[imol][i] = f[imol][i] - fc[imol][i];

}
/**************************************************************************
* print_mat                                                               *
**************************************************************************/
void print_mat(a)
mat_mt a;
{
```

```
int i;


for(i=0;i<3;i++)
  {
    fprintf(stderr,"\n");
    fprintf(stderr,"%lf %lf %lf",a[i][0],a[i][1],a[i][2]);
  }
}




/*****************************************************************************
 * mat_mul   Multiply two 3 x 3 matrices. Result can NOT overwrite input.     *
 *****************************************************************************/
void mat_mul(a, b, c)
mat_mt  a,                                    /* Input matrix 1        (in) */
        b,                                  /* Input matrix 2        (in) */
        c;                                  /* Result matrix        (out) */
{
   register int i, j;                          /* Counters                  */




   for(i = 0; i < 3; i++)
      for(j = 0; j < 3; j++)
         c[i][j] = a[i][0]*b[0][j] + a[i][1]*b[1][j] + a[i][2]*b[2][j];
}
/*****************************************************************************
 *  mat_sca_mul.  Multiply a 3x3 matrix by a scalar                         *
 *****************************************************************************/
void mat_sca_mul(s, a, b)
register double s;                   /* Scalar                    (in)  */
mat_mt  a,                           /* Input matrix                (in)  */
        b;                         /* Result matrix             (out)  */
{
   register int i, j;
   for(i = 0; i < 3; i++)
      for(j = 0; j < 3; j++)
         b[i][j] = s * a[i][j];
}
/*****************************************************************************
 * mat_add   Add two 3 x 3 matrices.                                      *
 *****************************************************************************/
void mat_add(a, b, c)
mat_mt  a,                                    /* Input matrix 1        (in) */
        b,                                  /* Input matrix 2        (in) */
        c;                                  /* Result matrix        (out) */
{
   register int i, j;                          /* Counters                  */
  for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
      c[i][j] = a[i][j] + b[i][j];
}
/*****************************************************************************
 * Transpose  Transpose a 3 x 3 matrix.  Will handle case of a = b            *
 *****************************************************************************/
void transpose(a, b)
mat_mt  a,                                    /* Input matrix          (in) */
        b;                                    /* Transposed matrix    (out) */
{
  mat_mt tmp;

  memcpy(tmp, a, sizeof tmp);

  b[0][0] = tmp[0][0]; b[1][1] = tmp[1][1];   b[2][2] = tmp[2][2];
  b[0][1] = tmp[1][0]; b[1][0] = tmp[0][1];
  b[0][2] = tmp[2][0]; b[2][0] = tmp[0][2];
  b[1][2] = tmp[2][1]; b[2][1] = tmp[1][2];
}
/*****************************************************************************
 *  Det.  Determinant of a 3 x 3 matrix                                      *
 *****************************************************************************/
double det(a)
mat_mt  a;                                    /* Matrix              (in) */
{
  int  i, j, k;                               /* Counters                 */
  register double      deter = 0.0;

  for(i = 0, j = 1, k = 2; i < 3; i++, j=(j+1)%3, k=(k+1)%3)
    deter += a[0][i] * (a[1][j]*a[2][k] - a[1][k]*a[2][j]);
  return(deter);
}
/*****************************************************************************
 * invert.  Calculate the inverse of a 3x3 matrix.  Adjoint method.           *
 *****************************************************************************/
void invert(a, b)
    mat_mt     a,                        /* Input matrix         (in) */
 b;                              /* Inverse matrix (out) */
{
 int   i, j, k, l, m, n;                   /* Counters                 */
 register double       deter;             /* Reciprocal of determinant  */


 if((deter = det(a)) == 0.0)
   fprintf(stderr,"\n det = 0 !");
 deter = 1.0 / deter;
 for(i = 0, j = 1, k = 2; i < 3; i++, j=(j+1)%3, k=(k+1)%3)
   for(l = 0, m = 1, n = 2; l < 3; l++, m=(m+1)%3, n=(n+1)%3)
     b[l][i] = deter*(a[j][m]*a[k][n] - a[j][n]*a[k][m]);
}
```

```
/***************************************************************************
 * 3 x 3  Matrix - vector multiply  (of multiple vectors)              *
 * The input and output vectors need not necessarily be distinct         *

***************************************************************/



double my_abs(double x)
{
  if(x > 0.0) return x;
  else return -1.0*x;
}





/***************************************************************/

void mat_vec_mul(m, in_vec, out_vec, number)
int           number;
mat_mt          m;
double          in_vec[][3],
                out_vec[][3];
{
   int i;
   register double      aaa0, aaa1, aaa2;


   if(in_vec == out_vec)
   {

      for(i = 0; i < number; i++)
      {
         aaa0 = in_vec[i][0];  aaa1 = in_vec[i][1];  aaa2 = in_vec[i][2];

         in_vec[i][0] = m[0][0]*aaa0 + m[0][1]*aaa1 + m[0][2]*aaa2;
         in_vec[i][1] = m[1][0]*aaa0 + m[1][1]*aaa1 + m[1][2]*aaa2;
         in_vec[i][2] = m[2][0]*aaa0 + m[2][1]*aaa1 + m[2][2]*aaa2;
      }
   }
   else
   {

      for(i = 0; i < number; i++)
      {
         aaa0 = in_vec[i][0];  aaa1 = in_vec[i][1];  aaa2 = in_vec[i][2];

         out_vec[i][0] = m[0][0]*aaa0 + m[0][1]*aaa1 + m[0][2]*aaa2;
         out_vec[i][1] = m[1][0]*aaa0 + m[1][1]*aaa1 + m[1][2]*aaa2;
         out_vec[i][2] = m[2][0]*aaa0 + m[2][1]*aaa1 + m[2][2]*aaa2;
      }
   }

}

/////////////////////////////////////////////////////////////////////////////
    /***************************************************************************
 *  Det.  Determinant of a 3 x 3 matrix                                    *
****************************************/
double Det(mat_mt a)
{

   register double      deter = 0.0;

   deter=a[0][0]*(a[1][1]*a[2][2]-a[1][2]*a[2][1]);
   deter-=a[0][1]*(a[1][0]*a[2][2]-a[2][0]*a[1][2]);
   deter+=a[0][2]*(a[1][0]*a[2][1]-a[2][0]*a[1][1]);

   return(deter);
}

#include "initdata.h"
#include "functions.h"
#include <math.h>

void evalPT()
{
 extern  mat_mt pten,h ;
extern double f[Atoms][3],r[Atoms][3],v[Atoms][3],temp,s_temp,s_press,virsum,ksi,op,kinE;


  int i,l,k;
  double vvsum;

  vvsum=0.0;

  real_frame(v,Atoms);
  for(i=0;i<Atoms;i++)
    {
      for(k=0;k<3;k++)
        vvsum+=v[i][k]*v[i][k];
     for(l=0;l<3;l++)
       for(k=0;k<3;k++)
         pten[l][k]+=v[i][l]*v[i][k];
    }


  kinE=1.0*vvsum;
  s_temp=vvsum/(3.0*kb*Atoms);
  s_press=(s_temp*Atoms*kb+virsum/3.)/det(h);

   ksi+=3.*Atoms*dt*kb*(s_temp-temp)/Q;
 /*fprintf(stderr,"\n %lf %lf ",ksi,edot); */
 for(i=0;i<Atoms;i++)
   for(k=0;k<3;k++)
     {
       f[i][k]-=ksi*v[i][k];

     }
 scaled_frame(v,Atoms);


}
/*——————————————————*/
real CalcTemp()
{
 extern double v[Atoms][3];
 int i;
 real  vvsum;

 vvsum = 0;

    for (i = 0; i<Atoms; i ++)
    vvsum+=Sqr(v[i][0])+Sqr(v[i][1])+Sqr(v[i][2]);

  vvsum/=(3.0*kb*Atoms);


  return(vvsum);

}






#include "initdata.h"
#include "functions.h"


void corrector()
{
 extern double r[Atoms][3];
 extern double v[Atoms][3];
 extern double f[Atoms][3];
 extern double f1[Atoms][3];
 extern double f2[Atoms][3];
 extern double r1[Atoms][3];
 extern double v1[Atoms][3];

  double cr[]={3.0,10.0,-1.0};
  double cv[]={7.0,6.0,-1.0};
  double div=24.0;

  int i,k;

  for(i=0;i<Atoms;i++)
    for(k=0;k<3;k++)
      {
        r[i][k]=r1[i][k]+dt*v1[i][k]+
          (dt*dt/div)*(cr[0]*f[i][k]+cr[1]*f1[i][k]+cr[2]*f2[i][k]);

        v[i][k]=(r[i][k]-r1[i][k])/dt +
          (dt/div)*(cv[0]*f[i][k] + cv[1]*f1[i][k] + cv[2]*f2[i][k]);

        if(r[i][k] > 1.0) r[i][k]-=1.0;
        else  if(r[i][k] < 0.0) r[i][k]+=1.0;
      }
}
/*——————————————————-*/
void predictorP(hvec,hdvec,hddvec)
     double hvec[9],hdvec[9],hddvec[9];
{
 extern double h1vec[9],hd1vec[9],hdd1vec[9],hdd2vec[9];
  double cr[]={19.0,-10.0,3.0};
  double cv[]={27.0,-22.0,7.0};
  double div=24.0;

  int i;

  for(i=0;i<9;i++)
    {
      h1vec[i]=hvec[i];
      hd1vec[i]=hdvec[i];
      hvec[i]+=dt*hdvec[i]+
        (dt*dt/div)*(cr[0]*hddvec[i]+cr[1]*hdd1vec[i]+cr[2]*hdd2vec[i]);

      hdvec[i]=(hvec[i]-h1vec[i])/dt +
        (dt/div)*(cv[0]*hddvec[i]+cv[1]*hdd1vec[i]+cv[2]*hdd2vec[i]);
      hdd2vec[i]=hdd1vec[i];
      hdd1vec[i]=hddvec[i];
    }
}
/*————————————————————————————————*/
void correctorP(hvec,hdvec,hddvec)
     double hvec[9],hdvec[9],hddvec[9];
```

```
{ extern double h1vec[9],hd1vec[9],hdd1vec[9],hdd2vec[9];
  double cr[]={3.0,10.0,-1.0};
  double cv[]={7.0,6.0,-1.0};
  double div=24.0;

  int i;
  for(i=0;i<9;i++)
    {
      hvec[i]=h1vec[i]+dt*hd1vec[i]+
        (dt*dt/div)*(cr[0]*hddvec[i]+cr[1]*hdd1vec[i]+cr[2]*hdd2vec[i]);
      hdvec[i]=(hvec[i]-h1vec[i])/dt+
        (dt/div)*(cv[0]*hddvec[i]+cv[1]*hdd1vec[i]+cv[2]*hdd2vec[i]);
    }
}
/*——————————————————-*/
void predictor()
{
 extern double r[Atoms][3];
 extern double v[Atoms][3];
 extern double f[Atoms][3];
 extern double f1[Atoms][3];
 extern double f2[Atoms][3];
 extern double r1[Atoms][3];
 extern double v1[Atoms][3];

  double cr[]={19.0,-10.0,3.0};
  double cv[]={27.0,-22.0,7.0};
  double div=24.0;

  int i,k;
  for(i=0;i<Atoms;i++)
    for(k=0;k<3;k++)
      {
        r1[i][k]=r[i][k];
        v1[i][k]=v[i][k];
        r[i][k]+=dt*v[i][k]+(dt*dt/div)*
          (cr[0]*f[i][k]+cr[1]*f1[i][k]+cr[2]*f2[i][k]);

        v[i][k]=(r[i][k]-r1[i][k])/dt +
          (dt/div)*(cv[0]*f[i][k]+cv[1]*f1[i][k]+cv[2]*f2[i][k]);
        f2[i][k]=f1[i][k];
        f1[i][k]=f[i][k];
      }
}


/*****************************************************
                         rdf.c  -  description
                            ———————-
    begin                : Mon Jan 13 2003
    email                : phsorkin@technion.ac.il
```

```
   This part of software is based on copyrighted material, reproduced from the book
   "The Art of Molecular Dynamics Simulation" by D. C. Rapaport,
   published by Cambridge University Press (1995).

   void  EvalRdf (vector_int histRdf, rdf_list *rdf, int mode);
   real Min3R (real x1, real x2, real x3);
   real Max3R (real x1, real x2, real x3);
   void PrintRdf (char *fn, vector histRdf, rdf_list *rdf);
   int RestoreRdfBackUp(char *fn, vector histRdf, rdf_list *pt );
   void BackUpRdf(char *fn, vector histRdf, rdf_list *pt );
*******************************************************************/
 #include "initdata.h"
 #include "functions.h"

 void  EvalRdf (vector histRdf, rdf_list *rdf, int mode)
   {
    extern double r[Atoms][3];
    extern mat_mt h;

    real  rangeRdf;
    real  deltaR, normFac, rr, rrRange;
    int j1, j2, m;
    vector rdis;
    mat_mt  G,htr;

    rdis=AllocVecR(NDIM);

   /*Build G*/
  transpose(h,htr);
  mat_mul(htr,h,G);


      if (mode== 0)
    {
    if( RestoreRdfBackUp("rdf.back",histRdf,rdf) )
    {
      /*Initialization*/
      for (m = 0; m < sizeHistRdf; m ++) histRdf[m] = 0.;
      rdf->countRdf=0;
      rdf->deltaRdf=0.;
    }

    }

    /*Calculate number of measurements*/
    rdf->countRdf ++;;
    rangeRdf=Min3R(h[0][0]/2,h[1][1]/2,h[2][2]/2) ;

    rrRange = Sqr (rangeRdf);
    deltaR = rangeRdf / sizeHistRdf;
    rdf->deltaRdf+=deltaR;

      for (j1 = 0; j1 < Atoms - 1; j1 ++)  {
      for (j2 = j1 + 1; j2 < Atoms; j2 ++) {

     /*Calculate the distance between the pair of atoms (j1,j2) */
   rdis[0]=r[j1][0]-r[j2][0];
   if(rdis[0]>0.5)rdis[0]-=1.;
   else if(rdis[0]<-0.5)rdis[0]+=1.;



   rdis[1]=r[j1][1]-r[j2][1];
   if(rdis[1]>0.5)rdis[1]-=1.;
    else if(rdis[1]<-0.5)rdis[1]+=1.;

   rdis[2]=r[j1][2]-r[j2][2];
   if(rdis[2]>0.5)rdis[2]-=1.;
   else if(rdis[2]<-0.5)rdis[2]+=1.;



         rr=rdis[0]*G[0][0]*rdis[0] +rdis[0]*G[0][1]*rdis[1] +rdis[0]*G[0][2]*rdis[2];
    rr+=rdis[1]*G[1][0]*rdis[0] +rdis[1]*G[1][1]*rdis[1] +rdis[1]*G[1][2]*rdis[2];
    rr+=rdis[2]*G[2][0]*rdis[0] +rdis[2]*G[2][1]*rdis[1] +rdis[2]*G[2][2]*rdis[2];

       /*Check the range and add*/
       if (rr < rrRange) {
         m = (int) (sqrt (rr) / deltaR) ;
         histRdf[m]= histRdf[m] + 1.;
                                                                    }
        } }

   if(mode==2)
{
      /*Calculate the rdf functions */
      normFac = det(h)/ (2. *pi * pow (deltaR, 3.) * Atoms * Atoms*rdf->countRdf);
      for (m = 0; m < sizeHistRdf; m ++)
      histRdf[m] = histRdf[m] * normFac / Sqr (m + 0.5);

      PrintRdf ("rdf.txt", histRdf, rdf);

  } /** output ***/

  FreeVecR(rdis);
  }

/******************************************/
 real Min3R (real x1, real x2, real x3)
{
    if (x1 < x2 && x1 < x3) return (x1);
    else if (x2 < x1 && x2 < x3) return (x2);
    else return (x3);
  }
/******************************************/
  real Max3R (real x1, real x2, real x3)
  {
    if (x1 > x2 && x1 > x3) return (x1);
    else if (x2 > x1 && x2 > x3) return (x2);
    else return (x3);
     }
/*********************************************/
void PrintRdf (char *fn, vector histRdf, rdf_list *rdf)
{

    int m;
    real  rBin;
    FILE *fd;

      if((fd=fopen(fn,"w"))!=NULL)
     {
       if(rdf->countRdf )
{
       for (m = 0; m <sizeHistRdf; m ++)
                                           {
                                            rBin = (m+ 0.5) *(rdf->deltaRdf/rdf->countRdf ) ;
                                            fprintf (fd, "%8.4f %8.4f \n", rBin, histRdf[m]);
                                                  }
}
       fclose(fd);
     }

else
  {
        fprintf(stderr,"I can't open the file '%s'",fn);
  exit(0);
  }
}

int RestoreRdfBackUp(char *fn, vector histRdf, rdf_list *pt )
{
    int m;
    FILE *fd;

      if((fd=fopen(fn,"r"))!=NULL)
     {
      for (m = 0; m <sizeHistRdf; m ++)
                 fscanf (fd, "%lf  \n", & histRdf[m]);

                       fscanf (fd, "%lf  \n", & pt->deltaRdf);
                 fscanf (fd, "%ld  \n", & pt->countRdf);
      fclose(fd);
      return(0);
     }
```

```
else
  {
        fprintf(stderr," \n I can't open the file '%s': Initialization of rdf ",fn);
  return(1);
  }
}
/**********************/
void BackUpRdf(char *fn, vector histRdf, rdf_list *pt )
{
  int m;
  FILE *fd;

      if((fd=fopen(fn,"w"))!=NULL)
     {
      for (m = 0; m <sizeHistRdf; m ++)
                 fprintf (fd, "%lf  \n",  histRdf[m]);

                       fprintf (fd, "%lf  \n",  pt->deltaRdf);
                 fprintf (fd, "%ld  \n", pt->countRdf);
      fclose(fd);

     }

else
  {
        fprintf(stderr,"I can't open the file '%s'",fn);
  }
}




#include <math.h>

#include "initdata.h"
#include "functions.h"


/* vector r contains scaled position of the
atoms, to get the real dis between tow atoms
 we use : dis^2 = transpose(si-sj)*G*(si-sj).*/

void scaled_frame(vec,n)
     double vec[][3];
     int n;
{
   extern mat_mt h;
   mat_mt invh;
  invert(h,invh); /*invert h*/
  mat_vec_mul(invh,vec,vec,n);
}
/*————————————————————————————*/
void real_frame(vec,n)
     double vec[][3];
     int n;
{
 extern mat_mt h;
  mat_vec_mul(h,vec,vec,n);

}
```

ord.c - description

```
 #include "initdata.h"
 #include "functions.h"
 #define sizeHistFFT        600
  void EvalOrd(input_list *data)
{
 extern double r[Atoms][3];
 extern mat_mt h;
 int i,j,k;
 matrix_int layer;
 vector order;
 real delta, height, klat;
 real cossum,sinsum;
 //int num;

 order=AllocVecR(size);
 layer=AllocMatI(size, Atoms) ;


  /*Initialization*/
 for(j=0;j<size;j++)
 for(i=0;i<Atoms;i++)
 layer[j][i]=0;


 real_frame(r,Atoms);

 /*Distribute Atoms between layers*/
 height=pow(det(h),1./3.);
 delta=height/size;

 for(i=0;i<Atoms;i++)
 {
 j=(int)(r[i][2]/delta+0.5);

 if (j>size-1){ j=size-1;}
 if (j<0){ j=0;}
 layer[j][0]++;
 layer[j][layer[j][0]]=i;
 }

  /* Number of atoms
  num=0;
  for(i=0;i<size;i++)
  {
  num+=layer[i][0];
  }
  fprintf(stderr,"Atoms=%d",num);  */


 /*Calculate Order in each layer*/

 klat=2.0*pi*size/height;
 for(j=0;j<size;j++)
 {
 cossum=0.;
 sinsum =0.;
 for (k=1;k<=layer[j][0];k++)
 {
 i=layer[j][k];
 cossum+=cos(klat*r[i][2]);
 sinsum+=sin(klat*r[i][2]);
 } //i
 order[j]=sqrt(cossum*cossum+sinsum*sinsum)/layer[j][0];
 } //j

 SaveOrdDis(order,data);
 scaled_frame(r,Atoms);
 FreeMatI(layer ,size);
 FreeVecR(order);
}
/***********************/
void CalcOrder()
{

extern  mat_mt h ;
extern double r[Atoms][3],op;

  int i;
  double klat;
  double cossum=0.0;
  double sinsum=0.0;

  klat=2.0*pi*size/pow(det(h),(1./3.));
  real_frame(r,Atoms);

  cossum=sinsum=0.0;

  for(i=0;i<Atoms;i++)
  {
    cossum+=cos(klat*r[i][0]);
    sinsum+=sin(klat*r[i][0]);
  }

 op=sqrt(cossum*cossum+sinsum*sinsum)/Atoms;

 scaled_frame(r,Atoms);

}
/*****************/
void SaveOrdDis(vector order, input_list *data)
{
 int m;
 char fn[40];
 FILE *fd;

 m=0;
 while(m<=size )
{
     m++;
      sprintf(fn,"ord_lay_%d.txt",m);
      if((fd=fopen(fn,"a"))!=NULL)
      {
                  fprintf (fd, "%ld %lf  \n", data->Step,order[m]);
      fclose(fd);
      }

                        else
                         {
                                fprintf(stderr,"I can't open the file '%s'",fn);
                         }
}

}
/*********************/
void CalcOrderFFT(input_list *input)
{

extern  mat_mt h ;
extern double r[Atoms][3];

  int i,k;
  double a_0,klat;
  double cossum=0.;
  double sinsum=0.;
  real delta;
  vector order;

  order=AllocVecR(sizeHistFFT);

  a_0=pow(det(h),(1./3.))/size;
  delta=3.*a_0/sizeHistFFT;

  real_frame(r,Atoms);

  for(k=1;k<=sizeHistFFT ;k++)
 {
   cossum=sinsum=0.;

   klat=2*pi/(k*delta);

   for(i=0;i<Atoms;i++)
  {
    cossum+=cos(klat*r[i][2]);
    sinsum+=sin(klat*r[i][2]);

  }

 order[k-1]=sqrt(cossum*cossum+sinsum*sinsum)/Atoms;
   }

 scaled_frame(r,Atoms);
 SaveOrdFFT(order, delta, input);
 FreeVecR(order);
}

/********************/
void SaveOrdFFT(vector order, real delta, input_list *input)
{     FILE *fd;
      int m=0;
      char fn[50];

      sprintf(fn,"fft_%d.txt",input->Step);
      if((fd=fopen(fn,"w"))!=NULL)
      {
     for(m=0;m<sizeHistFFT;m++)
     fprintf (fd, "%lf %lf  \n", (m+1)*delta,order[m]);

     fclose(fd);
      }
                    else
                     {
                            fprintf(stderr,"I can't open the file '%s'",fn);
         }

}



#include <stdio.h>
#include <stdlib.h>
```

```
#include "initdata.h"
#include "functions.h"
void  Aviz(char *fn)
{
extern double r[Atoms][3];
int i;
FILE * fd;


if((fd=fopen(fn,"w"))!=NULL)
{

 fprintf(fd," %d \n",Atoms);
 fprintf(fd,"##Cijkl simulations with Defects\n");
  for (i=0;i<Atoms-Interstitials;i++)
{
  fprintf(fd,"Va %lf  %lf  %lf \n", r[i][0],r[i][1],r[i][2]);

}
 for (i=Atoms-Interstitials;i<Atoms;i++)
 fprintf(fd,"Cu %lf  %lf  %lf \n", r[i][0],r[i][1],r[i][2]);
 fclose(fd);

}

 else
  {
        fprintf(stderr,"I can't open the file '%s'",fn);
  exit(0);
  }

 }
/////////////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "initdata.h"
#include "functions.h"


int MODE,LAST_MODE;
long N_meas_step1,N_meas_step2,step_aviz1,step_aviz2;




double r[Atoms][3];
double v[Atoms][3];
double f[Atoms][3];
double f1[Atoms][3];
double f2[Atoms][3];
double r1[Atoms][3];
double v1[Atoms][3];



int nei[Atoms][300];
double nei_data[Atoms][200][5];
double n[Atoms];


double temp;
double s_temp,s_press;
double potE,kinE,virsum;
double op;
mat_mt pten;
mat_mt h,hdot,hddot;
mat_mt h_zero;
mat_mt e;
mat_mt e_avg;


double  ee_avg[3][3][3][3];

double h1vec[9],hd1vec[9],hdd1vec[9],hdd2vec[9];
double ksi;


double vol,l;
//double tau2,beta;
long  step;
long  step0;
long step_e;

int SL[Atoms][7];

int main()
{
  int lattice_measurements=1;
  long i,j;
  double lat;
  char fn[30];

 /*Let's analyze it*/
  long *t;
  double *x,*y,*z,A[2][2],B[2];
  double  Det,Dett,kx,ky,lx;
  double  err,av;
  int counter;
        FILE *fd;
//Let's analyze the obtained results
```

```
/*Read files and calculate number of elements*/

sprintf(fn,"h_11_22_33.txt");
step =202000;

counter=(int)1.*step/N_neigh_step;
fprintf(stderr,"\n Number of lines in %s = %ld \n",fn,counter);

t=(long *)malloc(counter*sizeof(long));
x=(double *)malloc(counter*sizeof(double));
y=(double *)malloc(counter*sizeof(double));
z=(double *)malloc(counter*sizeof(double));

counter=0;
if(fd=fopen(fn,"r"))
{
while(!feof(fd))
{
fscanf(fd,"%ld %lf %lf %lf \n ",
&t[counter],&x[counter],&y[counter],&z[counter]);
//fprintf(stderr,"%ld %lf %lf %lf \n ",
//t[counter],x[counter],y[counter],z[counter]);
counter++;
}
fprintf(stderr,"\n Hello, I've found N=%d elements",counter);
fclose(fd);
}
else
{
fprintf(stderr,"\n \t Can not open file '%s' ",fn);
exit(0);
}

A[0][0]=A[1][0]=A[0][1]=A[1][1]=0.;
B[0]=B[1]=0.0;



//A[0][0]=2;A[0][1]=1.2;A[1][1]=0.2;A[1][0]=2;B[0]=1;B[1]=0.0;

err=0.0;
j=(int)floor(0.4*counter+0.5);
fprintf(stderr,"\n err=%lf av*av=%d counter=%d j=%ld",err,(int)floor(counter*0.2),counter,j);

for (i=j;i<counter;i++)
{
A[0][0]+=t[i]*t[i];
A[1][0]+=t[i];

B[0]+=t[i]*x[i];
B[1]+=x[i];
err+= x[i]*x[i];
}

counter-=j;
A[0][1]=A[1][0];
A[1][1]=counter;



Det=determ(A[0][0],A[1][0],A[0][1],A[1][1]);
Dett=determ(B[0],B[1],A[0][1],A[1][1]);
lx=Dett/Det;
Dett=determ(A[0][0],A[1][0],B[0],B[1]);
kx=Dett/Det;

av=B[1]/counter;
err=err/counter;
fprintf(stderr,"\n err=%lf av*av=%lf counter=%d j=%ld",err,av*av,counter,j);
err=err-av*av;



fprintf(stderr,"\n \t av=%lf err=%lf lx=%lf kx=%lf ",2*av/size,2.*err/size,2*lx/size,2*kx/size);

exit(0);
/****************/
```

```
 // Let's check if the initial data are determined in selfconsistent     way
  if(size*size*size/4-(Atoms-Interstitials +Vacancies) )
    {
      fprintf(stderr,"\n \t Check the size or Number of Atoms \n \t ");
      fprintf(stderr,"\n \t You should setup Atoms=%d, for the size=%d \n \t ",size*size*size/4+Interstitials -Vacancies,size);
      exit(0);}

 //Defaults values
   l=a0*size/2.0;
  if(Vacancies)
  {lattice_measurements=0;fprintf(stderr,"\n Vacancy Concentration c=%lf \%",Vacancies*1./Atoms);}
  if(Interstitials)
  {lattice_measurements=0;fprintf(stderr,"\n Self Interstitials Concentration c=%lf \%",Interstitials*1./Atoms);}
  if(lattice_measurements)init_lattice();

  fprintf(stderr,"\n \t Atoms %d",Atoms);
  fprintf(stderr,"\n \t lattice box length=%g",l);
  get_inform();
```

```
if(MODE==0)
{
fprintf(stderr,"\n \t Starting from the beginning ");
init(1);
step0=0;
}
else
{
if(MODE==1)
{
fprintf(stderr,"\n \t Continue approaching  to the steady state ");
LAST_MODE=1;
}
else
{
fprintf(stderr,"\n \t Cijkl measurements ");
LAST_MODE=2;
}

init(2);
step=step0;
}



 build_nei_list();
 compute_f();
 fprintf(stderr,"\n potE=%g virial=%g volume=%g ",potE,virsum,det(h));
 rahman();




  if(MODE!=2)
  {
  for(i=0;i<N_meas_step1;i++)
  {
  for(j=0;j<N_neigh_step;j++)
        {
          step++;
          predictor();
          compute_f();
          evalPT();
          rahman(h);
          predictorP(h,hdot,hddot);
          parinello();
          correctorP(h,hdot,hddot);
          corrector();

        }

   build_nei_list();
  if(step % step_EPTVO_meas==0)
        {
  save_EPTVOh();
if(lattice_measurements)
{
lat=lattice();
fprintf(stderr,"\n a0=%lf ",lat);
}

  }

  if(step % step_backup == 0.0)
{
 save_h();
 save_h0();
 save_to_file(0);
 save_inform();
}

if(step % step_aviz1==0)
{
sprintf(fn,"pos_%ld.xyz",step);
real_frame(r,Atoms);
Aviz(fn);
scaled_frame(r,Atoms);
}
 /*if(step % step_rv_meas==0)save_to_file(step+step0,1); // For the future Fourier Transformation*/

   fprintf(stderr,"\n step=%d order=%lf temp=%lf",step,op,s_temp);
   fprintf(stderr,"\n energy=%lf volume=%lf pressure=%lf",potE+kinE,det(h),s_press);



   }

/////////////////////////
real_frame(r,Atoms);
Aviz("Va1.xyz");system("aviz -ar Va1.xyz &");
scaled_frame(r,Atoms);
//////////////////////////

if(N_meas_step2){MODE=2;LAST_MODE=1;}
save_h();
save_h0();
save_to_file(0);
save_inform();

  } // if MODE !=2

///////////////////////////////////////////////////////////////////////////

  for(i=0;i<N_meas_step2;i++)
  {
  for(j=0;j<N_neigh_step;j++)
        {
          step++;
          predictor();
          compute_f();
          evalPT();
          rahman(h);
          predictorP(h,hdot,hddot);
          parinello();
          correctorP(h,hdot,hddot);
          comp_c();
          corrector();
                }

  build_nei_list();

 if(step % step_EPTVO_meas==0)
        {
  save_EPTVOh();
 if(lattice_measurements)
{
lat=lattice();
fprintf(stderr,"\n a0=%lf ",lat);
}
  }

  if(step % step_backup == 0.0)
{
 save_h();
 save_h0();
 save_to_file(0);
 save_inform();
 e_ee_backup();
}

if(step % step_aviz2==0)
{
real_frame(r,Atoms);
sprintf(fn,"pos_%ld.xyz",step);
Aviz(fn);
scaled_frame(r,Atoms);
}
   fprintf(stderr,"\n step=%d order=%lf temp=%lf",step,op,s_temp);
   fprintf(stderr,"\n energy=%lf volume=%lf pressure=%lf",potE+kinE,det(h),s_press);

    }
/////////////////////////////////////////////////////////////////////


// Now, save it
```

```
   save_h0();
   save_h();
   save_to_file(0);
   save_inform();
   e_ee_backup();
// And visualize it
   real_frame(r,Atoms);
   real_frame(v,Atoms);
// Now, save velocities and coordinates for NVT
    write_rv();
   // And visualize it
 if(N_meas_step2){Aviz("Va2.xyz");system("aviz -ar Va2.xyz &");}
 fprintf(stderr,"\n\t********* The Happy End ***************\n");




 return(0);

}

/**********************/
double determ(double x00,double x01,double x10,double x11)
{return(x00*x11-x01*x10); }
```

References

[1] F. Lindemann, Z.Phys, **11**, 609, (1910)

[2] Roberts and Miller, *Heat and Thermodynamics*, Blackie and Son, London, 1960

[3] J.J.Gilvary, Phys. Rev., **102**, 308, (1956)

[4] P. Debye, Ann. Physik, **49**, 49, (1914)

[5] J. N. Shapiro, Phys. Rev, **18**, 3982, (1970)

[6] Z. H. Jin, P. Gumbsch, K. Lu, and E. Ma, Phys. Rev. Lett., **87**, 055703, (2001)

[7] L. H. Cohen, W. Klement and G.C. Kennedy, Phys. Rev., **145**, 519, (1966)

[8] Y. Ida, Phys. Rev., **187**, 951, (1969)

[9] M. Born, Journ. of Chem. Phys., **7**, 591, (1939)

[10] L. Hunter, S. Siegel.Phys. Rev., **61**, 84 (1942)

[11] Y. P. Varishni, Phys. Rev. B, **2**, 3953, (1970)

[12] R. M. Niklow and A. A. Young, Phys. Rev., **129**, 1936, (1963)

[13] A. W. Lawson, Phys. Rev., **48**, 85, (1950)

[14] J. Wang, S.Yip, S. R. Phillpot and D. Wolf, Phys. Rev. Lett., **71**, 4182, (1993)

[15] J.Wang, J. Li, and S. Yip, Phys. Rev. B, **52**, 12627, (1995)

[16] J.L.Tallon, Phyl. Mag.A, **39**, 151, (1978)

[17] A. Kanigel, J. Adler and E. Polturak, International Journal of Modern Physics C, **12** , 727, (2001)

[18] J. Wang, J. Li, S. Yip and D. Wolf, Physica A, **240**, 356, (1997).

[19] A. R. Ubbeldone, *Melting and Crystal Structure*, Clarendon press, Oxfrord, 1965

[20] T.A. Weber and F.H. Stillinger, Journ. of Chem. Phys., **81**, 5095, (1984)

[21] A. Granato, Phys. Rev. Lett., **68**, 974, (1992)

[22] Tammann, Z. Phys. Chem. Stoechiom. Verwandtschalft, **68**, 205, (1910)

[23] J. Frenkel and J.F. Van der Veen, Phys. Rev. B, **34**, 7506, (1986)

[24] H. Hakkinen and U. Landmann, Phys. Rev. Lett., **71**, 1023, (1993)

[25] D. M. Zhu and J.G. Dash, Phys. Rev. Lett., **57**, 2959, (1986)

[26] J. Frenkel and J. F. Van der Veen, Phys. Rev. Lett., **54**, 134, (1986)

[27] J. F. van der Veen, *Phase Transitions in Surface Films 2*, ed. H. Traub, Plenum Press, New York, 1991

[28] W. Lipowsky, Phys. Rev. Lett, **49**, 1575, (1982)

[29] W. Lipowsky and W. Speth, Phys. Rev. B, **28**, 3983, (1983)

[30] W. Lipowsky, U. Breuer, K. C. Prince, and H. P. Bonzel, Phys. Rev. Lett., **62**, 913, (1989)

[31] O. Tomagnini, F. Ercolessi, S. Iarlori, F.D. Di Tolla and E. Tosatti, Phys. Rev. Lett., **76**, 1118, (1996)

[32] F.D. Tolla, F. Ercolessi, S. Iarlori, Surf. Sci., **211/212**, 55, (1989)

[33] S. Iarlori, P. Carnevali, F. Ercolessi, and E. Tosatti, Surf. Sci.**211/212**, 75 (1989).

[34] O. M. Magnussen, Phys. Rev. Lett., **74**, 4444, (1998)

[35] I. Rice, J. Brawn, S. Pott, Chem. Phys., **87**, 3069, (1987)

[36] A. Traynov and E.Tossati, Phys. Rev. B, **38**, 6961, (1989)

[37] A. S. Levi and E.Tossati, Surf. Sci., **189/196**, 641, (1990)

[38] J. Krim, J. P. Coulomb and J. Bouzidi, Phys. Rev. Lett., **58**, 583, (1987)

[39] C. S. Jayanti et al., Phys. Rev. B, **31**, 3456, (1985)

[40] R. Ohnesorge, H. Lowen, and H. Wagner, Phys. Rev. E, **50**, 4801, (1994)

[41] J. P. Hansen and I. R. McDonald, *Theory of Simple Liquids*, 2nd ed., Academic, London, 1986

[42] M. Finnis and J. Sinclair, Philos., Mag. A, **56**, 11, (1987)

[43] S. Foiles, M. Baskes, and M. Daw, Phys. Rev. B, **33**, 7983, (1986)

[44] M. Finnis and J. Sinclair, Philos., Mag. A, **50**, 49, (1984)

[45] K. Jacobsen, J. Norskov, and M. Puska, Phys. Rev. B, **35**, 7423, (1987)

[46] F. Ercolessi and J. Adams, Europhys. Lett., **26(8)**, 583, (1994)

[47] F. Ducastelle, *Computer simultions in material science*, Kluwer, Doredrect, 1991

[48] A. Sutton, *Electronic Structure of Materials*, Clarendon Press, Oxford, 1993

[49] S. Foiles, M. Baskes, and M. Daw, Phys. Rev. B, **33**, 7983, (1986)

[50] P. Bujard, *PhD Thesis* , University of Geneva, 1982

[51] C. Kittel, *Introduction to Solid State Physics*, 5th edition, NY, Willey, 1983

[52] R. Rebonato, D. O. Welch, R. D. Hatcher and J. C. Billelo, Philos., Mag. A, **55**, 655, (1987)

[53] M. Marchese, G. Jacucci and C. P. Flynn, Philos. Mag. Lett., **57**, 25, (1988)

[54] D. Rapaport, *The art of MD simulations*, Cambridge, University Press, 1991

[55] H. Haile, *MD simulation elementary methods*, Willey, 1989

[56] Gear, *Numerical initial value problems in ordinary differential equations*, Prentice Hall, EngleWood, 1973

[57] J. Ray and A. Rahman, J. Chem. Phys., **80** , 4423, (1984)

[58] J. Ray and A. Rahman, J. Chem. Phys., **82** , 4243, (1985)

[59] G. Sutmann, *Classical Molecular Dynamics*, **http://www.fz-juelich.de/nic-series/volume10**

[60] B. Ziff, Computers in Physics, **12**, 385, (1998)

[61] S. Nose, J. Chem. Phys., **81**, 511, (1984)

[62] G. Sundman, Celestial Mech., **11**, 469, (1985)

[63] W. G. Hoover, Phys. Rev. A, **31**, 1695, (1985)

[64] D. Frenkel, B. Smit, *Understanding Molecular Simulations: From Algorithms to Applications* Academic Press, 1996

[65] W. G. Hoover, *Time Reversibility, Computer Simulation and Chaos*, World Scientific, 1999

[66] M. Parinello and A. Rahman, Phys. Rev. Lett., **45**, 1196, (1980)

[67] A. Kanigel, *The Role of Defects in the Melting Transition*, Research Thesis, Technion, 1999

[68] M. Born and H. Huang, *Dynamical Theory of Crystal Lattices*, Clarendon Press, 1988

[69] I. Yukito, Intern. Journ. Of Mod. Physics C, **12, No.5**, 623, (2001)

[70] M. Lubartsev and V. Vorontzov, J. Chem. Phys., **96**, 1776, (1992)

[71] E. Marinary, R. Parisi, Europhys. Lett., **69**, 2292, (1992)

[72] J. Adler, A. Hashibon, N. Schreiber, A. Sorkin, S. Sorkin and G. Wagner, Comp. Phys. Comm., **12, No.5**, 623, (2002) **http://phycomp.technion.ac.il/∼aviz/index.html**

[73] G.J. Ackland and R. Thefford, Phil. Mag. A, **50**, 313, 1987

[74] *Handbook of Chemistry and Physics*, 81'st edition 2000-2001 year, D. R. Lide, editor in chief, CRC Press

[75] P. H. Dederichs, C. Lehmann, and T. Scholz, Phys. Rev. Lett., **31**, 1130, (1973)

[76] J. Frenkel, *Kinetic Theory of Liquids*, Clarendon, Oxford, (1946)

[77] J. F. Lutchko, D. Wolf, S. R. Phillipot and S. Yip, Phys. Rev. B, **40**, 2841, (1989)

[78] P. Stolz, J. K. Norskov, and U. Landman, Phys. Rev. Lett., **61**, 440, (1988)

[79] E. T. Chen, R. N. Barnett, and U. Landman, Phys. Rev. B, **41**, 439, (1990)

[80] H. Hakkinen, and M. Manninen, Phys. Rev. B, **46**, 1752, (1992)

[81] A. A. Maradudin and P. A. Flinn, Phys. Rev., **129**, 2529, (1963)

[82] M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids*, Clarendon Press, Oxford, 1987

[83] H. Hakinen and M.Mannine, Phys. Rev. B, **46**, 1690, (1991)

[84] P. Carnevali, E. Ercollesi and E. Tossati, Phys. Rev. B, **36**, 6701, (1987)

[85] K. D. Stock and E.Menzil, Surf. Sci., **61**, 272, (1976)

[86] U. Landmann and R. N. Barnett, Phys. Rev. Lett., **45**, 2032, (1980)

[87] U. Landmann and R. N. Barnett, Phys. Rev. B, **37**, 4637, (1988)

[88] B. Bilgramm, Phys. Rep., **153**, 1, (1983)

[89] E. T. Chen, R. N. Barnett, and U. Landman, Phys. Rev. B., **40**, 924, (1989)

[90] D. Wolf, P. R. Okamoto, S. Yip, J. F. Lutsko, and M. Kluge, J. Mater. Res, **5**, 286, (1990)

[91] J. Q. Broughton, G. H. Gilmer, and K. A. Jackson, Phys. Rev. Lett., **49**, 1496, (1982)

[92] E. Polturak, private communication

[93] J. Daeges, H. Geiter and J. H. Peperezko, Phys. Lett. A, **119**, 79, (1986)

[94] W. K. Bucton, N. Carbera and F. C. Frank, Philos. Trans. R. Soc. London, **243**, 299, (1951)

[95] G. Grimvall and S. Sjodin, Physica Scripta, **10**, 340, (1974)

[96] S. Nose, Molec. Phys. , **52**, 255, (1984)

[97] H. Löwen, Phys. Rep, **237**, 249, (1994)

[98] A. Hashibon, "Atomistic study of structural correlations at a model solid/liquid metal interface", Research Thesis, Technion,(2002)

[99] A. Hashibon, J. Adler, M. Finnis and W. D. Kaplan, Computational Materials Science, (2002) **24**, 443,(2001)

[100] B. Lee, M. I. Baskes, H. Kim and Y. K. Cho, Phys. Rev. B, **64**, 184102, (2001)

פגמים נקודתיים, מבנה הסריג והתכה

סלבה סורקין

פגמים נקודתיים, מבנה הסריג והתכה

חיבור על מחקר

לשם * מילוי חלקי של הדרישות לקבלת תואר

מגיסטר למדעים

פיסיקה

סלבה סורקין

הוגש לסנט הטכניון — מכון טכנולוגי לישראל

אלול תשס"ה חיפה ספטמבר 2005

חיבור על מחקר נעשה בהדרכת דר' יוחנה אדלר ופרופ' אמיל פולטורק
בפקולטה פיסיקה

הכרת תודה

המחקר נעשה בהנחיית דר' יוחנה אדלר ופרופ' אמיל פולטורק בפקולטה לפיסיקה בטכניון. ברצוני להודות לדר' יוחנה אדלר ופרופ' אמיל פולטורק על הדרכתם המסודר, הידע הרב שתרמו לי, על סבלנותם הרבה במשך זמן ההשתלמות ובעיקר על העידוד הרב שנתנו לי כאשר התוצאות סירבו להגיע.

אני מודה למחב"א על שאפשרו לי לעבוד על מחשיבים על ו $Linux\ Cluster$.
ברצוני להודות במיוחד ליונתן טל ולמושא גולדברג על עזרתם ועל שיתוף הפעולה במקביליזצית החישובים. אני מודה לאישתי אנסטסיה, לחברי אדהם חשיבון, גארי וגנר, עמית קניגל, ניר שרויבר על העזרה בתחזוקת המחשבים ועל העצות המועילות.

אני מודה להטכניון ולהקרן גרמניה ישראל על התמיכה הכספית הנדיבה בהשתלמותי

תוכן ענינים

רשימת איורים

רשימת טבלאות

תקציר

אל עף פי שההתכה היא מתופעות הטבע היומנית ביותר והיא מוכרת לאדם זמן רב, לא קיים תיאור מיקרוסקופי מוסכם של תהליך ההתכה. במשך השנים הוצוע תאוריות רבות, התאוריה הראשונה הידועה ביותר היא תאוריה של *Lindemann*. ידוע שככל שטמפרטורת הגביש עולה כך משרעת התנועה של האטומים סביב נקודת שיווי-המשקל לשהם גדלה. על פי המודל ההתכה תקרה כאשר משרעת הממוצעת תהיה גדולה עד שאטום אחד יפריע לשכנו והגביש לא יהיה יותר יציב. כעשר שנים מאוחר יותר פיתח *Born* מודל להסבר תהליך ההתכה. *Born* חישב את התנאים הכלליים בהם מבנה הגביש יהיה יציב. על פי המודל ההתכה תתרחש כאשר אחד ממקדמי הגזירה (*shear elastic moduli*) C_{44} או C' יתאפס. בנוסף, קיימות תאוריות רבות הקושרות את המנגנון המיקרוסקופי של ההתכה עם קיומם של פגמים נקודתיים בגביש. לרוב ריכוז הפגמים הוא נמוך מאד וקשה להניח שדי בנוכחותם כדי להתיך את הגביש. הוצעו מספר מודלים המבוססים על נוכחותם של פגמים שונים, כגון חדירויות (*interstitials*), העדרויות (*vacancies*). המכנה המשותף לכל המודלים הללו הוא זיהוי של תהליך ההתכה עם גידול פאתומי בריכוז הפגמים בטמפרטורת המעבר.

אחת מהמטרות של העבודה הזו היא לשפוך מעט אור על תפקידם של פגמיים נקודתיים בתהליך ההתכה של גבישיים בעלי מבנה סריג קובי מרוכז גוף. לשם כך בחרנו למידול גביש של ונדים. עשינו שימוש בכלים נומריים כגון דינמיקה מולקולרית (*MD*) ומונטה-קרלו. הבסיס של השיטות הנומריות האלו הוא הפוטנציאל הבין-אטומי שנבחר כדי לתאר

את האינטראקציה בין האטומים. בחרנו לעבוד עם פוטנציאל של ונדיום שהוא פוטנצאיל רב-גופי שפותח על ידי *Finnis* ו *Sinclair*. לשם מידול דגמי ונדיום בטמפרטורות שונות השתמשנו בתוכניות MD שמדמות הצבר הקנוני (אלגוריתם של $Nose - Hoover$) ולשם סימולציות בלחץ קבוע השתמשנו באלגוריתם של $Parinello - Rahman$.

תחילה בדקנו את הפוטנצאיל על ידי חישוב של מספר גדלים (מקדם התפשטות תרמי, מקדמים אלסטיים) שניתנים להשוואה עם נתונים ניסיוניים. ההתאמה לגדלים ניסיוני-ים היא סבירה. לאחר מכן בדקנו את התצורה והאנרגיה של פגמים נקודתיים בתוך הגביש של ונדיום. לשם כך השתמשנו באלגורתם הנקרא *simulated tempering* . עבור *interstitials* קיים מספר רב של תצורות אפשריות בעלות אנרגיות שונות. מצאנו התצורה בעלת האנרגיה הנמוכה ביותר היא $split-interstitialcy$ $< 110 >$, זהו פגם שנוצר כאשר מכניסים אטום נוסף לתא יחידה של הגביש. מבחינת האנרגיה הפוטנצאלית לגביש כדאי להזיז את האטום הנמצא במרכז תא היחידה ממקומו כך ששני האטומים הללו (אחד שי-יך לתא היחידה ו-אחר הוא האטום הנוסף) יסתדרו סביב מיקום רגיל של אטום בתא היחידה.

השפעת פגמיים נקודתיים על הקבועים האלסטיים נבדקה על ידי שימוש בדינמיקה מולקולרית. חישבנו את הקבועים האלסטיים של דגמים בעלי ריכוזים שונים של הדירויות והעדרויות בטמפרטורות שונות. נמצא שההדירויות מורידות את הקבועים האלסטיים בטמפרטורות גבוהות, לעמת זאת העדירויות לא משפיעות כלל על המקדמים האלסטיים, לפחות בריכוזים נמוכים. בטמפרטורות קרובות לנקודת ההתכה השפעת הדירויות על המקדם האלסטי C' יותר גדולה מהשפעה על המקדם האלסטי C_{44}. כלומר בטמפרטורות גבוהות מספיק ריכוז קטן של הדירויות כדי להוריד את ערכו של C' לאפס ועל ידי כך לגרום להתכה של גביש על פי מודל של *Born*.

מדדנו את השפעת הפגמים על טמפרטורת ההתכה של ונדיום. מצאנו כי ריכוז של 0.5% מוריד את טמפרטורת ההתכה ב 40 מעלות. תהליך ההתכה זה זה נקרא התכה מכנית, בשונה מתהליך התכה תרמו-דינמית. מצאנו שהתכה מכנית מתרחשת בתנאי

שהנפח הממוצע שווה לנפח קריטי, ואין תלות כלשהיא בדרך שבאה מגיעים לנפח הקרי-טי הנ"ל, כלומר ניתן לחמם גביש אידאלי או לנפח אותו על ידי הוספת פגמים נקודתיים, בסיכומו של דבר התוצאה היא התכה מכנית באותה טמפרטורה קריטית.

עובדה ניסיונית ידועה היטב היא שלא ניתן לחמם גביש מעבר לטמפרטורת ההתכה שלו. אמנם $Daeglas$ ושותפיו הצליחו לחמם גביש כסף מעל טמפרטורת ההתכה על ידי ציפוי בשכבה של זהב שהוא חומר בעל מבנה סריג זהה לכסף, אך טמפרטורת ההתכה שלו יותר גבוה מטמפרטורת ההתכה של כסף. ניסוי זה מראה שלפני השטח תפקיד מרכזי בתהליך ההתכה, פרושו של דבר- ההתכה לא מתרחשת בצורה הומוגנית בתוך נפח הגביש, אלא מתחילה על פני השטח ומתקדמת פנימה.

בסימולציות שלנו הצלחנו לחמם את הדגם של ונדיום מעבר לטמפרטורת ההתכה התרמו-דינמית שלו על ידי שימוש בתנאי שפה מחזוריים בשלושת הכיוונים. נמצא שנ-קודת ההתכה עבור התכה מכנית של ונדיום היא $T_b = 2500 \pm 5K$, בזמן שטמפרטורת ההתכה התרמו-דינמית של ונדיום על פי ניסוי היא $T_m = 2183K$. ניתן לחשב את טמפר-טורת ההתכה התרמו-דינמית בסימולציות אם מפעלים תנאי שפה מחזוריים בשני כיוונים בלבד, ואילו בכיוון השלישי מפעילים תנאי שפה הופשיים. כאשר מבצעים את הסימולציות בטמפרטורה שהיא מעבר לטמפרטורת ההתכה התרמו-דינמית מופיעה שכבת נוזל על פני המשטח של הגביש. שכבת הנוזל מתקדמת פנימה לתוך הגביש במהירות שתלויה בהפרש בין טמפרטורת הדגם לבין טמפרטורת ההתכה התרמו-דינמית. על ידי מדידת מהירות ההתקדמות במספר טמפרטורות ניתן להעריך את הטמפרטורה שבה מהירות ההתקדמות של פאזת הנוזל תתאפס. זוהי טמפרטורת ההתכה התרמו-דינמית. הסימולציות נערכו עבור כיוונים שונים של פני השטח של ונדיום, כגון (011), (001) ו-(111). נמצא שטמפרטורת ההתכה התרמו-דינמית עבור הדגם בעל מבנה המשטח (111) (כלומר המשטח בעל צפיפות אטומיסטית הנמוכה ביותר) היא $T_m = 2220 \pm 5K$.

בנוסף לסימולציות של תהליך התכה מכנית חקרנו בעזרת דינמיקה מולקולרית את התופעה הנקראה התכת קדם של פני-השטח ($surface\ premelting$) של ונדיום. העבודה

הניסיונית על אי-מציאות של חימום על (*superheating*) בגבישים מרמזת על תופעה התכת קדם של פני השטח, כלומר מתחת לנקודת ההתכה התרמו-דינמית מופיעה שכבה דקה של נוזל על פני-שטח הגביש. הודות לשכבה זו לא קיימים מחסומים אנרגטיים להיווצרות של הפאזה החדשה, ולכן אי אפשר לצפות במעבדה בתופעת חימום על. לעמת זאת ניתן לקרר נוזל מעבר לנקודת הקיפאון עקב קיום מחסומים אנרגטיים בתהליך קפיאה. מבחינת תאוריה ניתן להתייחס להתכת קדם משטחית כי אל הרטבת פני השטח של הגביש על ידי הפאזה הנוזלית, כאשר במקום משטח ביניים אחד - מוצק-גז מופעים שני משטחים ביניים : מוצק-נוזל ונוזל-גז. רוב המודלים התאורטיים של התכה משטחית הם מודלים פנומנולוגים, ורק לאחרונה הוצעו מודלים מיקרוסקופים שמבוססים על קירוב של השדה הממוצע. המודלים הללו מסוגלים לתאר את התכת פני השטח של גבישי גזים אינרטיים. במידה רבה רק סימולציות מסוגלות לשפוך אור על הפרטים המיקרוסקופים של התכת פני השטח של גבישים. ממחקרים הללו עולה כי המעבר הזה קורה בהדרגתיות תוך כדי היווצרות שכבה נוזלית על פני הגביש. כאשר האינטראקציה בין האטומים היא קצרת-טווח כמו במתכות, עובי השכבה הזו הולכת וגדלת באופן לואגרתמי עם טמפרטורה.

בעבודה זאת תהליך התכת פני השטח של ונדיום נחקר ע"י שימוש בדינמיקה מול-קולרית. לשם כך בנינו שלושה דגמים של הגביש ולכל הדגם מבנה פני השטח שונה, כגון (001), (011) ו-(111). השתמשנו בתנאי שפה מחזוריים בשני הכיוונים ואילו בכיוון השלישי - בתנאי שפה הופשיים. במהלך הסימולציה נמדד פרופיל הצפיפות בתוך הדגם, פאראמ-טר הסדר בכל שכבה, פונקצית התפלגות הרדיאלית הדו-ממדית, כמו כן מקדמי הדיפוזיה ואיכלוס השכבות של הגביש ומרחק ביניהן כתלות בטמפרטורה. נמצא שבטמפרטורה מסוימת מתחיל תהליך היווצרות של פגמים נקודתיים בשכבה הראשונה של פני השטח. כתוצאה מזאת מופיעה שכבה נוספת על פני השטח של הגביש. הפגמים הנקודתיים מורי-דים את ערכם של מקדמי הגזירה ובסופו של דבר מופיע המשטח המפריד בין מוצק לנוזל המורכב ממספר שכבות אטומיות בעלות ריכוז פגמים גדול. המשטח הזה מתקדם פנימה ע"י דיפוזיה והיווצרות פגמים חדשים ככל שטמפרטורת הדגם עולה. יש לציין שגבול בין

הנוזל לבין המוצק אינו גבול חד ותלול, אלא מעבר ממוצק לנוזל קורה בהדרגתיות. מצאנו שניתן להבחין בהתכת פני השטח בולטת של ונדיום בכיוון (111) , שבו צפיפות האטומים נמוכה ביותר, אפילו בטמפרטורה יחסית נמוכה $(T \cong 1800)$. לעומת זאת בכיוון (011), שעבורו צפיפות האטומים גבוהה ביותר, ניתן להתבונן בהתכה משטחית אף ורק בסביבת נקודת ההתכה התרמו-דינמית.

השתמשנו במודל של $Born$ על מנת להסביר את התופעה של התכת של פני השטח הגביש. הריכוז הגבוה של פגמים נקודתיים על פני השטח גורם להתפשטות של השכבה המשטחית של הגביש, ולכן הנפח הממוצע לאטום בשכבה זו מגיע לערך הקריטי, כאשר מעבר לערך הזה סריג אינו יציב ובאופן פתאמי חל התהליך של התכה מכנית. על פי $Polturak$ ישום של המודל הזה להתכת פני השטח הגביש מוליך ליחס לינארי בין טמ-פרטורת ההתכה התרמודינמית ולבין אנרגית היווצרות של פגמים משטחיים. בדקנו את הנבואה זאת ע"י סימולציות, כלומר מדדנו את האיכלוס של השכבה הנוספת $(adlayer)$ כתלות בטמפרטורה ועל סמך זאת חישבנו את אנרגית היווצרות של פגמים משטחיים. כמו כן השתמשנו בנתונים ניסיוניים להערכת האנרגיה הזו על פי המודל התאורתי. עבור פני השטח בכיוון (111) נמצא שאנרגית היווצרות של פגמים משטחיים $E_s = 0.3 \pm 0.05eV$ ומספר זה קרוב לנבואה התאורתית $E_s = 0.44eV$. על סמך התוצאות הסימולציות הגענו למסקנה שמודל של $Born$ אכן מתאר גם התכה מכנית וגם התכה תרמו-דינמית. מסקנתנו היא שרבום הסיכויים שמודל זה הוא למעשה "החוליה ההוסרת" שבסופו של דבר תקשר שני תפריטי ההתכה.