UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Molecular Dynamics Study of Void Initiation and Growth in Monocrystalline and Nanocrystalline Copper

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Engineering Sciences (Mechanical Engineering)

by

Sirirat Traiviratana

Committee in charge:

Professor David J. Benson, Chair
Professor Marc A. Meyers, Co-Chair
Professor Eduardo M. Bringa
Professor Petr Krysl
Professor Vlado A. Lubarda
Professor Vitali F. Nesterenko

2008

The dissertation of Sirirat Traiviratana is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2008

DEDICATION

To my passed father, Mr. Prasert Traiviratana
To my passed brother, Master Sergeant 1st Class Sirisak Traiviratana
To my patient mother, Mrs. Pranorm Traiviratana
To the Royal Thai Navy

Lieutenant Sirirat Traiviratana

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Financial support from LLNL grant B558558, ILSA contract number W-7405-Eng-48 and NSF EVO CBET-0742730 Program is gratefully acknowledged. I would like to thank Hussam Jarmakani and Paul Erhart for help with nanocrystalline samples of nickel, Eduardo Bringa for his help and guidance on molecular dynamics, Prof. Meyers for encouraging me to take a class of "Imperfections in solids" and last but not least, Prof. Benson for providing, time and time again, incredible practical solutions to many problems throughout my research.

Sections in the chapter 3 concerning void growth in single crystalline copper including the calculations of dislocation interactions were published as an article in Acta Materialia, vol 56 (2008), page 3874-3886 with the co-authors Eduardo M. Bringa, David J. Benson and Marc A. Meyers.

Sections in the chapter 3 concerning void growth in single crystalline copper with uniaxial loading direction along [110] and [111] and their analysis are in preparation for publication.

VITA

April 10, 1976	Born, Samut Songkhram, Thailand,
1995	Armed Force Academy Preparatory School, Bangkok, Thailand,
2000	B. S., Royal Thai Naval Academy, Samut Prakarn, Thailand,
2002-2003	Teaching assistant, Department of Mechanical Engineering, Oregon State University,
2003	M. S., Oregon State University, Corvallis, OR, USA,
2004-2008	Teaching and Research assistant, Department of Mechanical and Aerospace Engineering, University of California, San Diego,
2008	Ph. D., University of California, San Diego, La Jolla, CA, USA

PUBLICATIONS

D. J. Benson, S. Traiviratana, M. A. Meyers, P. Dixit, A. Koniges, D. Kalantar. "Void Growth and Coalescence Nanocrystalline Metals: Molecular Dynamics Modeling, Continuum Modeling, and Experiments." In M. Elert, M. D. Furnish, R. Chau, N. C. Holmes, and J. Nguyen, editors, *Shock Compression of Condensed Matter*, number 1, pages 343-346. 15th APS Tropical Conference, American Institute of Physics, 2007.

S. Traiviratana, E. M. Bringa, D. J. Benson, and M. A. Meyers. "Void growth in metals: Atomistic calculations." *Acta Materialia*, vol 56: pages 3874-3886, 2008.

ABSTRACT OF THE DISSERTATION

A Molecular Dynamics Study of Void Initiation and Growth in Monocrystalline and Nanocrystalline Copper

by

Sirirat Traiviratana
Doctor of Philosophy in Engineering Sciences (Mechanical Engineering)
University of California San Diego, 2008
Professor David J. Benson, Chair
Professor Marc A. Meyers, Co-Chair

MD simulations in monocrystalline and nanocrystalline copper were carried out with LAMMPS to reveal void growth mechanisms. The specimens were subjected to both tensile uniaxial and hydrostatic strains; the results confirm that the emission of (shear) loops is the primary mechanism of void growth. The expansion of the loops and their cross slip leads to the severely work hardened layer surrounding a growing void. Calculations were carried out on voids with different sizes, and a size dependence of the stress response to emitted dislocations was observed, in disagreement with the Gurson model [1] which is scale independent. The growth of voids simulated by MD is compared with the Cocks-Ashby constitutive model and significant agreement is found. The density of geometrically-necessary dislocations as a function of void size is calculated based on the emission of shear loops and their outward propagation. Calculations were also carried out for a void at the interface between two grains sharing a tilt boundary. The results show similar dislocation behaviors.

A code that uses Voronoi tessellation for constructing nanocrystalline structures was developed and used to prepare the structures for simulations. Nanocrystal simulations reveal grain sliding and grain rotation as the nanocrystal deformed.

Voids were nucleated at grain junctions and grew to coalescence as dislocations accommodated the material transfer.

A code that can be used during post-processing to extract useful dislocation information from MD simulation data was partially developed and proved the feasibility of automatically analyzing dislocations.

1

Introduction

1.1 Background

Classical, or Newtonian mechanics has become an important tool which physicists use to describe numerous phenomena. It has been developed and used since Newton himself stated his three laws: Law of motion, Law of inertia and Law of action and reaction. Calculus is another tool which was developed to fully express these laws. Motions, velocities, accelerations, forces, momentums and energies are defined and used through the equations derived from the laws of Newton. Classical mechanics can be used to describe motion of point particle very effectively. As studies move toward more complex systems, more advanced mechanics have been introduced, such as Lagrangian and Hamiltonian mechanics. Quantum mechanics and Relativity theory were later developed where Classical mechanics became invalid. Continuum mechanics employs techniques of classical mechanics in order to study the behavior of continuum materials. Solid mechanics is simply a branch of Continuum mechanics. More recently, the mechanics of materials has been developed, taking into account the complex microstructures of materials. Computational methods play a key role in the mechanics of materials.

The fundamentals of solid mechanics were established when the need to describe the behavior of solid continuum medium surfaced. The definitions of various quan-

tities were created, such as deformation, stress and strain, in order to effectively describe transformations of a body at rest into a deformed body. The deformation of a deformed body relative to a body at rest is called strain and the intensity of the force needed for the deformation is called stress. When the strain is small and is directly proportional to stress, the state is called linearly elastic. The coefficient of proportionality is called the modulus of elasticity or Young's modulus. When the stress reaches and passes a threshold, the yield stress, the body is deformed plastically and cannot recover its original state after being unloaded. This state is called plasticity.

Continuum mechanics enforces the following rules [2];

- Equilibrium condition
- Compatibility condition
- Hooke's Law
- Conservation of mass
- Conservation of momentum
- Conservation of energy

Linear elasticity for metals is described by Hooke's law. The material is simply modeled as a spring with Young's modulus in place of spring constant. When the load is removed the material returns to the original shape. In case of a long rod, since it is one dimensional, the stress is directly proportional to strain with Young's modulus, E.

$$\sigma = E\varepsilon \tag{1.1}$$

In three dimensions, stress and strain are described with second order tensors and the modulus of elasticity is a fourth order tensor. For isotropic materials, its components are functions of Young's modulus, E and Poisson's ratio, ν.

$$\boldsymbol{\sigma} = \mathbf{D}(E, \nu) : \boldsymbol{\varepsilon} \tag{1.2}$$

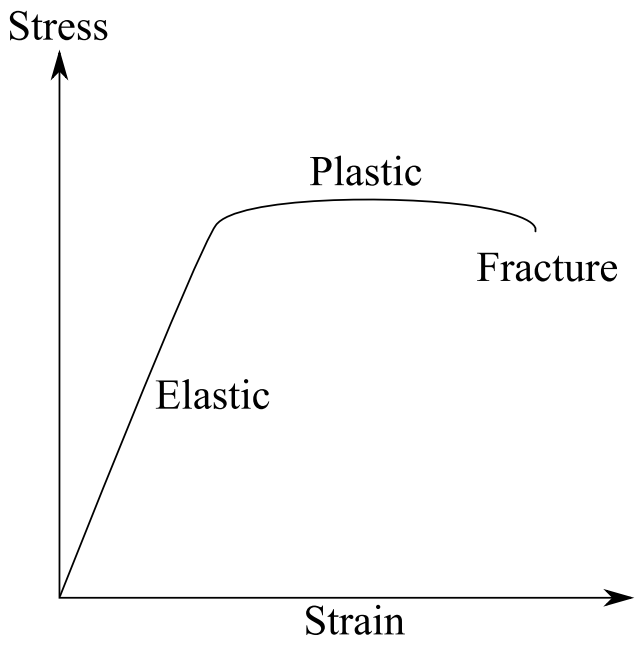


Figure 1.1: Stress-strain curve for general description of elasticity and plasticity.

Plasticity is a study of the state of strain which has passed the yield point, in one dimensional case, or passed a yield surface in three dimensional case, see Figure 1.1. At this state, material deformation can become greatly increased, softening, or greatly decrease, hardening. In any cases, the body have failed to return to the original shape when the load is removed. One of the basic yield surface is called J_2 yield theory. Hardening or softening parameters, also referred as flow rule, usually accompany with the yield criterion as it is used to predict the state of stress within the body passed yield. Often times, it is described in terms of strain rate, $\dot{\varepsilon}$ [2].

The Finite Element Method is a numerical tool which is used to solve partial differential equations governing a continuum mechanics system. It models the solution domain with shape functions and then assembly into governing equations which enforce required conditions according to continuum mechanics. FEM has its strong points where it is able to implement and solve problem with complex domain well. Another advantage is that FEM doesn't have intrinsic length scale built into the formulation. This enables it to solve problems in arbitrary size scale

without spending much of computation power, where as the length-scale has to be implemented in the material model or constitutive model. Models that include length scale are hard to produce and give accurate results only with its special system size. It cannot be used for general cases and this is FEM's draw back [3].

Since the development of X-ray diffraction, crystallography has evolved into one of the largest branches of material sciences. Lattice spaces are found to be the fundamental structure of material, enabling a well established structure for advanced studies about material behaviors. This leads to molecular dynamics(MD) studies/simulations which is based on particles representing atoms of material interacting under potential energy. The length scale of the system is already built into the MD calculation, which is a great advantage, but this also limits the system size as the present computing resources are not able to simulate a system large on the continuum scale. For stability, the integration timestep size has to be smaller than the vibration period of material atoms in order to capture this vibration and this limits MD to high strain rate simulations [3]. MD and crystallography will be discussed further in the next chapter.

Crystalline materials have been a topic of interest among researchers over the past decade, especially, nanocrystalline materials. It was predicted by the Hall-Petch relations [4, 5] that the material strength will increase as the grain size decreases. This drove researchers to find ways to reduce the grain size in order to obtain materials with great strength. There was also a report of inverse Hall-Petch relationship [6] which added to more confusion. As a result, a large volume of data under various conditions for nano materials was presented. The results were scattered, and each researcher tried to come up with a theory/mechanism which described their observed specific behavior. Only a limited range of experimental results can be compared to the numerical studies/simulations because of their limitations. As technology improves, more results from different studies will emerge and a better understanding of nanocrystalline materials will lead to more highly optimised materials [3].

1.2 Experimental Observations

Dislocation activity around a void growing in the spall regime of shock compressed copper was reported by Meyers and Aimone [7] and Christy et al. [8]. Fig. 1.2 contains evidence of dislocation activity surrounding a void with the plastically deformed zone [9]. In Fig. 1.2(a), a TEM micrograph of copper shows a peanut-shaped void. The picture was taken in the KRATOS, a 1 MeV TEM at the National Center for Electro Microscopy. Fig. 1.2(b) shows a spherical void in aluminum [9]. Fig. 1.3 shows slip bands emanating from voids that nucleated at grain boundaries in copper. However, the exact nature of the dislocation generation and evolution cannot be obtained from these observations. This requires detailed analysis methods such as molecular dynamics.

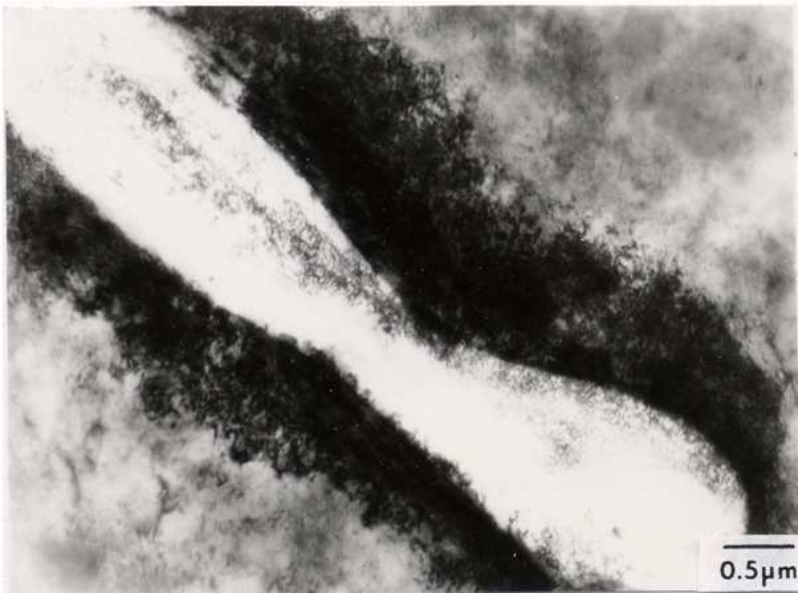
1.3 Literature Review

Fracture of ductile metals occurs by nucleation, growth and coalescence of voids [10, 11, 12]. There have been numerous studies trying to understand the mechanisms and to predict the strength of materials under different configurations, e.g., [13, 14, 15, 16]. The following subsections will briefly review some of the literatures that helped to guide my research in this field.

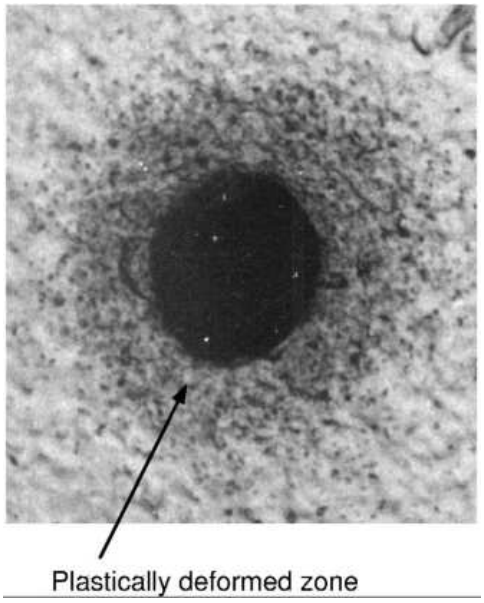
1.3.1 Simple Continuum Models for Void Expansion

The first continuum treatment of the expansion of void is based on a spherical hole under internal pressure [17]. An elasto-plastic solution was obtained. The original treatment is given in form of Lamé solution by Timoshenko [18] and Southwell [19]. Fig. 1.4(a) shows the diagram of a thick spherical shell with internal and external radii a and b subjected to the internal pressure p.

The stress components within the plastically deformed zone (defined by c in Fig. 1.4, $a \leq r \leq c$) are

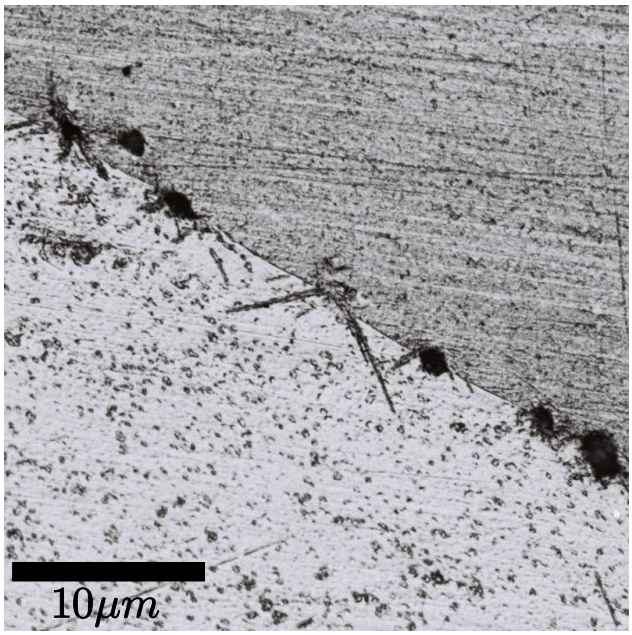


(a) Peanut-shaped void

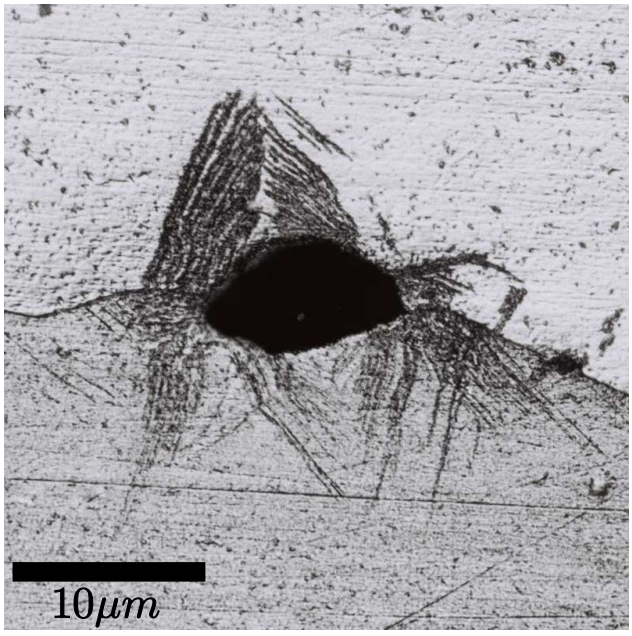


(b) Void with plastic region

Figure 1.2: Plastically deformed zone around a void, (a) TEM by Christy et al. [8]; (b) SEM by Ahn et al. [9].



(a) Small voids



(b) Large void

Figure 1.3: Evidence of slip around growing voids from Meyers and Aimone [7].

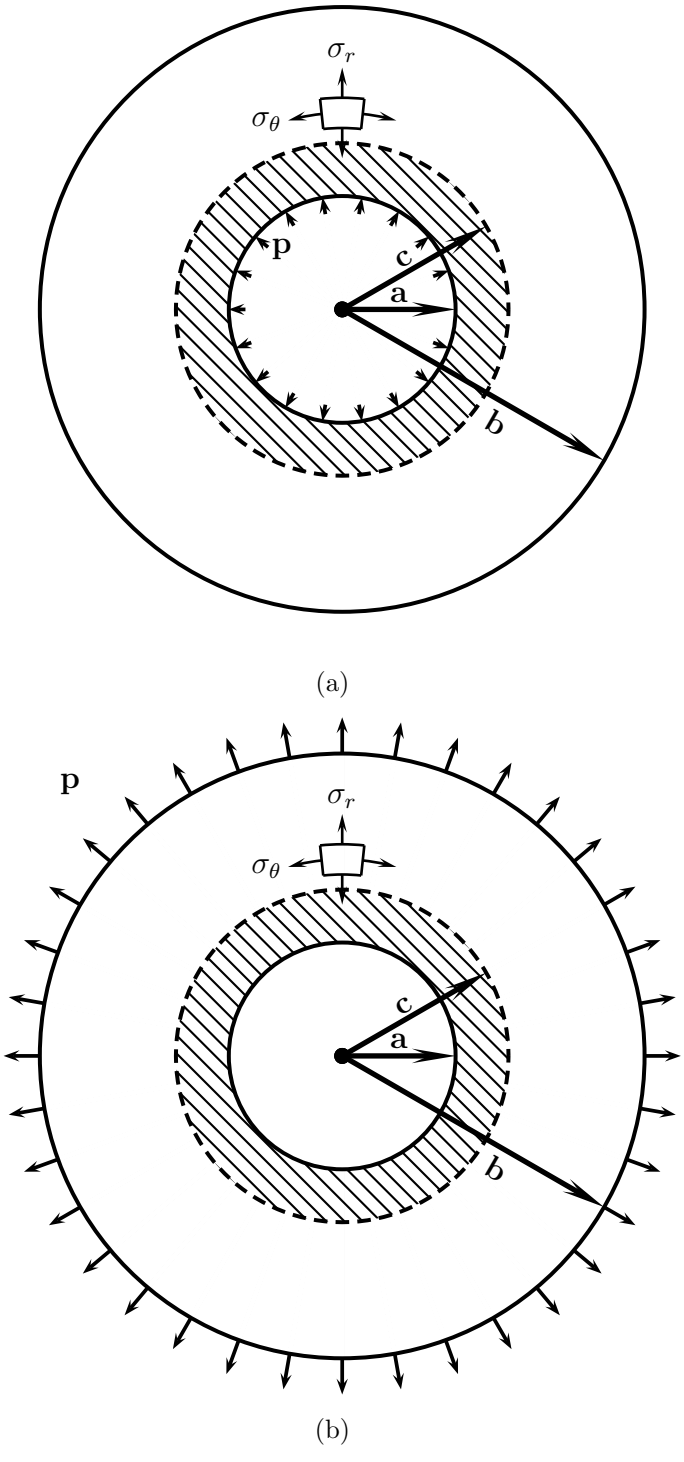


Figure 1.4: Plastic region round a spherical cavity expanded by uniformly distributed internal pressure.

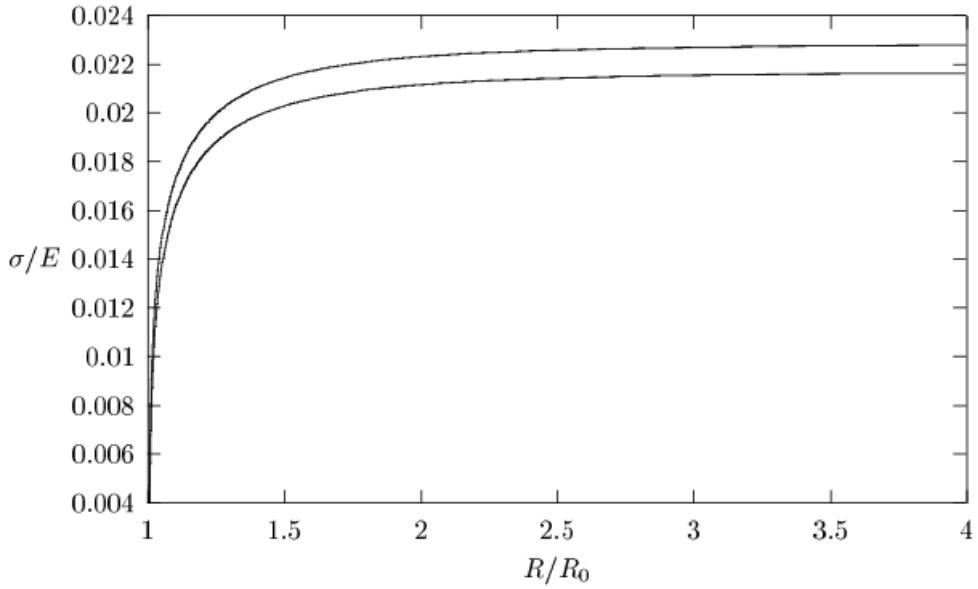


Figure 1.5: The normalized stress σ/E vs. expanded void radius R/R_0, with $Y/E = 3/500$ and two values of Poisson's ratio $\nu = 1/3$ (lower curve) and $\nu = 1/2$ (upper curve)(from Lubarda and Meyers [20]).

$$\sigma_r = -2Y \ln\left(\frac{c}{r}\right) - \frac{2Y}{3}\left(1 - \frac{c^3}{b^3}\right) \tag{1.3}$$

$$\sigma_\theta = Y - 2Y \ln\left(\frac{c}{r}\right) - \frac{2Y}{3}\left(1 - \frac{c^3}{b^3}\right) \tag{1.4}$$

where Y is the value of the yield stress. In case of a non-hardening material, we have $\sigma_\theta - \sigma_r = Y$ according to the von Mises or Tresca yield criterian. When the pressure is applied externally to the thick sphere, as in Figure 1.4(b), the current radius of the void in terms of the applied stress [20] is

$$R = R_0 \left[1 - 3(1-\nu)\frac{Y}{E}\exp\left(\frac{3\sigma}{2Y} - 1\right)\right]^{1/3} \tag{1.5}$$

The plot of normalized stress σ/E vs expanded void radius R/R_0 from the above Equation is shown in Fig. 1.5 [20].

1.3.2 Continuum Models for Material with Porosity

There have been several proposed continuum models for the growth of voids in both two and three dimensions. However, until recently there was no well established atomistic mechanism for void growth. Some exceptions are Meyers and Aimone [7] and Stevens et al. [21], who proposed a dislocation model for void growth in spalling, see Figure 1.6.

The following subsections summarize main ideas of some of the widely accepted methods.

Gurson Model

An approximate plastic constitutive theory was developed by Gurson which takes into account void nucleation and growth [1]. This constitutive theory provided the basis for many models which improved and modified Gurson's model to better model ductile fracture with porosity.

For spherical void and fully plastic flow, the yield function is

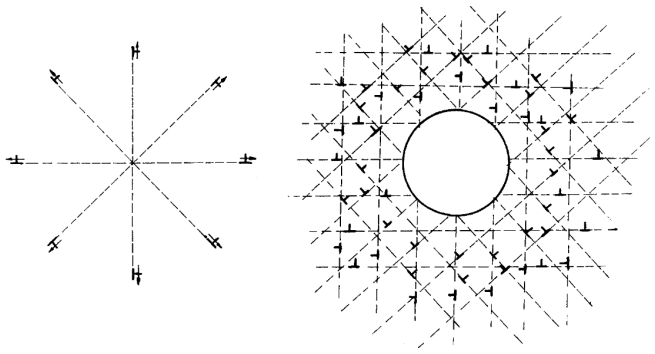
$$\Phi = \frac{\sigma_e^2}{\sigma_y^2} + 2f\cosh\left(\frac{\sigma_h}{2\sigma_y}\right) - 1 - f^2 = 0 \tag{1.6}$$

where σ_y is tensile yield stress of material, f is the randomly distributed void volume fraction, $\sigma_h = 3\sigma_m$, $\sigma_m = \frac{1}{3}\mathbf{Tr}(\boldsymbol{\sigma})$ or mean hydrostatic stress. σ_e is the equivalent von Mises stress.

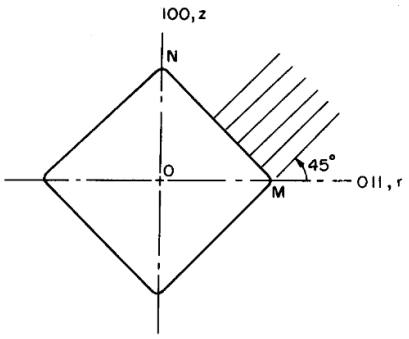
$$\sigma_e^2 = \frac{3}{2}\sigma'_{ij}\sigma'_{ij} = \frac{3}{2}\mathbf{Tr}(\boldsymbol{\sigma}'^2) = 3J_2 \tag{1.7}$$

$$\sigma'_{ij} = \sigma_{ij} - \sigma_m\delta_{ij} \tag{1.8}$$

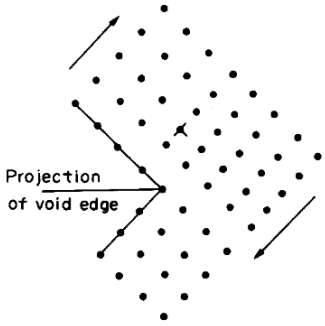
$\boldsymbol{\sigma}'$ is the deviatoric stress of the macroscopic Cauchy stress, $\boldsymbol{\sigma}$. The resulting yield function in Eq.1.6 has the properties of convexity and normality, which are used during analytical analysis as constraint conditions.



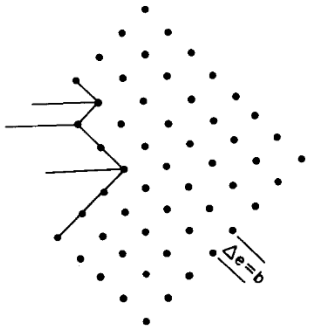
(a)



(b)



(c)



(d)

Figure 1.6: Dislocation models for void growth from (a) Meyers and Aimone [7], (b,c,d) Stevens and Davison [21].

Needleman and Tvergaard Modification of Gurson Model

In order to improve Gurson's model, Eq.1.6, a parameter q_1 was introduced by Tvergaard [22, 23] during the study of the behavior of a periodic array of voids which result in interaction between neighboring voids.

$$\Phi = \frac{\sigma_e^2}{\sigma_y^2} + 2f^* q_1 \cosh\left(\frac{\sigma_h}{2\sigma_y}\right) - 1 - (q_1 f^*)^2 = 0 \tag{1.9}$$

$$f^*(f) = \begin{cases} f & \text{for } f \leq f_c \\ f_c + \frac{1/q_1 - f_c}{f_F - f_c}(f - f_c) & \text{for } f > f_c \end{cases} \tag{1.10}$$

The results from shear band instabilities were compared with continuum constitutive relations and suggest that $q_1 = 1.5$. Furthermore, the function $f^*(f)$ was introduced [24] to model the loss of load-carrying capacity associated with void coalescence happening at void spacing of the order of the void diameter/length [25].

f_c is the critical value of the void volume fraction, when the material capacity to carry stress starts to decay rapidly. f_F is the actual void volume fraction associated with the complete loss of stress-carrying capacity [25].

LS-DYNA Implementation

The LS-DYNA manual [26] adds an additional parameter q_2 and uses the same $f^*(f)$ as shown in Eq.1.10.

$$\Phi = \frac{\sigma_e^2}{\sigma_y^2} + 2f^* q_1 \cosh\left(\frac{q_2\sigma_h}{2\sigma_y}\right) - 1 - (q_1 f^*)^2 = 0 \tag{1.11}$$

Wen et al. Modification of Gurson Model

The flow stress derived from Taylor dislocation model [27] was used by Wen et al. [28] to replace tensile yield stress, σ_y, in Gurson's model [1]. For a rigid-perfectly plastic solid with yield stress σ_y, the flow stress becomes

$$\sigma_{flow} = \sigma_y \sqrt{1 + \sqrt{\frac{5}{2}} L f^{1/3} \frac{b^4}{r^4}} \tag{1.12}$$

where L is given in Eq. 1.15, b is Burgers vector and r is the distance away from the void center. For a rigid-perfectly plastic solid, the J_2 flow theory of plasticity gives the microscopic deviatoric stress in terms of the microscopic strain rate by (for example, Hill [17])

$$\sigma'_{ij} = \frac{2\dot{\varepsilon}_{ij}}{3\dot{\varepsilon}} \sigma_y \tag{1.13}$$

Replacing σ_y by σ_{flow} in Eq.1.12

$$\sigma'_{ij} = \frac{2\dot{\varepsilon}_{ij}}{3\dot{\varepsilon}} \sigma_y \sqrt{1 + \sqrt{\frac{5}{2}} L f^{1/3} \frac{b^4}{r^4}} \tag{1.14}$$

$$L = \frac{E_{kk} l}{a} = \frac{\mathbf{Tr}(\mathbf{E})}{a} \tag{1.15}$$

Where $\mathbf{E}$ is the macroscopic strain, l is intrinsic material length, $l = 18\alpha^2 \left(\frac{G}{\sigma_y}\right)^2 b$, a is void radius. α is an empirical material constant, $\alpha \approx 0.3$, G is shear modulus and b is Burgers vector. f is void volume fraction and r is distance away from void center, $r = a$ at void surface.

After integration and solving the set of constraint equations given in Gurson's analysis [1], the yield function can be symbolically written as

$$\Phi\left(\frac{\Sigma_e}{\sigma_y}, \frac{\Sigma_{kk}}{\sigma_y}, f, L\right) = 0 \tag{1.16}$$

Wen et al.'s yield function [28] doesn't consider the effect of f^* suggested in Needleman and Tvergaard's model [25]. It can be seen that Wen et al.'s yield function introduces a scale in the void equation.

P-α Model

The separation of the volume change due to pore collapse from the compression of the matrix material allowed the construction of a thermodynamically consistent constitutive relation which covers most of the observed features of stress wave propagation in porous materials [29]. It has been shown that functions for the compaction relation, $\alpha = g(P)$, and equation of state, $P = f(V, E)$, suffice to fit experimental Hugoniot data for porous iron and aluminum.

The porosity, α, is defined as $\alpha = V/V_s$, E is the specific internal energy, V is the specific volume of porous material and V_s is the specific volume of the corresponding solid material at the same temperature and pressure. The porosity becomes unity when the material is solid.

There are a few assumptions that have been made in this theory: 1) voids do not reopen in the time scales of interest in stress wave propagation, 2) shear strength is negligible throughout, 3) neglect of the melting phase change limits the pressure range over which this theory might be expected to be applicable.

Later work [30] suggested a modification of $P = f(V/\alpha, E)$ to $P = \alpha^{-1} f(V/\alpha, E)$. The discrepancy between the original equation and the modified one is significant in the lower pressure range, where the porosity α is different from unity.

1.3.3 Single Crystal Studies

Several studies on void growth and coalescence, for example, [31, 32, 33, 34, 35, 36, 37, 38, 39], have shown the strength of the finite element method that one can vary several parameters and loading configurations to investigate the parametric influences on the growth of void. Nevertheless, the results only show plastic deformation in the range of micro meter scale, not at the atomistic level. To achieve the dislocation level of plastic deformation, slip, one must use the molecular dynamics technique.

Geometrically Necessary Dislocations

Ashby [40] suggested that materials that are composed of two or more phases with different mechanical properties will deform plastically in a non-uniform way with one phase deforming preferentially. This includes polycrystalline and nanocrystalline materials. The difference in deformation results in a gradient of deformation. Such materials are called plastically non-homogeneous. In order to have different phases of material deform in a compatible way, "geometrically necessary" dislocations were proposed as a characteristic of the microstructure in addition to "statistically stored" dislocations which occur in homogeneous materials, see Figure 1.7. The density and arrangement of geometrically necessary dislocations can be calculated in an iterative manner. Starting with the incompatible deformation among phases, the dislocations are introduced which produce the deformation, then the stress field of this array of dislocations is calculated. If the stress exceeds local yield stress, then new dislocations must be introduced to lower the stress. When the density of geometrically necessary dislocations exceeds that of the statistically stored dislocations, geometrically necessary dislocations control the stress-strain curve and the work-hardening rate is higher.

Void Growth by Dislocation Emission

It was found in a high strain rate laser shock experiment that voids initiated and grew as the material deformed. Lubarda et al. [41] proposed that voids grow by dislocation loop emission. The vacancy diffusion mechanism is unable to provide fast enough transportation for the void to evolve, and therefore dislocations were suggested to be the main mechanism in a combination of prismatic and shear loops. Prismatic loops had been proposed earlier by Jones and Mitchell [42], Silcox and Hirsch [43] and Humphreys and Hirsch [44], but the shear loops were a new mechanism. Seitz [45] and Brown [46] postulated prismatic loops forming at the interface between a rigid particle and the matrix. As the energy of dislocations are calculated for different size of voids, it was found that the critical stress required

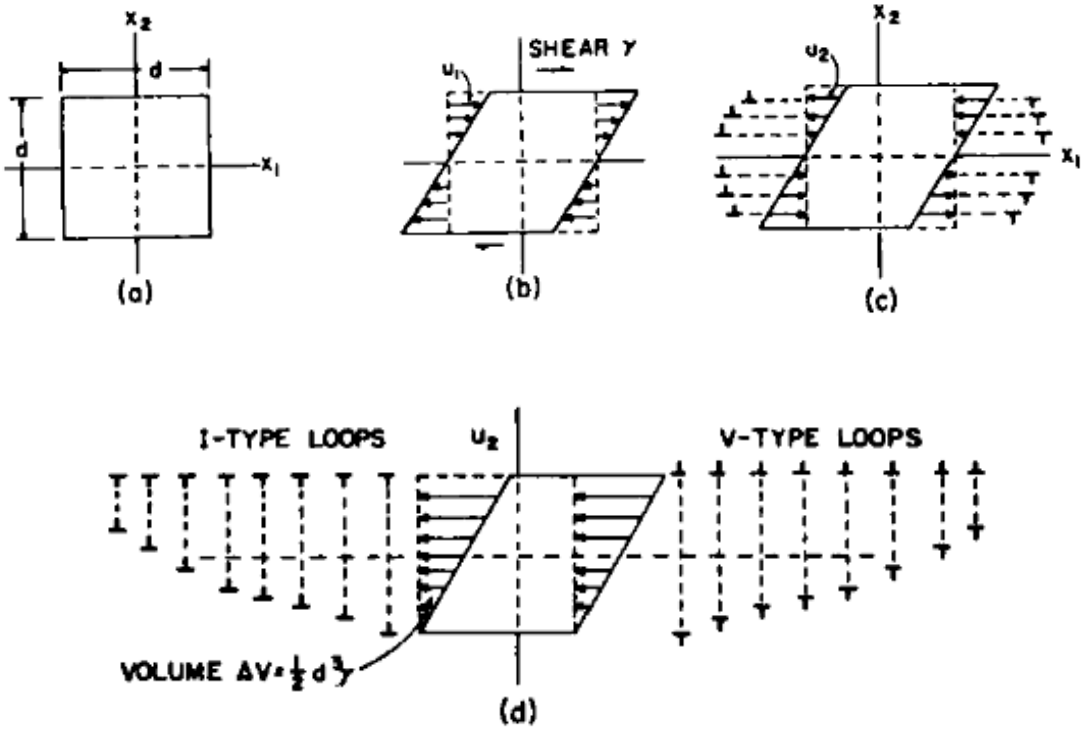


Figure 1.7: Geometrically necessary dislocation around a rigid particle in softer material, from Ashby [40], (a) undeformed model, (b) deformed model, (c) shear loop dislocations, and (d) prismatic loop dislocations.

to form dislocations decreases as the void size increases [41, 47]. Figure 1.8 shows these loops as proposed by Lubarda et al. [41].

There is a dearth of information on the void initiation process, and it is generally assumed that it is governed by the diffusion of vacancies towards a central point, creating and nourishing a void. One of the most rapid diffusion mechanisms is "pipe" diffusion, in which vacancies migrate along the dislocation line. Cuitiño and Ortiz [48] developed a specific mechanism for this mode, with the following equation (Eq. 1.17) predicting the time change of void radius, R, in terms of the pipe diffusion coefficient, D,

$$\frac{dR}{dt} = \frac{1}{R} D \left(c_0 - c_{eq} \right) \tag{1.17}$$

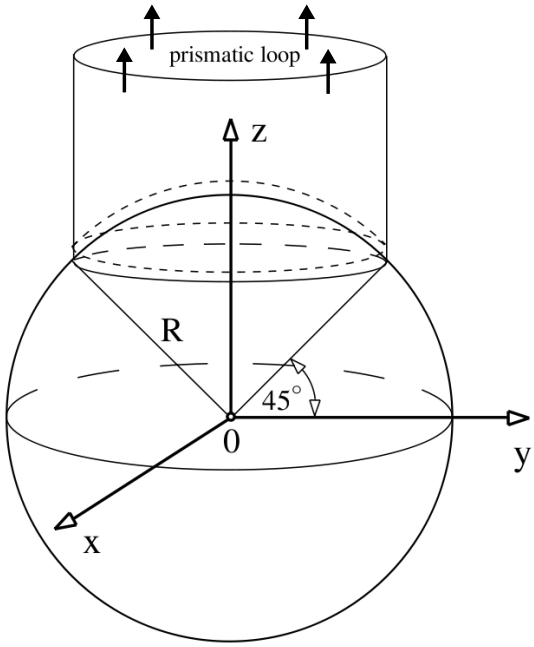
where c_0 is the plastic deformation void concentration and c_{eq} is the equilibrium vacancy concentration at the surface of the void. It is safe to assume $c_0 \gg c_{eq}$ since $c_0 \sim 10^{-4}$ after significant plastic deformation and $c_{eq} \sim 3 \times 10^{-15}$ at room temperature [48]. Integrating with respect to time,

$$\frac{R}{R_0} = \left(1 + \frac{2 D c_0}{R_0^2} t \right)^{1/2} \tag{1.18}$$

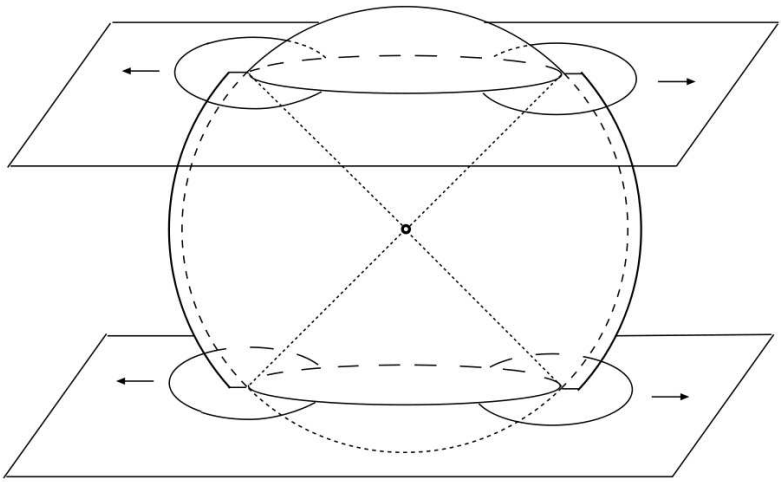
where

$$D = D_0 \exp\left(-\frac{Q}{RT} \right) \tag{1.19}$$

The initial void radius, R_0, was taken as the atomic radius for copper (a model material), 0.128 nm. D_0 is the pre-exponential factor in the diffusion coefficient ($= 4.86 \times 10^{-7} \mathrm{m}^2/\mathrm{s}$); Q is the activation energy ($= 72.47 \times 10^3 \mathrm{J/mol}$). These values are taken for ultra pure copper from Surholt and Herzig [49]. R is the gas constant, T is the temperature in Kelvin and t is the time in seconds. Figure 1.9 shows the growth of a void at 300, 400 and 600 K for copper. Conventional plastic deformation at a conservative strain rate $10^{-2} s^{-1}$ will lead to a failure time of 102 s assuming a strain of 1. Failure is typically characterized by voids with radii ranging in the micrometers. From Fig. 1.9, it can be seen that the time predicted by Cuitiño and Ortiz [48] at 300 K is much longer ($\sim 10^{10}$ s). Even at 600 K, voids



(a)



(b)

Figure 1.8: Dislocation loops postulated by Lubarda et al. [41] with the direction of dislocation motion marked by arrows. (a) Prismatic loops and (b) shear loops.

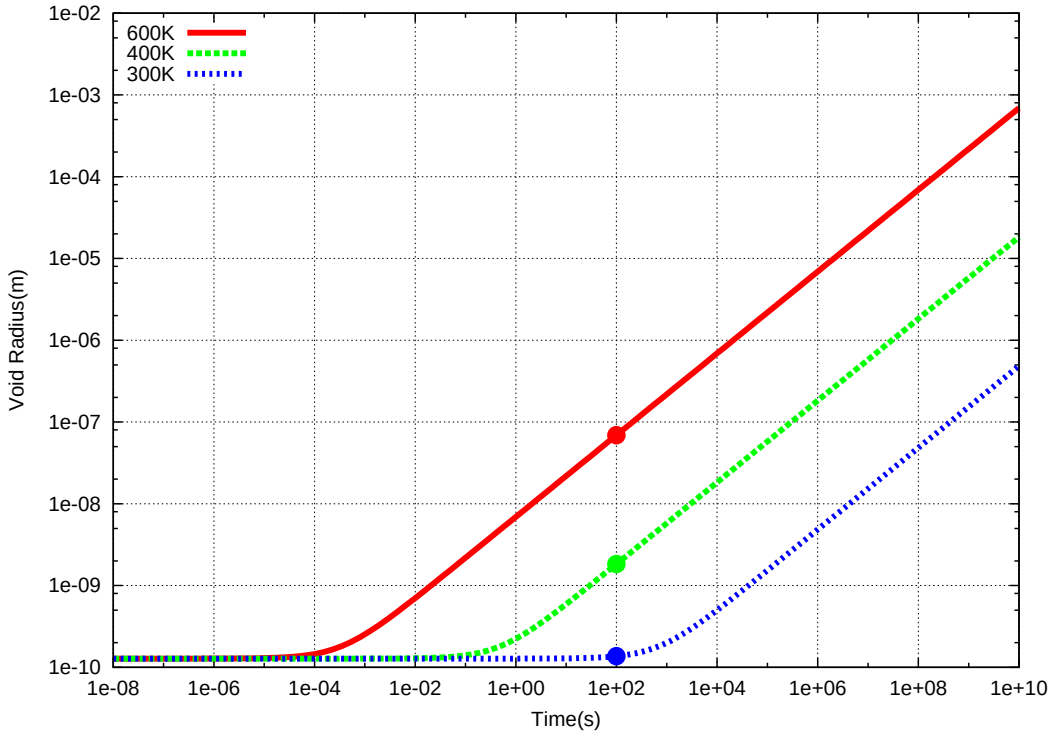


Figure 1.9: Void-size prediction from growth through vacancy diffusion along dislocations (Cuitiño-Ortiz model); void sizes reached in 100 s at the three temperatures of 300, 400, and 600 K marked.

cannot grow to a size equal to 0.1 μm in 10^2 s (Fig. 1.9). Thus vacancy diffusion, that is the principal mechanism of void growth in creep fracture, as treated by Raj and Ashby [50], cannot be the operating mechanism in conventional plastic deformation.

The model proposed by Cuitiño and Ortiz [48] is based on vacancy pipe diffusion which was proved effective only for low strain rate and/or high temperature by Lubarda et al. [41] and therefore this model is only relevant to creep deformation. Failure in creep is preceded by void nucleation and growth at the grain boundaries and has been successfully modeled using the diffusion equation by Raj and Ashby [50]. The regime encountered in laser shock compression is radically different, with strain rates on the order of 10^6 s^{-6} and higher. The characteristic material length, determined by the dislocation structure in materials, can also become a limiting

factor for growth of void at small size [51].

Expansion of Single Crystalline Materials

The continuum approach by Ohashi [52] and the molecular dynamics simulations by Seppälä, Belak and Rudd [53, 54, 55], Marian, Knap and Ortiz [56, 57], Srinivasan et al. [58] and Gungor and Maroudas [59] indicate that the dislocation emission from growing voids is the primary mechanism of the radial material transfer required for expansion. Void collapse calculations [60] lead to similar (but opposite in sign) dislocation configurations. Both prismatic and shear loops were postulated [41] and also observed in the above mentioned MD simulations and in the recent work by Ahn and Sofronis [9]. Figure 1.8 shows the two types of dislocation loops. It should be noted that Ashby [40] had also postulated prismatic and shear loops in the deformation of metals containing rigid particles to accomodate the strain gradients imposed: these are the "geometrically necessary dislocations". Size scale dependency of the yield stress for a fixed void volume fraction is observed by Potirniche et al. [61].

Atomistic simulations performed by Horstemeyer et al. [62] in simple shear using between 10^2 and 10^8 atoms revealed significant differences in the flow stress when expressed as a function of a scale parameter (volume/surface of sample). The resolved shear stress for plastic flow increases significantly with the decrease in size scale, confirming experimental measurements related to gradient plasticity effects (e.g., Fleck et al. [63, 64]). Interestingly, the MD results indicate that dislocation nucleation effects, and not strain gradient effects (calculations in simple shear do not produce strain gradients), are responsible for the significant differences in shear flow stress obtained with the change in dimensional scale. These results have a significant bearing in what is perceived to be gradient plasticity.

Studies of void growth in single crystal materials, [65, 66] showed that yield strength also varies with size scale of the domain, not only the void volume fraction and strain rates. One interesting feature of void growth in single crystal FCC

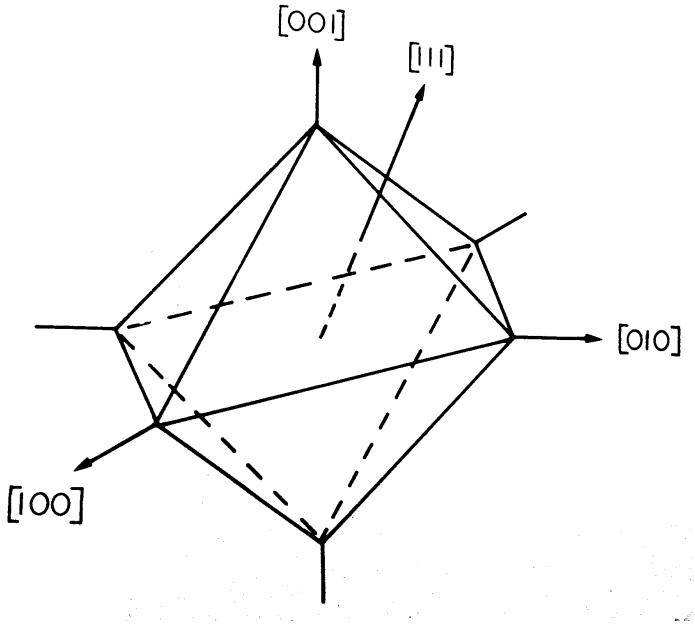
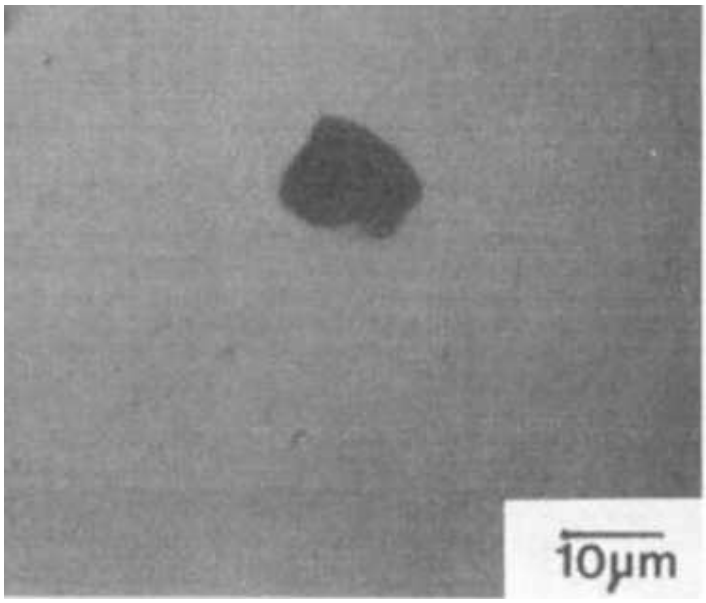


Figure 1.10: Growing void developed into an octahedron shape defined by slip planes.

materials is that the void shape developed into an octahedron with faces parallel to FCC slip planes, Fig. 1.10. This is the result of the dislocation mechanism that transports material along the slip directions Experimental observations by Meyers et al. [7, 8, 67] show that small voids (smaller than the grain size) often had this geometric shape, Figure 1.11. This was also observed in aluminum by Stevens et al. [21], Figure 1.12. Later, the study of stress triaxiality effects on void growth, [53] showed that there are strong relations between the stress triaxiality and rate of growth of the void.

Fatigue Loading of Single and Nanocrystalline Materials

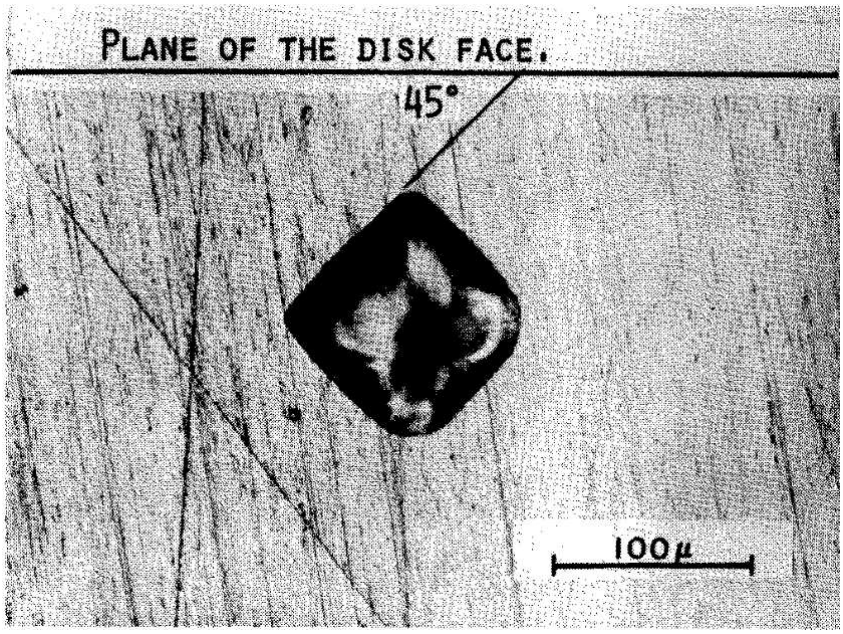
Studies of fatigue loading in single crystal copper by Potirniche and coworkers [68, 69] indicated that the shear band and the crack propagation directions depend heavily on the slip direction or orientation of the crystal. For fatigue loading in nanocrystalline materials, the growth of the crack across a grain boundary with



(a)

(b)

Figure 1.11: Growing voids developed into geometric shapes from (a) Meyers and Aimone [7] and (b) Christy et al. [8].



(a)

Figure 1.12: Growing void developed into geometric shape from Stevens and Davison [21].

a high mis-orientation angle can change direction when moving from one grain to another [70].

1.3.4 Nanocrystal Studies

The study by Ashmawi and Zikry [71] showed that mobile dislocation density saturation, void size and shape, and dislocation density interactions within the grains and the grain boundaries are interrelated triggering mechanisms that lead to void nucleation, growth and localization.

Deformation Mechanism in Nanostructured Materials

Nanocrystalline materials behave much differently from polycrystalline materials in general. They have high strength, as predicted by the Hall-Petch relation, and low ductility. This is because of their low hardening rate which allows failure via strain localization and their inability to allow extensive plastic deformation such as crack progression. Many deformation mechanisms have been suggested to account for specific responses [3];

- Pile-up breakdown.
- Grain-boundary sliding.
- Core and mantle effect.
- Grain-boundary rotation and grain coalescence.
- Shear band formation.
- Gradient models.
- Twinning.
- Grain-boundary dislocation creation and annihilation.

Expansion of Nanocrystalline Material

Void nucleation and growth were studied by Belak, Rudd and coworkers [65, 72, 73, 74]. They found that nucleation happens at triple junctions while the

growth of the void progresses along the grain boundary. Dislocations are the main mechanism which transport material allowing the void to grow.

Compression of Nanocrystalline Material

High pressure-high strain rate compression simulations in ductile porous metals showed the formation of nano-grains resulted from highly localized plastic deformation induced by the presence of voids. The voids served as dislocation sources during the process of grain formation. The collective interactions later lead to a very high dislocation density and the reduction of the dislocation velocities as the porosity decreased [75].

1.4 Outline of the Problems

Most metallic materials used in engineering are polycrystalline aggregates and are highly susceptible to nano-scale failure induced by extreme conditions. In order to effectively predict the overall failure of a structure, one must understand the mechanism of failure propagation created from the atomic level. Material models of finite element methods currently do not account for the mechanism at the atomic/nano levels. Most of the molecular dynamics studies, show only the quantitative mechanisms of dislocations at the atomistic level, and provide us with mechanisms by observation and simple calculations because the interactions of dislocations are complex and vary from one simulation to another. A consistent link between dislocation mechanisms and statistical mechanics is still missing, resulting in a missing relation of nano-scale mechanisms to micro-scale mechanisms. Once the link has been established, continuum mechanics research would benefit greatly from future molecular dynamics studies.

1.5 Objectives of Research

Dislocations are the main mechanism by which crystalline metals locally relax the stresses induced by deformation of the entire body. Dislocations also grow from grain boundaries and grain junctions in the case of polycrystalline and nanocrystalline structures. Dislocations change direction and plane according to the orientations of adjacent grains. The rate of multiplication or annihilation of different dislocations (which run into each other) should be statistically studied and summarized into the possibility at which the failure can occur. With the availability of vast amount of information one can obtain from MD simulations, and the proper computational routines to separate useful information out from complex interactions, the link between atomistic scale mechanisms and the continuum level can be established.

1.6 Acknowledgement

Some figures and materials in sections 1.2 and 1.3 also appear in S. Traiviratana, E. M. Bringa, D. J. Benson, and M. A. Meyers. "Void growth in metals: Atomistic calculations." *Acta Materialia*, vol 56: page 3874-3886, 2008.

2

Molecular Dynamics Fundamentals

2.1 The Molecular Dynamics Simulation Process

In general, performing a molecular dynamics study requires a set of procedures which can be roughly organized into domain construction, equilibrium and relaxation, objective run, and post processing. All of these steps are normally repeated for different models or different simulations. The third step, or objective run, can be repeated with different kinds of loading conditions and then followed by a post processing session, which allows us to start each simulation at the same exact domain configuration.

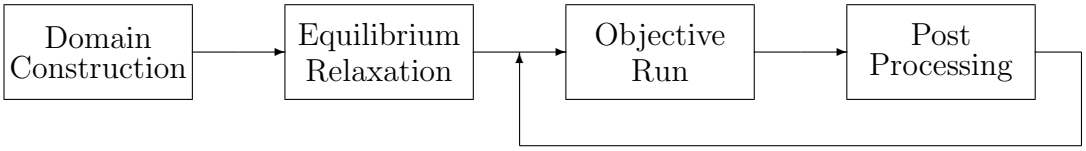


Figure 2.1: Flow chart of general MD simulation.

2.1.1 Domain Construction

Domain construction is a process where particles and their locations are generated. The information includes the size and organization of the microstructure. Most of the time particles will be organized into a crystal structure according to the type of metal or material and their lattice data. There can be one type of crystal for the entire domain, i.e., a single crystal, or a combination of two or more crystals in one domain. The latter will simulate a bi-crystal or polycrystal structure. There has to be a configuration that defines how different crystals are located next to, or against, other crystals. This leads to bi-crystal and polycrystal structure construction which will be explained later.

2.1.2 Equilibrium and Relaxation

Once all particles are imported or generated, the next process is to equilibrate all the particles, corresponding to their lowest energy state. The objective is to start with a state that is close to the real material crystal. Voids or holes can be introduced inside of a domain leaving empty spaces inside crystals. The surface particles of these spaces will have high potential energy compared to "regular" crystal particles and need to be moved into new positions with lower energy. Boundary conditions are usually assigned prior to this process. Both equilibrium and relaxation processes usually take place in predefined environments, for example, isothermal or isobaric conditions, to bring the status of a crystal into a specified temperature and pressure.

2.1.3 Objective Run

In this step, the domain is subjected to applied forces. Typical loading conditions include expansion, compression, and shock loading. During a shock compression simulation, a set of particles will be assigned with a piston velocity so that the entire group of particles will impact the rest of the domain, creating a shock

front that can travel through the domain in the direction of the piston velocity.

During an expansion simulation, the volume of the whole domain will be rescaled to a bigger size, forcing the particles inside of the domain to readjust their locations to new positions according to forces from the potential energy profile.

The timestep, Δt, which defines the step of integration time is usually fixed to about 1 femtosecond. This is to ensure the stability of numerical time integration. The number of timesteps (N) defines how long an entire simulation takes place and will dictate the strain rate, $\dot{\varepsilon}$, in the case of finite strain expansion or compression of the domain,

$$\dot{\varepsilon} = \frac{\varepsilon}{t} = \frac{\varepsilon}{N\Delta t} \tag{2.1}$$

where $N\Delta t$ is equal to number of timesteps multiplied by the timestep size (about 10^{-15} seconds). Since molecular dynamics takes into consideration a large number of particles and their properties, the simulation time is short, typically less than a nanosecond. In order to run for thousands of picoseconds, the number of particles has to be about 10^7 atoms with the computing power of up to 200 CPUs for EAM potentials. In any case, the strain rate for general MD simulation will be high, roughly 10^8 s^{-1} or greater, because, given $\varepsilon = 0.10$ or 10%, $\Delta t = 10^{-15}s$ and $N = 10^6$ steps;

$$\dot{\varepsilon} = \frac{0.10}{10^6 10^{-15} s} = 10^8 s^{-1} \tag{2.2}$$

2.1.4 Post-processing

During a simulation, the location of all particles can be written into files accompanied with their other parameters and attributes such as velocities, forces, stresses, particle ID and centrosymmetry parameter. Often times, this information will be collected at a certain number of timesteps as snapshots at different points during the whole simulation. Commercial software, e.g. RasMol [76], or

user created code (MDrender [77]) will be used to read in this information and render particle locations and their attributes into images where the investigator can examine the motion and behavior of the particles during the simulation. Color coded particles are usually used to represent a variation of particle attributes such as temperature and pressure. Studies of crystals and defects are mostly done using this process and much useful information can be extracted for later analysis.

2.2 Molecular Dynamics Codes, LAMMPS

LAMMPS is a molecular dynamics code that models the behavior of particles in a liquid, solid, or gas state. It can model atomic, polymeric, biological, metallic, granular, and coarse-grained systems using different types of force fields and boundary conditions [78].

LAMMPS integrates Newton's equations of motion for collections of atoms, molecules, or macroscopic particles that interact under short- or long-range forces with a variety of initial and/or boundary conditions. For computational efficiency LAMMPS uses neighbor lists to keep track of neighboring particles. On parallel machines, LAMMPS uses spatial-decomposition techniques to partition the simulation domain into small 3D sub-domains, each one assigned to a different processor. Processors communicate and store "ghost" atom information for atoms that surround their sub-domain. LAMMPS is most efficient when particles fill a 3D rectangular box with a roughly uniform density [78].

2.3 The Embedded Atom Method

The embedded atom method was developed to improve on previous pair potentials by accounting for the electron density. It views atoms as being embedded in the host containing all other atoms. The embedded energy is electron-density, ρ, dependent, where density is always definable [79, 80]. The total energy can be given by

$$E_{tot} = \sum_{i} F_i(\rho_{h,i}) + \frac{1}{2} \sum_{i,j}^{i \neq j} \phi_{ij}(R_{ij}) \tag{2.3}$$

where F_i is the embedded function of host density, ϕ_{ij} is the short-range pair potential and R_{ij} is the distance between atoms i and j. The host density ($\rho_{h,i}$) can be closely approximated by a sum of the atomic densities (ρ^a).

$$\rho_{h,i} = \sum_{j}^{j \neq i} \rho_j^a(R_{ij}) \tag{2.4}$$

where ρ_j^a is the contribution to the density from atom j and $\rho_{h,j}$ is the total host electron density at atom j. The energy is a simple function of the position of the atoms as implemented in LAMMPS [81].

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_\alpha(R_{ij}) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta}(R_{ij}) \tag{2.5}$$

The pair potential provides the attraction between atoms while the volume dependent term (suggested to be added to account for the compressibility of the electron gas [82]) serves to slightly expand the solid and gives the correct elastic constants. In contrast, the embedding energy is dominant and provides cohesion while the short-range repulsive pair interaction keeps the solid at a slightly larger lattice constant. Embedded energy has replaced the volume-dependent energy with an electron density-dependent one. This gives the advantage that electron density is always definable [80].

Potential energy profiles that are used in this research are the universal Cu interaction from the LAMMPS distributed package and the Cu potential from Mishin et al. [83].

2.4 Single-crystalline and Bi-crystalline Structure Generation

In order to generate an organized structure of crystal particles, one must know several parameters depending if the lattice is cubic or not. For cubic lattice, only two parameters are required. The first parameter is the lattice size (lattice parameter) of the material of interest and how the particles are packed into crystal structure (lattice type), for example, Body Centered Cubic (BCC), or Face Centered Cubic (FCC). This information can be looked up according to type of material that we want to model. Next, data describing how the crystal is oriented against the global coordinate system of the simulation domain in required. This data is usually determined by the simulation setting, for example one must align the <100> of the crystal to a preferred axis in order to apply the shock velocity in this direction. Examples of crystal structures are given later in Table 2.1

2.4.1 Basic Crystallography

X-ray diffraction was introduced into crystallographic studies in 1912. Later in the same year, the exact size and shape of fundamental units of structure of a crystal was found by Bragg. This leaded to the widespread use of X-ray diffraction for studies of crystallography and many other fields. Crystal structures that have a periodic nature can be classified into network of points in space which repeat themselves at a certain distance and direction. This is called a space lattice, see Figure 2.2. For a three-dimensional space lattice, three fundamental directions are needed [84].

Fig. 2.3(a) shows the three principal directions accompanied with their interaxial angles. There can be many possible combinations of lengths of principal directions and interaxial angles. However, there can only be seven crystal systems which are listed in Table 2.1, showing the relation of unit lengths and interaxial angles.

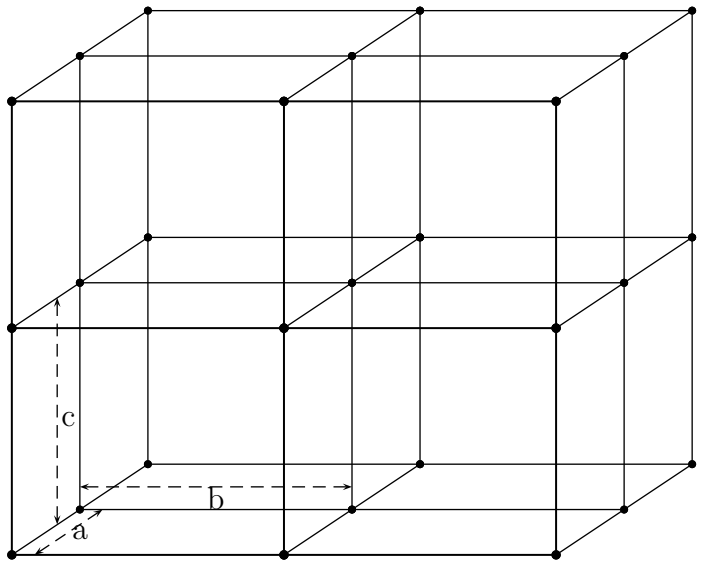
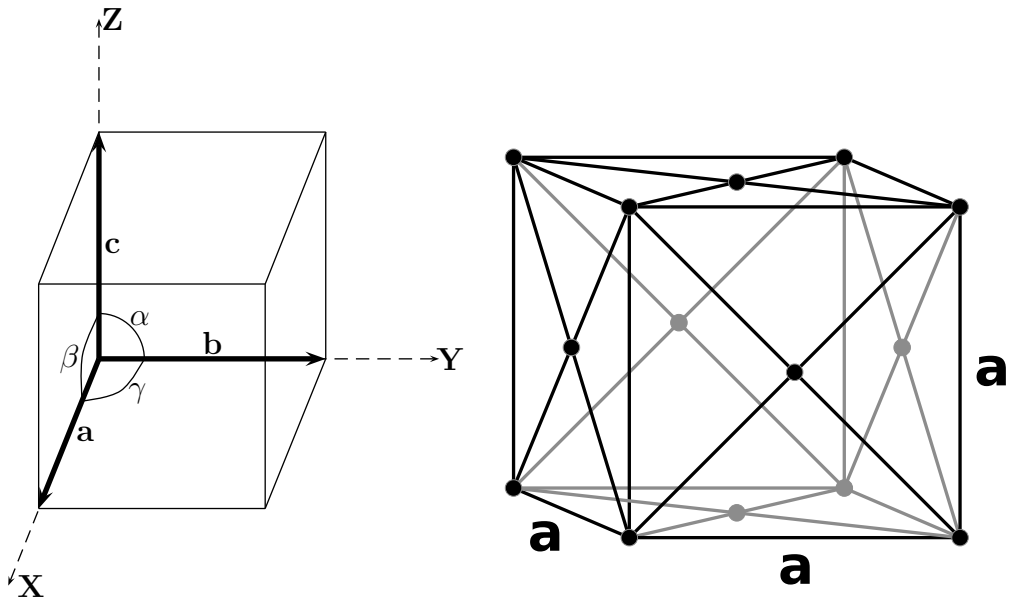


Figure 2.2: A space lattice according to Barrett and Massalski [84]

Table 2.1: The crystal systems [84]

System	Axes and interaxial angles	Examples
Triclinic	Three axes not at right angles, of any lengths	K_2CrO_7
	$a \neq b \neq c$ $\alpha \neq \beta \neq \gamma \neq 90°$	
Monoclinic	Three axes, one pair not at right angles, of any lengths	$\beta - S$
	$a \neq b \neq c$ $\alpha = \gamma = 90° \neq \beta$	$CaSO_4{\cdot}2H_2O$
Orthorhombic	Three axes at right angles, all unequal	$\alpha - S, Ga$
	$a \neq b \neq c$ $\alpha = \beta = \gamma = 90°$	Fe_3C
Tetragonal	Three axes at right angles, two equal	$\beta - Sn$
	$a = b \neq c$ $\alpha = \beta = \gamma = 90°$	TiO_2
Cubic	Three axes at right angles, all equal	$Cu, Ag, Au,$
	$a = b = c$ $\alpha = \beta = \gamma = 90°$	$Fe, NaCl$
Hexagonal	Three axes coplanar at 120°, equal	Zn, Cd
	Fourth axis at right angles to these	$NiAs$
	$a_1 = a_2 = a_3 \neq c$ $\alpha = \beta = 90°, \gamma = 120°$	
Rhombohedral	Three axes equally inclined, not at right angles; all equal	As, Sb, Bi $Calcite$
	$a = b = c$ $\alpha = \beta = \gamma \neq 90°$	



(a) (b)

Figure 2.3: (a) Lattice axes, interaxial angles and unit cell [84] and (b) configuration of atoms for the Face Centered Cubic structure with the lattice size of a, created by Bas Zoetekouw (bas@zoetekouw.net).

The cubic crystal system is the most simple case of all. There are three space lattices for the cubic crystal system, Simple Cubic(SC), Body Centered Cubic(BCC) and Face Centered Cubic(FCC). Fig. 2.3(b) shows the configuration of atoms in lattices. Since all sides of a cube have the same length, it is denoted as a or lattice spacing. For Copper(Cu), $a = 3.615$ Å. With the repetitive nature of lattice spacing, the term "unit cell" is used to define a group of atoms that repeats itself at specific directions and distances. For the cubic crystal system, these directions are the bases $\mathbf{a} = a\,[100]$, $\mathbf{b} = a\,[010]$ and $\mathbf{c} = a\,[001]$ for a fixed length of lattice spacing, a.

2.4.2 Single Crystalline Structure

The single crystal structure is the easiest and simplest type of crystal to generate. Once the basic type of crystal is known, the unit cell, which can be duplicated multiple times, can be calculated. As the size of a domain, and crystal orienta-

tion with respect to the global coordinates of the simulation domain are defined, the unit cell can be repeatedly copied to fill the entire domain according to their fundamental lattice directions. As a result, we have a simulation domain full of particles that are organized into the predefined orientation.

2.4.3 Bi-crystalline Structure

A domain needs to be divided into two parts with different orientations for each part. The unit cell of a known structure is multiplied into each part of the domain according to the specified orientation of that part. Directions of the crystal principal basis must be defined with respect to the global coordinate system of a simulation domain, prior to the copies of each unit cell. Some MD codes, such as LAMMPS, can handle this technique. LAMMPS allows the user to directly define three crystal directions to align with the global simulation coordinate system.

2.5 Polycrystalline Structure Generation

To simulate a polycrystalline structure which can closely model the natural organization of grains in a metallic material, most studies rely on Voronoi tessellation, for example, Kadau et al. [85].

2.5.1 Voronoi Tessellation

The Voronoi diagram, or Voronoi tessellation, is a special kind of area/volume decomposition determined by distances to a set of points in 2D or 3D space. It has been widely used in computational geometry. For instance, the Voronoi diagram is a possible solution for the cellphones and cellphone towers problem. Let us assume the following;

- There are n cellphone towers distributed in an area.
- One cellphone connects to a cellphone tower at a time.

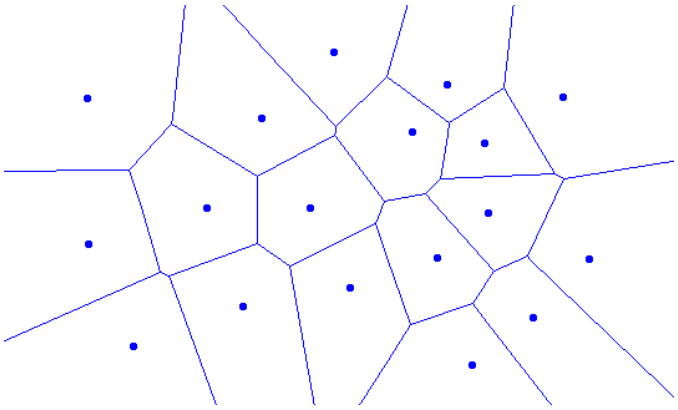


Figure 2.4: 2D Voronoi diagram.

- A cellphone will connect to the closest cellphone tower.

Voronoi tessellation will divide an area into number of cell sites corresponding to cellphone towers, where any cellphones that lie in this particular cell site will connect only to the cellphone tower in this site. This allows a cellphone to only connect to the closest tower and not to any other towers. Another cellphone that lies in another cell site will connect to their respective cellphone tower. This is the optimum solution where the radio wave will travel the shortest distance to the tower. At the same time, the Voronoi diagram divides the 2D spaces into many convex polygons. The convexity is important to many applications in physics, astronomy, robotics and other fields [86].

Voronoi tessellation was used in this investigation to generate the grain configuration in polycrystalline samples. All Voronoi tessellations in this work was generated by a software called "Qhull". Qhull computes the convex hull, Delaunay triangulation, Voronoi diagram, and much more. The software runs in 2D, 3D, and higher dimensions. Qhull makes use of the Quickhull algorithm to compute the convex hull. It handles roundoff errors from floating point arithmetic. Qhull also computes volumes, surface areas, and approximations to the convex hull [87].

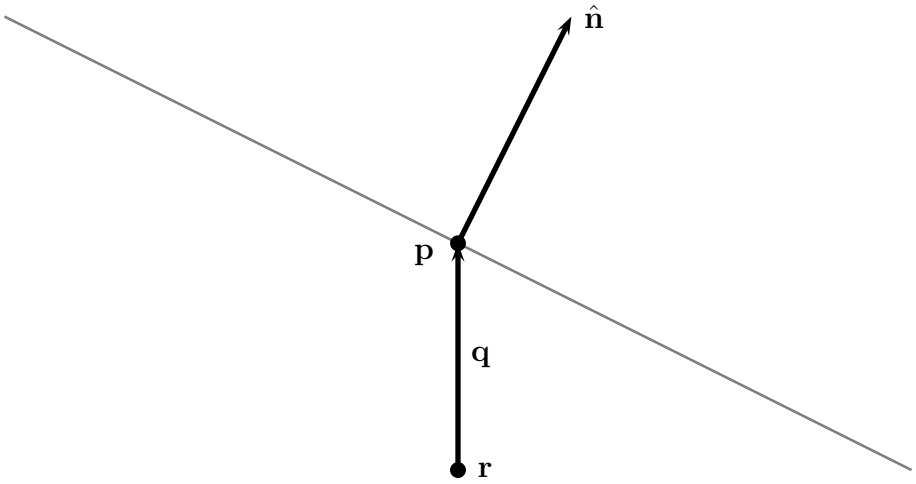


Figure 2.5: Plane and Normal vector sketch

2.5.2 Vectors and Planes

Each particle is described by a position vector containing **x**, **y**, and **z** coordinates. A plane is defined by two vectors: a point on the plane, and a normal vector pointing outward from it. If a plane divides space into two parts, front and back, where the front space accompanies a positive normal vector and the back space has a negative normal vector, one can identify if a particle, represented by a position vector, is located in front of, or behind a plane, see Figure 2.5.

Let $\mathbf{r}$ be a position vector of a particle in space, $\mathbf{p}$ be a point on a plane with a normal vector $\hat{\mathbf{n}}$. Let $\mathbf{q}$ be a vector from $\mathbf{r}$ to $\mathbf{p}$. If $\mathbf{q} \cdot \hat{\mathbf{n}} \geq 0$ then $\mathbf{r}$ is behind or on the plane. If $\mathbf{q} \cdot \hat{\mathbf{n}} < 0$ then $\mathbf{r}$ is in front of the plane or on the same side as $\hat{\mathbf{n}}$ [88].

2.5.3 Coordinate Rotation and Orientation

For simplicity, let us look at a 2D case. The $\mathbf{x}_1$ and $\mathbf{x}_2$ axes are rotated by angle θ (clockwise is positive) into $\acute{\mathbf{x}}_1$ and $\acute{\mathbf{x}}_2$.

$$\acute{\mathbf{x}} = \mathbf{R}(\theta)\mathbf{x} \tag{2.6}$$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.7}$$

For example, let $\mathbf{x}_1 = \mathbf{i}$ and $\mathbf{x}_2 = \mathbf{j}$. After rotation by a counter-clockwise (negative) θ using Equation 2.6, we have $\acute{\mathbf{x}}_1 = \cos(\theta)\mathbf{i} + \sin(\theta)\mathbf{j}$ and $\acute{\mathbf{x}}_2 = -\sin(\theta)\mathbf{i} + \cos(\theta)\mathbf{j}$.

Orientation of a cubic crystal system can be represented as a three dimensional rotation of bases(principal directions). For a cubic crystal system, those bases are $\mathbf{a} = a\,[100]$, $\mathbf{b} = a\,[010]$ and $\mathbf{c} = a\,[001]$. An orientation can be created by rotating the three basis with $\mathbf{R}$.

$$\acute{\mathbf{a}} = \mathbf{R}\mathbf{a}, \acute{\mathbf{b}} = \mathbf{R}\mathbf{b}, \acute{\mathbf{c}} = \mathbf{R}\mathbf{c} \tag{2.8}$$

$$\mathbf{R} \equiv \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.9}$$

In order to satisfy the physical rotation, that is, the rotated vector must have the same length as the original, $\mathbf{R}$ must be an orthogonal matrix. The following must be true [89]:

$$\mathbf{R}^{-1} = \mathbf{R}^{\mathrm{T}} \qquad or \qquad \mathbf{R}^{\mathrm{T}}\mathbf{R} = 1 \tag{2.10}$$

In addition, $\mathbf{R}$ must also be a proper orthogonal matrix [89], i.e. $\det(\mathbf{R}) = 1$.

2.5.4 Euler Angles and Euler Parameters

Euler's rotation theorem states that an arbitrary rotation may be described by only three parameters. For example, a rotation may be defined using Euler Angles, ϕ, θ and ψ in the x-convention. This consists of a sequence of three rotations, first a ϕ rotation about $\mathbf{z}$-axis, second a θ rotation about the rotated $\mathbf{x}$-axis, and last, a ψ rotation about the twice-rotated $\mathbf{z}$-axis [89]. Figure 2.6(a) shows this operation.

$$\mathbf{R}(\phi,\theta,\psi) = \mathbf{R}_1(\psi)\mathbf{R}_2(\theta)\mathbf{R}_3(\phi) \tag{2.11}$$

$$\mathbf{R}_3(\phi) \equiv \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$\mathbf{R}_2(\theta) \equiv \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.13}$$

$$\mathbf{R}_1(\psi) \equiv \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

Each component of $\mathbf{R}$ will become [89];

$$r_{11} = \cos(\psi)\cos(\phi) - \cos(\theta)\sin(\phi)\sin(\psi) \tag{2.15}$$

$$r_{12} = \cos(\psi)\sin(\phi) + \cos(\theta)\cos(\phi)\sin(\psi) \tag{2.16}$$

$$r_{13} = \sin(\psi)\sin(\theta) \tag{2.17}$$

$$r_{21} = -\sin(\psi)\cos(\phi) - \cos(\theta)\sin(\phi)\cos(\psi) \tag{2.18}$$

$$r_{22} = -\sin(\psi)\sin(\phi) + \cos(\theta)\cos(\phi)\cos(\psi) \tag{2.19}$$

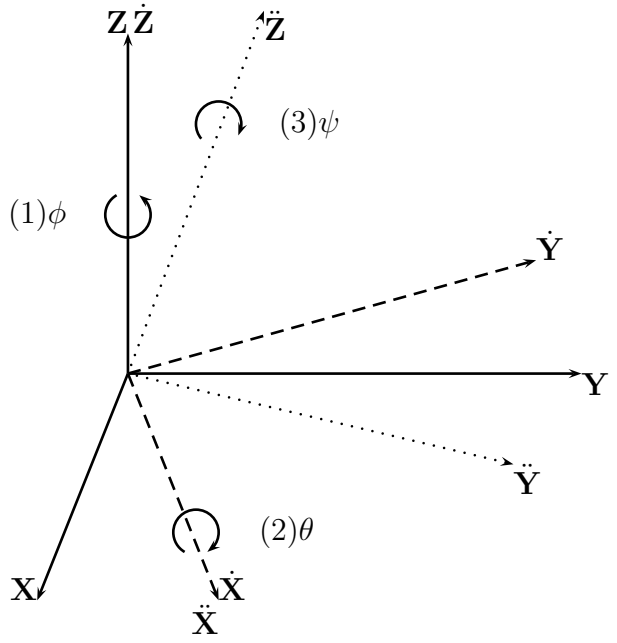
$$r_{23} = \cos(\psi)\sin(\theta) \tag{2.20}$$

$$r_{31} = \sin(\theta)\sin(\phi) \tag{2.21}$$

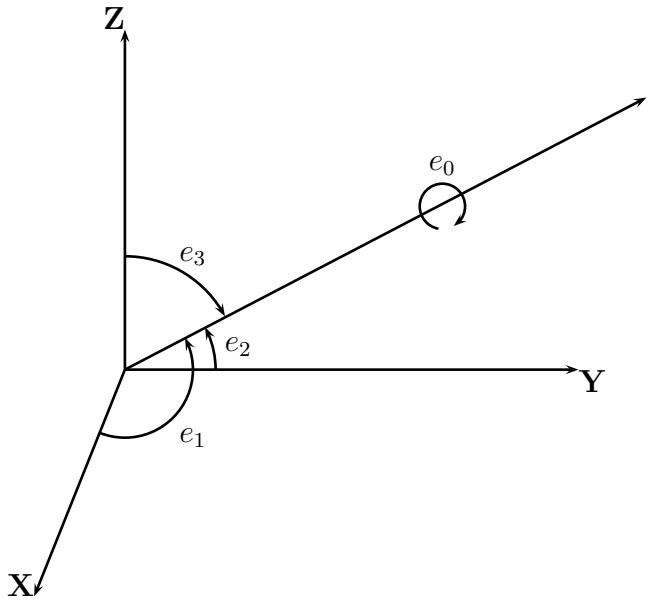
$$r_{32} = -\sin(\theta)\cos(\phi) \tag{2.22}$$

$$r_{33} = \cos(\theta) \tag{2.23}$$

Although a set of three parameters are needed to describe a rotation, there can be problems involving rotational symmetry. The other drawback for using Euler Angles to calculate a rotation matrix is that it involves a large number



(a) Euler angles



(b) Euler parameters

Figure 2.6: Euler rotation using (a) Euler angles and (b) Euler parameters

of trigonometric functions. Numerical computations usually prefer using Euler Parameters, which define a rotation matrix by four parameters, e_0, e_1, e_2 and e_3. This is equivalent to a finite rotation about an arbitrary axis, described by three components, see Fig. 2.6(b). The Euler parameters are defined as [89]

$$e_0 \equiv \cos\left(\frac{\phi}{2}\right) \tag{2.24}$$

$$\mathbf{e} \equiv \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \hat{\mathbf{n}} \sin\left(\frac{\phi}{2}\right). \tag{2.25}$$

Since the four parameters describe a rotation matrix that can also be defined using only three parameters, there must be a relation between them [89].

$$e_0^2 + \mathbf{e} \cdot \mathbf{e} = e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1 \tag{2.26}$$

Each component of $\mathbf{R}$ can be calculated using Euler Parameters by [89];

$$r_{11} = e_0^2 + e_1^2 - e_2^2 - e_3^2 \tag{2.27}$$

$$r_{12} = 2(e_1 e_2 + e_0 e_3) \tag{2.28}$$

$$r_{13} = 2(e_1 e_3 - e_0 e_2) \tag{2.29}$$

$$r_{21} = 2(e_1 e_2 - e_0 e_3) \tag{2.30}$$

$$r_{22} = e_0^2 - e_1^2 + e_2^2 - e_3^2 \tag{2.31}$$

$$r_{23} = 2(e_2 e_3 + e_0 e_1) \tag{2.32}$$

$$r_{31} = 2(e_1 e_3 + e_0 e_2) \tag{2.33}$$

$$r_{32} = 2(e_2 e_3 - e_0 e_1) \tag{2.34}$$

$$r_{33} = e_0^2 - e_1^2 - e_2^2 + e_3^2 \tag{2.35}$$

In an attempt to find a better method to generate a large number of random rotations, in this investigation, 1000 random rotation matrices were made and operated on a basis with $\mathbf{x}_1 = [100]$, $\mathbf{x}_2 = [010]$ and $\mathbf{x}_3 = [001]$.

- Euler Angles, a random selection of each of the three angles within the following limits;

$$\phi \in [0, 2\pi] \tag{2.36}$$
$$\theta \in [0, \pi] \tag{2.37}$$
$$\psi \in [0, 2\pi] \tag{2.38}$$

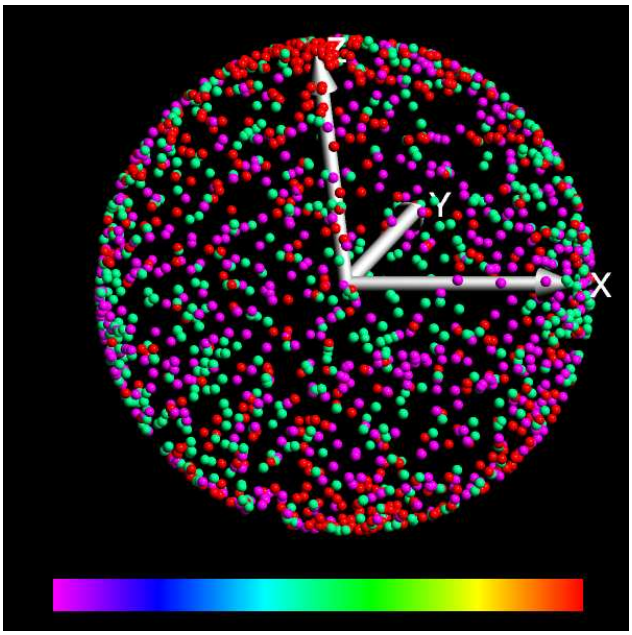
- Euler Parameters, random selection of each of the four parameters with the following rules;

$$e_0 \in [-1, 1] \tag{2.39}$$
$$\omega = \cos^{-1}(e_0) \tag{2.40}$$
$$a \in [-1, 1] \tag{2.41}$$
$$b \in [-1, 1] \tag{2.42}$$
$$c \in [-1, 1] \tag{2.43}$$
$$\mathbf{u} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k} \tag{2.44}$$
$$\hat{\mathbf{n}} = \frac{\mathbf{u}}{|\mathbf{u}|} \tag{2.45}$$
$$\mathbf{e} \equiv \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \hat{\mathbf{n}} \sin(\omega) \tag{2.46}$$

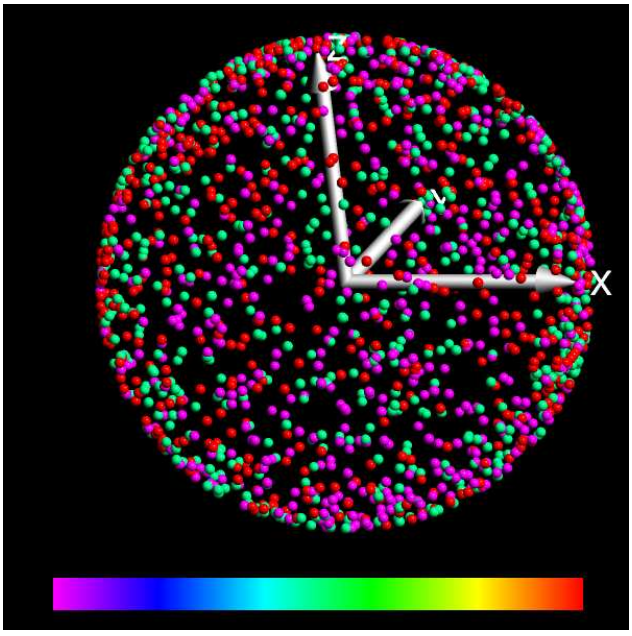
The result of the test is shown in Fig. 2.7. It can be seen that the distribution of rotated bases behaves more homogeneously in case of Euler parameters than Euler angles. The balls in purple, green and red represent the operated **x**, **y** and **z** axes. In case of Euler angles, the **z**-axes stack up near the original **z**-axis.

2.5.5 Search and delete algorithm

With the C language's dynamic memory allocation, one can quickly design a search algorithm that can work efficiently for the construction of polycrystalline



(a) Euler angles operation



(b) Euler parameters operation

Figure 2.7: Distribution of rotated bases by Euler angles and Euler parameters.

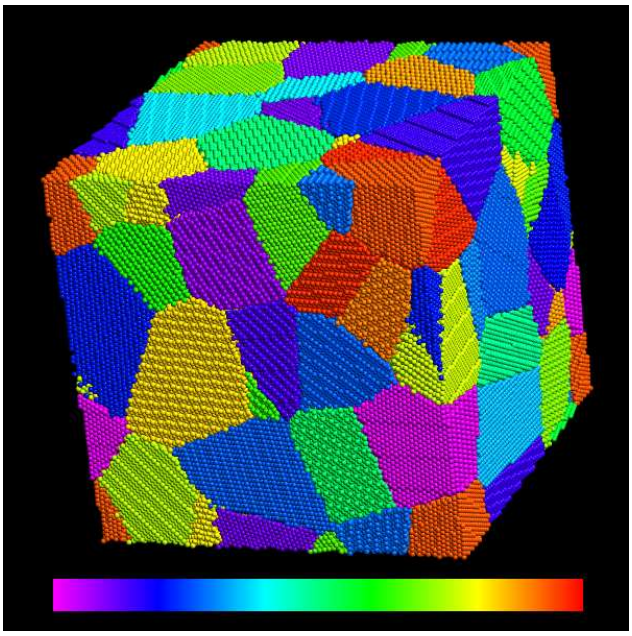


Figure 2.8: Generated polycrystalline structure with average grain size of 5nm.

samples, for example, Figure 2.8. Search boxes can be generated, dividing the domain equally in three dimensions. Each box will contain a number of particles and their information. While looping through the particles within a box, the distance between each arbitrary pair of particles can be calculated and checked for violation of a minimum particle distance which can be prescribed in advance. The particles that are too close to each other can be deleted. Looping and checking members of adjacent boxes takes the similar approach. This idea was suggested by Prof. Benson [88] and resulted in a very fast and effective algorithm that can make sure any two particles will not stay at the same location.

2.6 Periodic Boundary Conditions

It is preferable to apply periodic boundary condition in molecular dynamics simulation as it can vastly reduce the size of simulation domain and number of particles. This can, in turn, enable us to study the molecular behavior almost as if

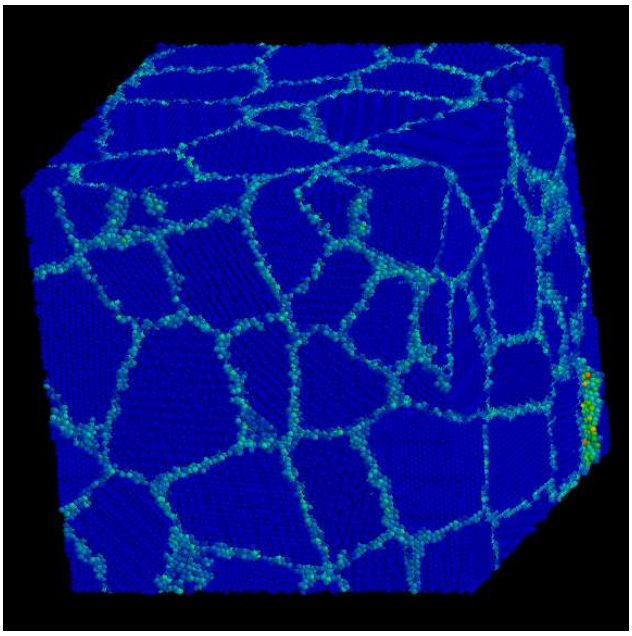


Figure 2.9: Relaxed polycrystalline structure colored by centrosymmetry parameter.

we had an infinite simulation domain free of any boundary effect. This boundary condition connects one end of the simulation box to the opposite end For example, any particles that come out the maximum x boundary will go through to the minimum x boundary. Figure 2.9 is an example of a periodic boundary condition, where grains are connected through the boundaries.

Using periodic boundary condition can also have a drawback. Once we use periodic boundary condition, it is as if we are working with a 3D array of the same simulation box stretching to infinity. Any dislocation mechanisms that happen in the simulation box will be copied to the other 26 imaginary boxes directly around it and it will feel the presence of the periodic images. This is an important limitation of this boundary condition. If we design our simulation to study molecular phenomena which are periodic, then the periodic boundary condition is flawless. Note; However, the simulation is still limited to the wavelength within the size of the simulation domain.

2.7 Centrosymmetry Parameter

For a material with cubic symmetry, each atom has pairs of equal and opposite bonds to its nearest neighbours. As the material undergoes homogeneous elastic deformation, these bonds will change direction and/or length, but they will remain equal and opposite. The equal and opposite relations for all the nearest pairs do not hold when a defect (such as dislocation or stacking fault) is introduced nearby.

Kelchner et al. [90] introduced the centrosymmetry parameter which is defined as:

$$P = \sum_{i=1,6} |\mathbf{R}_i + \mathbf{R}_{i+6}|^2 \tag{2.47}$$

where $\mathbf{R}_i$ and $\mathbf{R}_{i+6}$ are the vectors or bonds corresponding to the six pairs of opposite nearest neighbours in the FCC lattice, see Figure 2.10. The coordination number (i.e., number of closest neighbors) for the FCC structure is 12, and therefore there are 6 pairs of 2 nearest neighbor atoms. This parameter will be zero for the centrosymmetric material at a perfect crystal configuration, or under homogeneous elastic deformation, and the parameter will be nonzero for any plastic deformation of the material. The value will grow as the particle dislocates off from being in perfect crystal. The free surface within a void has the largest value of centrosymmetry parameter. The centrosymmetry parameter is useful for filtering out of all atoms with perfect crystal structure.

2.8 Schmid Factor

A characteristic shear stress is required for slip. Consider the crystal illustrated in Figure 2.11 which is being deformed in tension by an applied force F along the axis of the cylindrical crystal. If the cross-sectional area is A the tensile stress parallel to F is $\sigma = F/A$. The force has a component $F\cos\lambda$ in the slip direction, where λ is the angle between F and the slip direction. The force acts over the slip

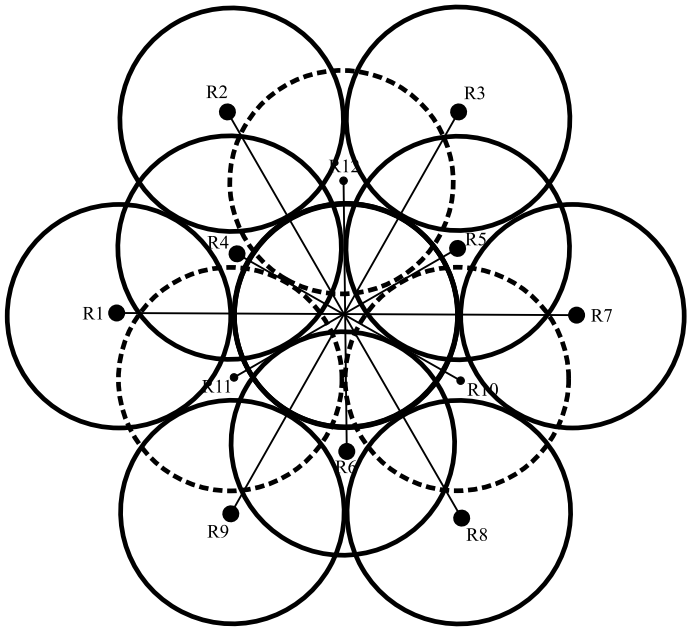


Figure 2.10: 12 nearest neighbor atoms surrounding a center atom, with the 3 dashed atoms belong in plane A, the 7 atoms belong in plane B and the last 3 atoms belong in plane C, in the FCC plane ABC configuration.

surface which has an area $A/\cos\phi$, where ϕ is the angle between F and the normal to the slip plane. Thus the shear stress, τ, resolved on the slip plane in the slip direction is

$$\tau = \frac{F}{A}\cos\phi\cos\lambda \tag{2.48}$$

The quantity $\cos\phi\cos\lambda$ is known as the "Schmid Factor" [91]. When the loading direction is known as $\vec{\mathbf{a}}$ and the slip direction and slip plane are $\vec{\mathbf{b}}$ and $\vec{\mathbf{c}}$, respectively. The Schmid factor can be calculated as

$$\text{S.F.} = \cos\phi\cos\lambda = \frac{\vec{\mathbf{a}}\cdot\vec{\mathbf{c}}}{|\mathbf{a}||\mathbf{c}|}\frac{\vec{\mathbf{a}}\cdot\vec{\mathbf{b}}}{|\mathbf{a}||\mathbf{b}|} \tag{2.49}$$

This calculation is used in the table of Schmid factor in section 3.8.2.

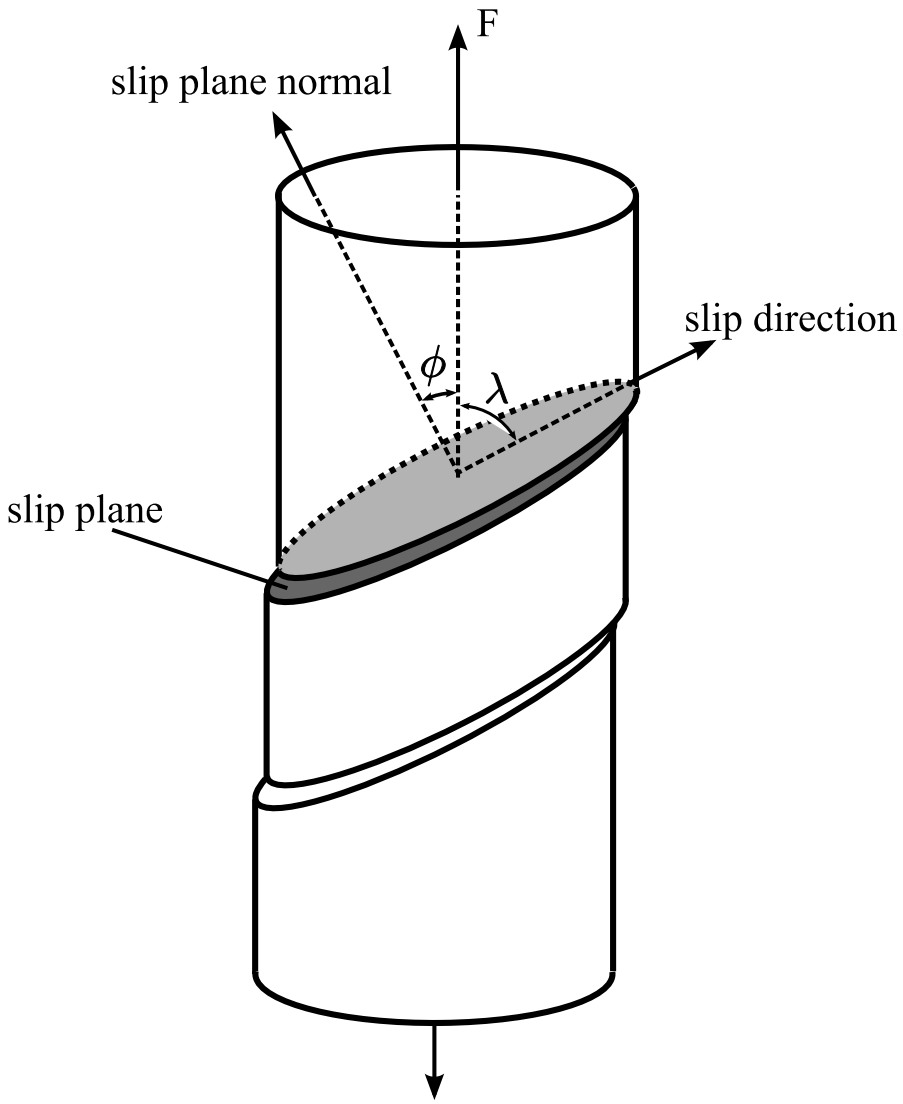


Figure 2.11: Illustration of the geometry of slip in crystalline material. Note that $(\phi + \lambda) \neq 90^\circ$ in general [91].

3

Results and Discussion

3.1 Void Growth in Single-crystal - Hydrostatic expansion

3.1.1 Simulation Setup

A cubic domain of single crystal copper with 108,000 atoms was generated using LAMMPS. The crystal orientation was aligned in order that all three principal directions of the crystal align with the global coordinate system. The domain has dimension of 10.845 by 10.845 by 10.845 nanometers which is equal to 30 unit cells by 30 unit cells by 30 unit cells of size $a_0 = 3.615$ Å or 0.3615 nm. The boundary conditions were set to be periodic. After the equilibration process (letting the simulation without loading run for a certain timesteps), a spherical void was cut at the cube center with radius of 1.446 nanometers, removing 1,061 particles. A second process of equilibration, or relaxation, was done to reduce the energy of the system. Once we have the domain with the proper temperature (T = 300K) and pressure ($\sim$ 1 atm), we started applying hydrostatic expansion by rescaling the volume of the domain in all three directions. See Figure 3.1

The first simulated cubic domain was rescaled to 12.9939 by 12.9939 by 12.9939 nanometers, which is equal to a 72% volume gain. The timestep used in LAMMPS

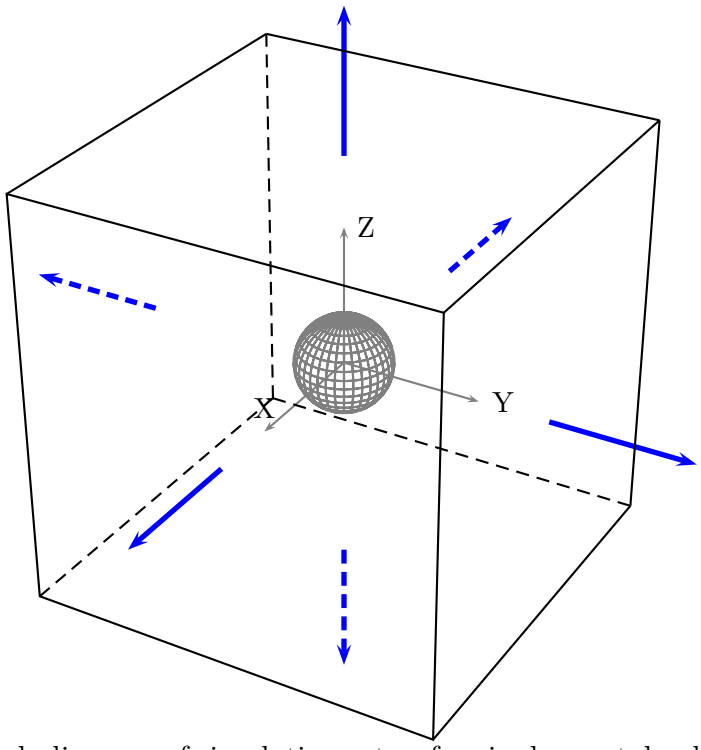


Figure 3.1: Sketch diagram of simulation setup for single crystal cubic domain and hydrostatic strain

was set to 1 femtosecond, and the number of integration steps would determine how fast we were expanding the domain. For instance, in the first simulation we used 2500 integration steps which give a strain rate of 2.88×10^{11} s^{-1}.

For the second simulation, the domain volume was rescaled to 11.84 by 11.84 by 11.84 nanometers, which results in a 30.1267% volume expansion. This was done in one million integration steps which give a lower strain rate of 3.01267×10^{8} s^{-1}. The followings sections give the results from these two simulations.

3.1.2 Results

High Strain Rate

With a high strain rate of 2.88×10^{11}, the domain expanded so fast that the transition between elastic into plastic deformation could not be determined clearly. The crystal dislocation mechanism which enables expansion of the precut void was

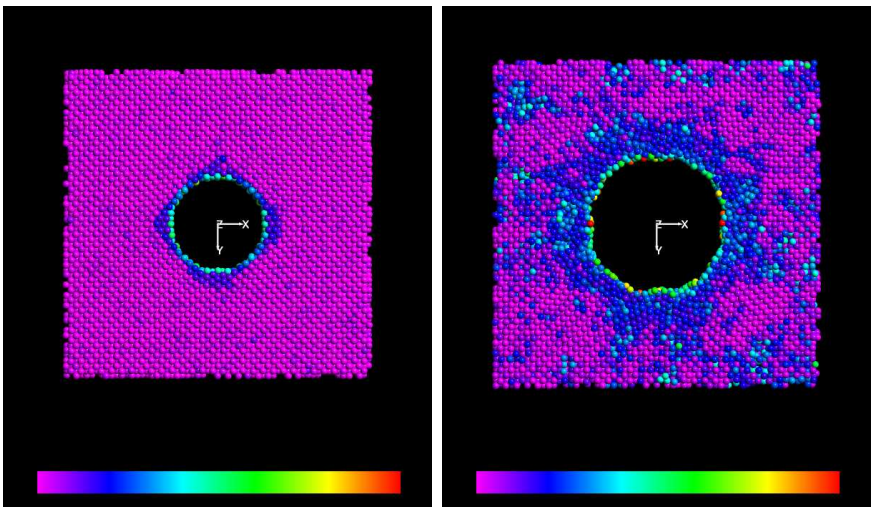
overwhelmed by several small defect spots that later developed into nucleation sites for smaller voids outside of the precut void. These defects/nucleations suppressed the dislocation mechanism to the minimum, Fig. 3.2. This result from applying very high strain rate is similar to the results by Gungor and Muroudas [92], Kuksin et al. [93, 94] and Norman and coworkers [95, 96].

Low Strain Rate

With the lower strain rate of 3.01267×10^8 s^{-1}, expansion of the domain can be observed. A suspected period of elastic deformation is followed by yield point and plastic period as dislocations develop. Dislocations start to form during the late "elastic" period and continue to grow as the domain moves into the plastic regime. As the precut void starts to grow, its shape evolves into a geometric shape facilitated by the influence of the dislocation planes which are aligned with the slip plane systems. Filtering of the atoms using the centrosymmetry parameter shows only the particles which have moved out of perfect crystal configuration, and we can see that the dislocations indeed form into an octahedron, Fig 3.3(d). We can also observe the multiplication and annihilation of dislocations that travel across the periodic boundary. The stress calculation shows that dislocations are a mechanism which relaxes internal stress while the domain is under tensile hydrostatic stress.

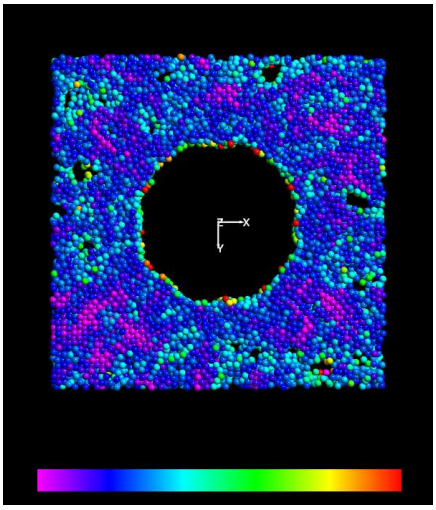
3.1.3 Interpretation

It is clear that strain rate significantly affects how dislocations can develop. Lower strain rate lets dislocations grow as they relax stress concentration from the growing void, while higher strain rates forced the void and the entire domain to grow in a very short time, rendering the dislocation mechanisms ineffective. We could not observe dislocation evolution in simulations with the applied strain rate in the order of 10^{11} s^{-1} or greater. It should be noted here that the above simulations used the copper EAM potential from the distribution package of LAMMPS(2005) and may differ from results using Mishin's copper potential,



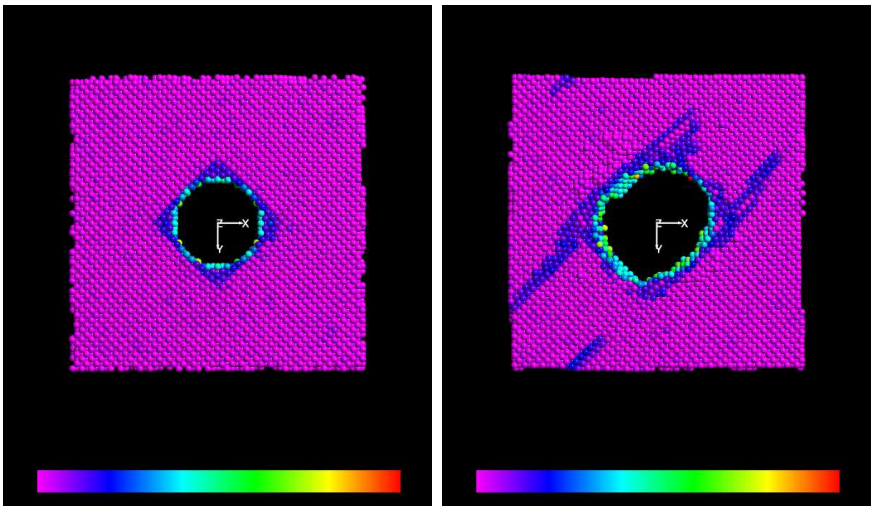
(a) Prismatic dislocation (0.9 ps)

(b) Growing defects (1.7 ps)



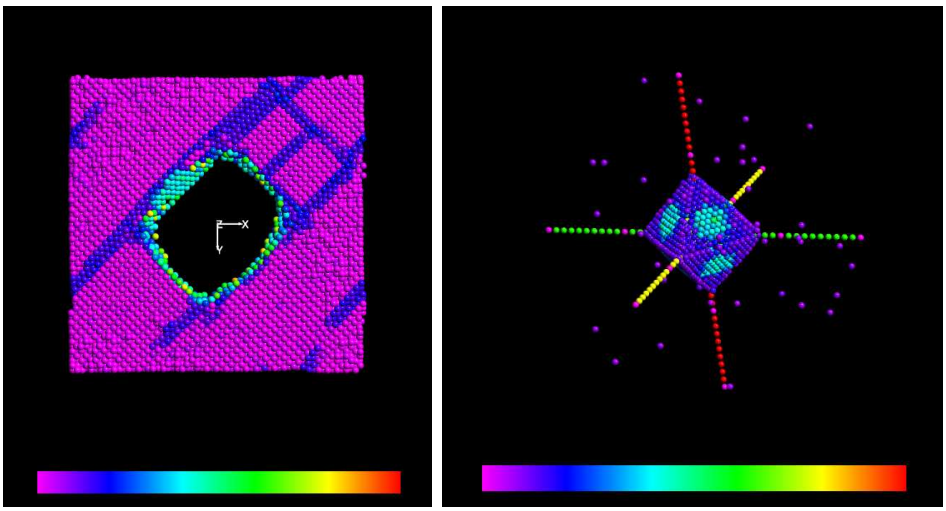
(c) Void nucleation (2.0 ps)

Figure 3.2: Growth of voids from very high strain rate(2.88×10^{11}) under hydrostatic expansion. The thin slab enables us to see defect atoms within the simulation box. The color bar (purple, blue, cyan, green, yellow and red) represents the values of centrosymmetry parameter from low to high.



(a) Dislocations emanating from void (340 ps)

(b) Expanding dislocation loops (380 ps)



(c) Dislocations traveling trough boundary (400 ps)

(d) Octahedral shape of defect atoms surrounding void surface in 3D view (the lines made of connecting atoms represent the three axes (340 ps))

Figure 3.3: Growth of void from 3.01267×10^8 s^{-1} strain rate: dislocation generation and motion, and shape changing of void under hydrostatic expansion (a,b,c are slabs of small Δz which is perpendicular to [001].

that were used in most of this work.

3.2 Void Growth in Bicrystal Structure

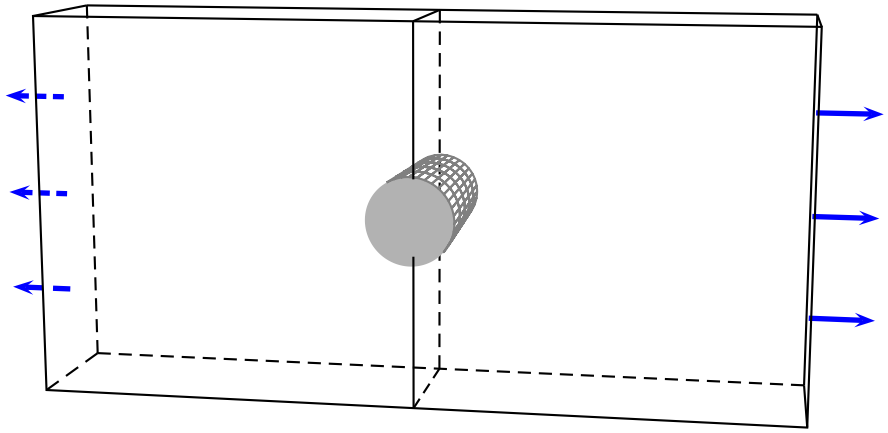
3.2.1 Simulation Setup

Simulation modeling is divided into two parts:

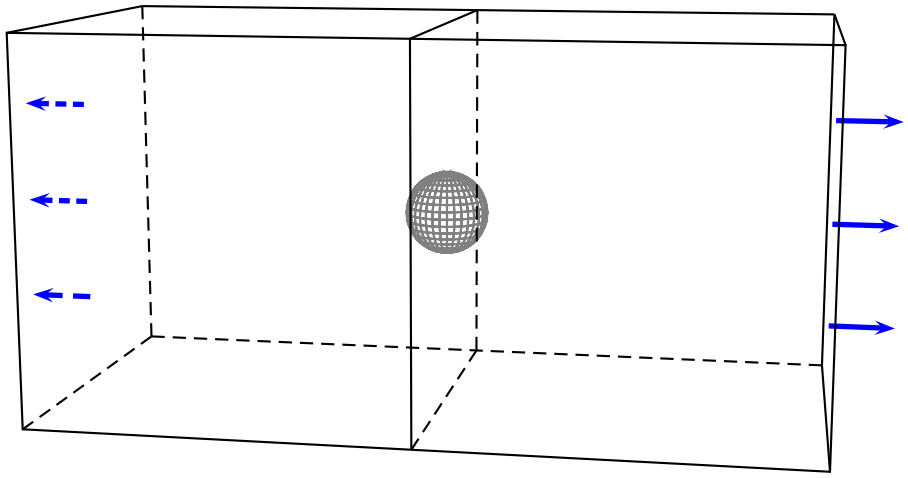
- a thin domain with dimensions of $72.3 \times 180.75 \times 361.5$ nm (comparable to $20 \times 50 \times 100$ unit cells) with a cylindrical void with radius of 1.8 nm (Figure 3.4(a));
- a thick domain with dimension of $180.75 \times 180.75 \times 361.5$ nm (comparable to $50 \times 50 \times 100$ unit cells) with a spherical void with radius of 1.8 nm (Figure 3.4(b)).

For the two setups, the domain had the same size in length and height with differences in thickness and internal void geometry. The domains were divided into two halves lengthwise. The right half was composed of a perfect crystal FCC copper with a 21.8014° clockwise rotation while the left half had the same perfect crystal but with 21.8014° counter-clockwise rotation. The total number of particles (atoms) for the cylindrical void case was 400,720 and for the spherical void case was 1,001,800.

After the creation of the domains, they underwent a session of equilibration to readjust the boundary between the two halves. This was done with constant temperature at 300K and pressure control within a range of 1 atm. Once the equilibration (running the simulation without applying a load) was run for a few thousand timesteps, cylindrical and spherical voids with radius of 1.8 nm was cut with their respective domains where the origins of either cylinder or sphere voids sit at the center of boundary of the two crystals. This was followed by another session of relaxation with same temperature and pressure control for a few more thousand timesteps to relax the internal surface of the voids.



(a) Cylindrical void



(b) Sphereical void

Figure 3.4: Diagram of simulation setup for bicrystal domain with (a) cylindrical void and (b) spherical void.

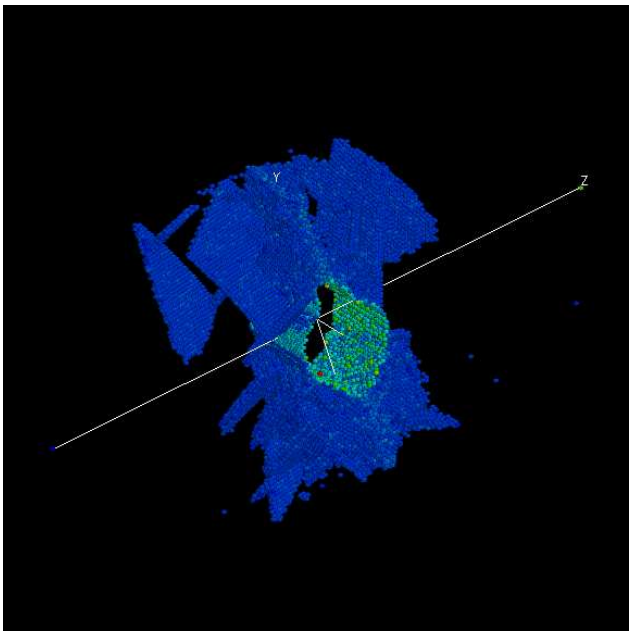


Figure 3.5: Dislocations emanating from cylindrical void (loading axis z)

Once the relaxation was done, the domain underwent a uniaxial extension with the rescaling of the volume in the length direction. The timestep was 1 femtosecond and it was done for 60,000 time steps for 30.354% volumetric expansion, giving 5.05902×10^9 s^{-1} in strain rate.

3.2.2 Results

Cylindrical Void

During the expansion of the domain, the transition from the elastic region through the yield point into the plastic region can be observed. The void grew along the boundary both in the upper and downward directions. Once the domain yielded, there is a sudden jump in the stress as the grain boundary failed to hold the two crystals together. Dislocations play a very important role in relaxing the internal stress by allowing the void to grow. As we have a thin domain with a cylindrical void, the dislocations come out from a relatively large void surface,

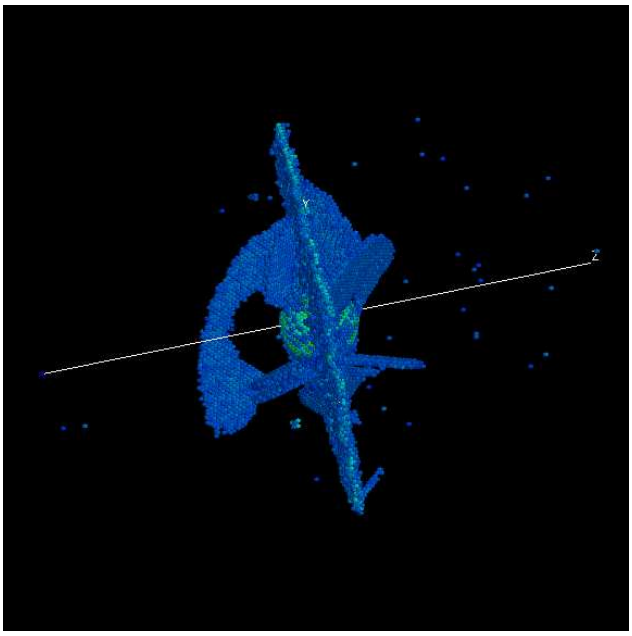


Figure 3.6: Dislocation loops emanating from spherical void (loading axis z)

growing toward the periodic boundary and interacting with their mirror images rather quickly, see Figure 3.5. This gives a large dislocation density in a time of few picoseconds.

Spherical Void

The result of uniaxial expansion in the case of a spherical void was similar to one with a cylindrical void. Transition from elastic into plastic behavior was observed. Dislocation propagation also showed similarities. What was significantly different from the cylindrical void case is how the dislocation loops spread outward radially instead of going through side boundaries and coming back the other side of our periodic boundary. Dislocations had more time to grow and relax the stress concentration from the growing void, see Figure 3.6.

The bicrystal uniaxial expansion run was simulated again later with the Cu potential from Mishin [83]. This potential gives a correct stacking fault energy, and the dislocation bands became narrower than in the previous results shown in

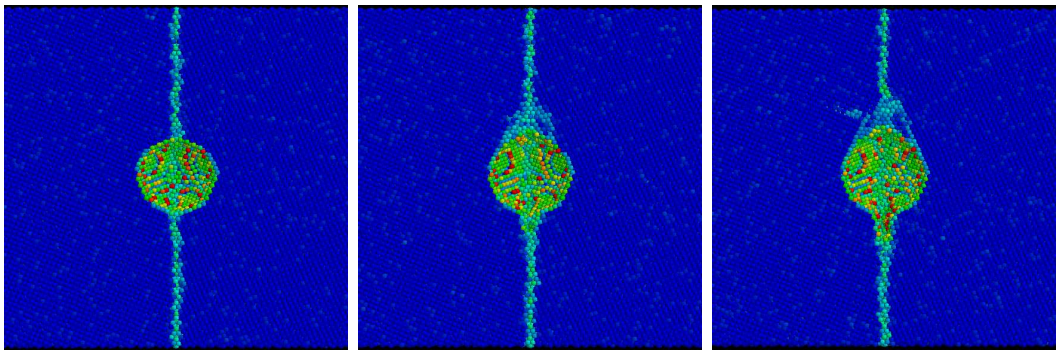
Figure 3.5 and 3.6 done with the Cu potential that came with LAMMPS. The higher stacking fault energy directly influences the distance of partial dislocations separation that composes a band of dislocations [97]. The interaction between dislocations also shows changed behavior as they become biplanar similar to interaction of dislocations in uniaxial expansion of single crystal with void(section 3.4). The configurations using the Mishin et al. [83] potential is presented in Figures 3.7 and 3.8. The stacking fault ribbons are narrower than in Figure 3.6. The loops generated are indeed shear loops that are analyzed in section 3.4.

3.2.3 Interpretation

When periodic boundary conditions are used, the simulation design of cylindrical void and a thin domain was unacceptable in terms of dislocation interactions. It tempting to think that we can represent dislocation activity with a simulation box similar to a 2D model used in finite element analysis. This is incorrect, as dislocation activity is indeed a 3D behavior. The best way to see dislocation propagation is to allow enough space and time for them to grow, while trying to limit the number of dislocation sources to the minimum. This produces a small number of dislocations that travel some distance before interacting with other dislocations.

3.2.4 Dislocation Activity

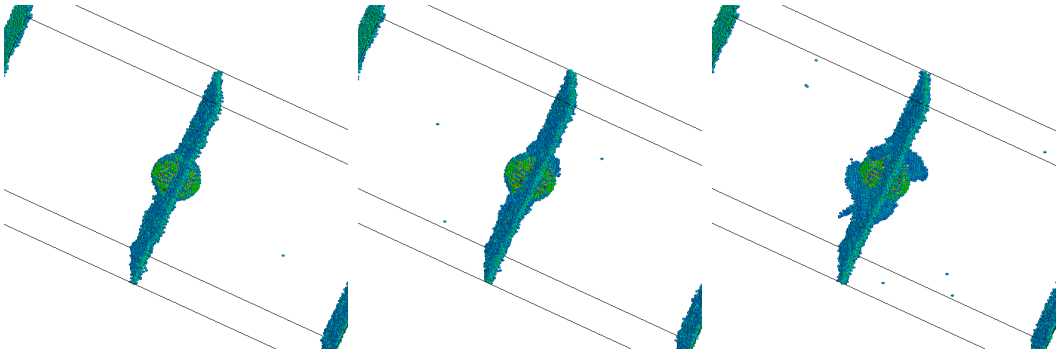
The bicrystal simulation, although done at much higher strain rate (5×10^9 s^{-1}), shows results consistent with single crystal void growth. Partial dislocation loops are emitted from the void surface, interact, and travel together as the strain increases, Fig. 3.9. One can see at least two biplanar shear loops emanating from the void in Fig. 3.9 (marked 1 and 2), one in each grain. The dislocation interactions are more complex because the number of slip planes involved is twice as high. Figures 3.7 and 3.8 show the two-dimensional sections perpendicular to the grain boundary (left) and three-dimensional views (right) at different times (40 to 45ps). The evolution of shear is evident, and the similarity with the experimental



(a) 40 picoseconds side view

(b) 42 picoseconds side view

(c) 43 picoseconds side view

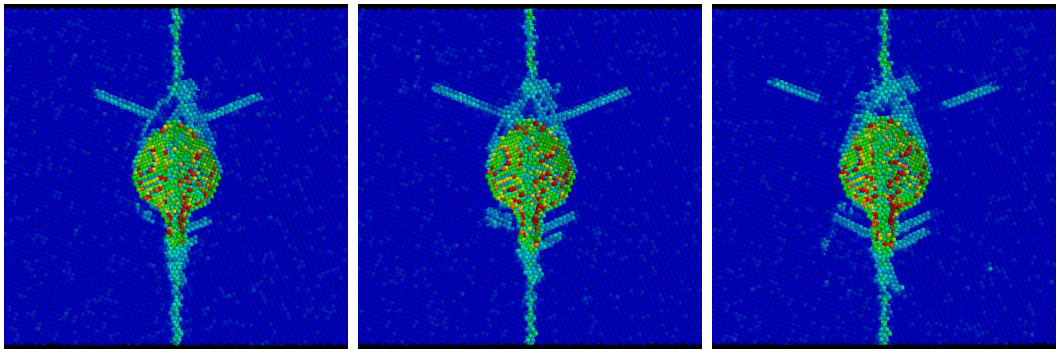


(d) 40 picoseconds perspective view

(e) 42 picoseconds perspective view

(f) 43 picoseconds perspective view

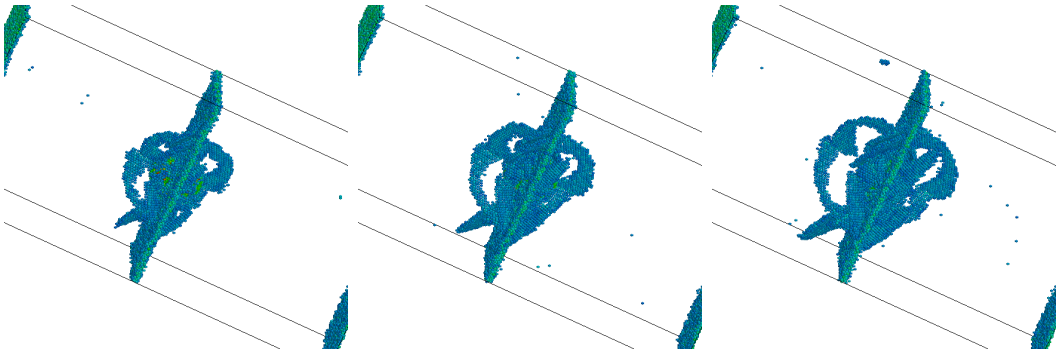
Figure 3.7: Sequence of loop nucleation and growth in the bicrystal simulation.



(a) 44 picoseconds side view

(b) 44.5 picoseconds side view

(c) 45 picoseconds side view



(d) 44 picoseconds perspective view

(e) 44.5 picoseconds perspective view

(f) 45 picoseconds perspective view

Figure 3.8: Sequence of loop nucleation and growth in the bicrystal simulation(continued).

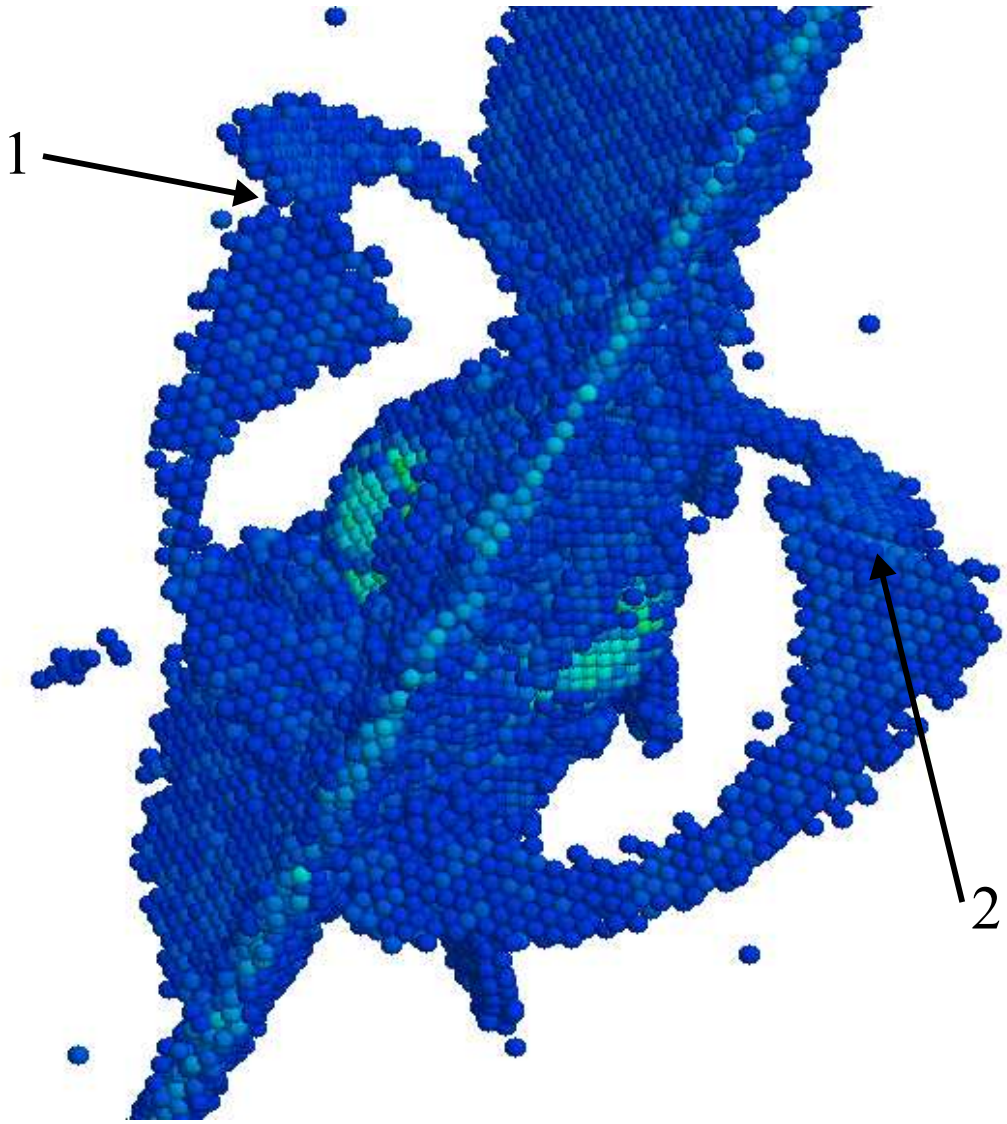


Figure 3.9: Shear loops and their interaction in bicrystal simulation with initial void at grain-boundary (uniaxial strain).

Table 3.1: Table of different potentials used for copper.

Potential	Reference	$\gamma_{SF}(mJ/m^2)$	$\gamma_{USF}(mJ/m^2)$
Cleri-Rosato	[98]	20.6	154.1
Schiøtz-Jacobsen	[99]	33.5	173.1
Mishin et al. 1	[83]	44.4	158
Mishin et al. 2	[83]	36.2	161

observations of Figure 1.3 is striking. Slip emanates from the void starting in Fig. 3.7(c) and propagates outward along $\langle 111 \rangle$; shear loops are activated in the two grains shown in Fig. 3.8(a-c). In Fig. 3.8(c), one can see that the trailing partial follows the leading partial. This can be seen better in the tridimensional views shown in Figs. 3.8(d) and 3.8(f).

The separation of partials in MD calculations has been the object of considerable study, and the potential used influences these values. Van Swygenhoven et al. [97] discuss this for nanocrystalline metals and point out the importance of two stacking fault energies: the stable and the unstable one. The generalized planar fault energy curve provides the barrier that the leading and trailing partial dislocations encounter. This barrier has two cusps with a trough between them. The first cusp corresponds to the unsteady SFE, and the second cusp to the steady SFE. Van Swygenhoven et al. [97] warn the readers of the limitations of the MD analysis, where both the high stresses and short timescales can affect the separation between partials. Table 3.1 shows the stable and unstable stacking fault energies for different potentials used. There is some variation. The nucleation of the trailing dislocation encounters the energetic barrier ($\gamma_{USF} - \gamma_{SF}$). Hence, we are aware of the limitations of MD in predicting the actual partial separation. Nevertheless, we observe both partials and perfect dislocations and the minimum partial separation observed (Fig. 3.9) is $(3-5)b$. However, the stacking fault can be considerably larger prior to the nucleation of the trailing partial.

The simulations were carried out under uniaxial strain, not uniaxial stress, as in Potirniche et al. [61]. The lateral stresses have been computed as a function

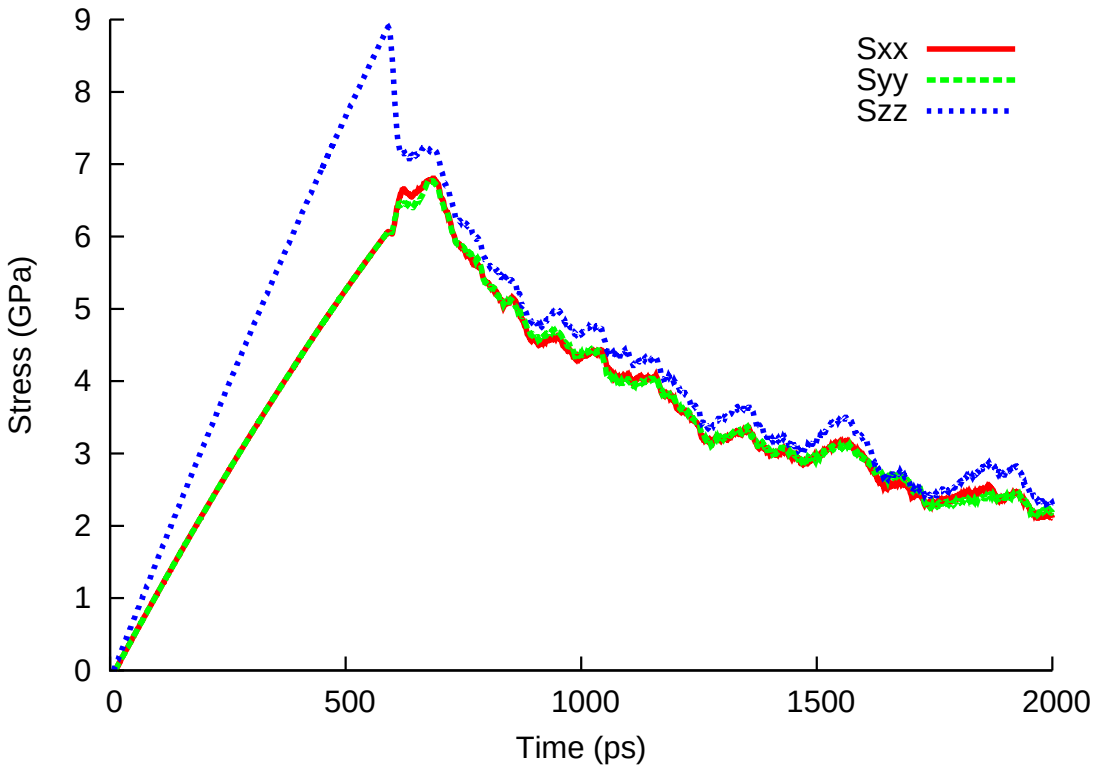


Figure 3.10: Lateral stresses generated when loading is applied in direction ZZ.

of time and are shown in Figure 3.10. The difference between longitudinal (S_{zz}) and lateral stresses (S_{xx} and S_{yy}) rises linearly until the maximum. At this point, the longitudinal stress decreases as a result of dislocation loop nucleation at the surface of void. There is a modest rise in the transverse stresses. As a result, the three stresses approach each other and the state of stress approaches a hydrostat. As the three stresses decay on unloading, their magnitudes are very close. Hence, the emission of dislocations at the void surfaces relaxes the deviatoric stresses.

3.3 Compression and Expansion of Nanocrystalline Nickel

TEM measurements observe voids in nanocrystals [97]. This simulation is motivated by the result of a measurement of voids and vacancies with in nanocrystal nickel. Positron annihilation analyses have been performed on nanocrystal nickel before and after shock loading experiment. The results showed that the amount of voids and vacancies within the nano-structure stayed unchanged. This brought up a number of questions, whether voids re-open after unloading or there were no voids but just vacancies distributed throughout the structure [100].

3.3.1 Simulation setup

A simulation box of nanocrystal nickel with an average grain size of 5 nanometers is created with the cubic domain size of $17.6 \times 17.6 \times 17.6$ nanometers. The nanocrystalline nickel structure was generated using Voronoi tessellation and was provided by Eduardo Bringa and Paul Erhart and it had a mean grain size of 5 nm. The nano structure was relaxed in LAMMPS with the following sessions [100]:

- 0.25 picoseconds of equilibrating at 0.01 Kelvin.
- 1.25 picoseconds of equilibrating at 300 Kelvin.
- 2.5 picoseconds of equilibrating at 5 Kelvin.
- 2.5 picoseconds of equilibrating with no control of temperature.

The equilibrated simulation box undergoes a uniaxial-compression up to a certain strain then it is hold at this strain for a period of time and it then followed by an unloading session. The unloading session is divided into: 1) unload to zero strain and 2) unload to zero pressure. Two shock loading levels were modeled: 23 GPa and 38 GPa shocks. This gives three different MD simulations:

- Uniaxial compression of 8% strain (P=23 GPa) with $\dot{\varepsilon} = 2 \times 10^{10}$ s^{-1}, holding for 10 picoseconds, and unloading to zero strain with the same $\dot{\varepsilon}$.
- Uniaxial compression of 13% strain (P=38 GPa) with $\dot{\varepsilon} = 3.25 \times 10^{10}$ s^{-1}, holding for 10 picoseconds, and unloading to zero strain with the same $\dot{\varepsilon}$.
- Uniaxial compression of 13% strain with $\dot{\varepsilon} = 3.25 \times 10^{10}$ s^{-1}, holding for 10 picoseconds, and unloading to zero pressure in 8 picoseconds.

The first run is comparable to the shock of 23 GPa while the second and the last run are comparable to 38 GPa shock. A nanocrystalline nickel sample with voids was given to us from Eduardo Bringa and Paul Erhart from LLNL. Voids were created at the triple junctions with a constant radius (R = 1 nm). There are four simulations for this sample with voids, two for two levels of loading and two for two types of unloading.

3.3.2 Results

At the beginning of the compressive loading the nano-structure behaves elastically for a very short period (up to 1% strain). Then it deforms plastically by mechanisms within the grain boundary, such as grain boundary sliding [99]. Up to this point there is no dislocation emission.

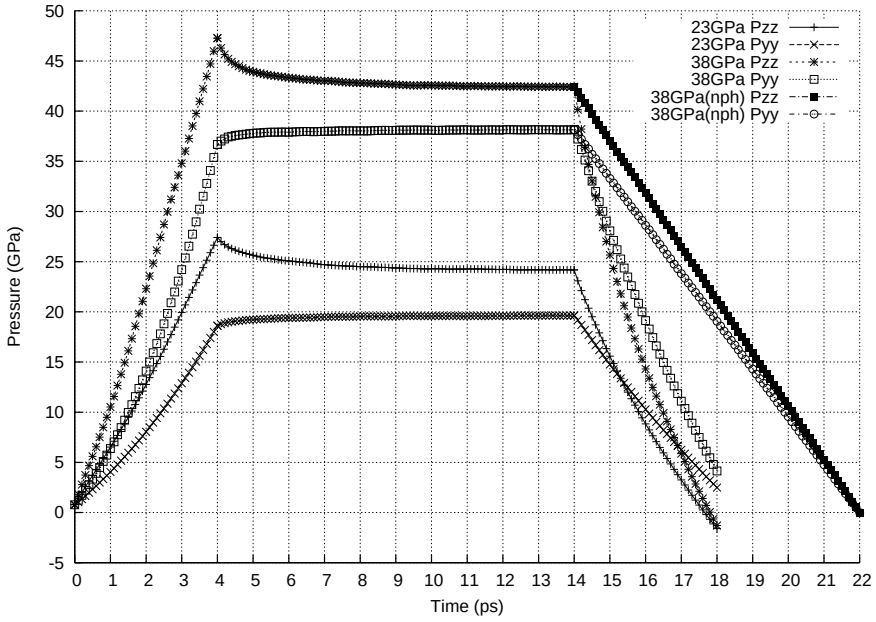


Figure 3.11: Plot of pressure against time.

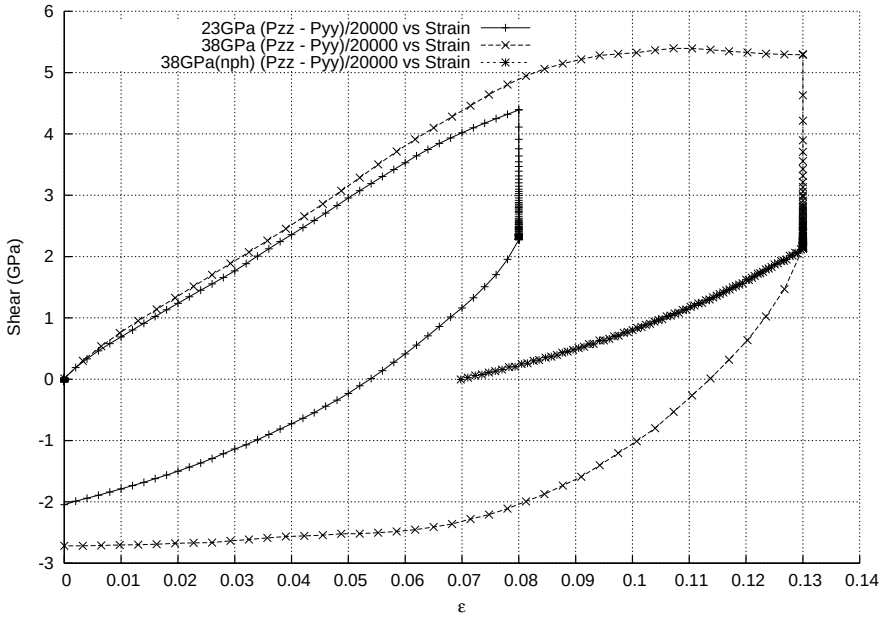
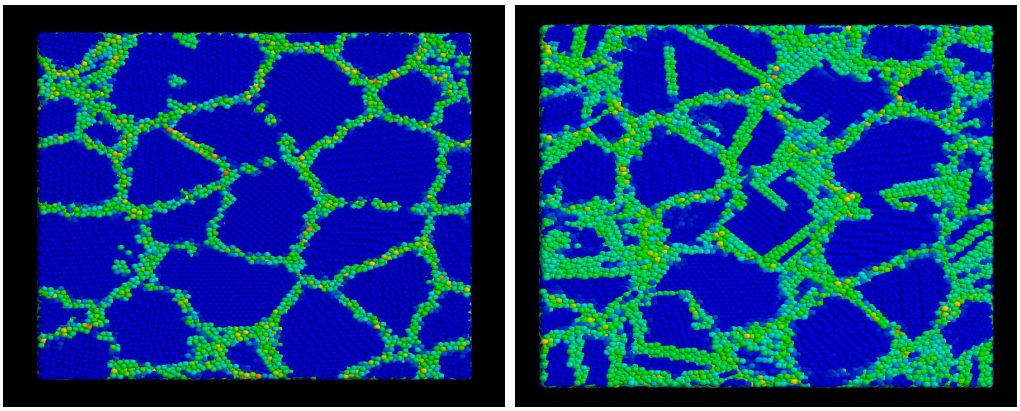
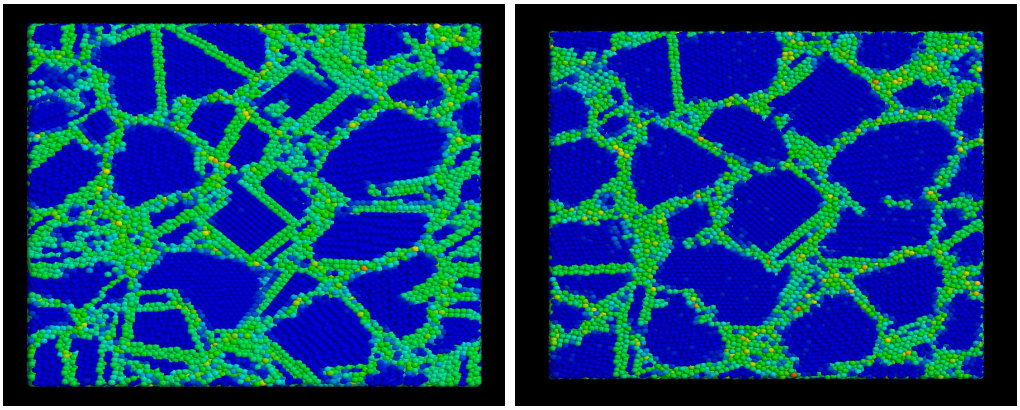


Figure 3.12: Plot of shear stress versus strain for nanocrystal Ni (d = 5 nm).



(a) Initial condition

(b) Fully loaded



(c) End of holding at constant strain

(d) Fully unloaded

Figure 3.13: Sequence of nanocrystalline nickel (d=5 nm) under 38 GPa loading. Note the disappearance of most dislocations after unloading.

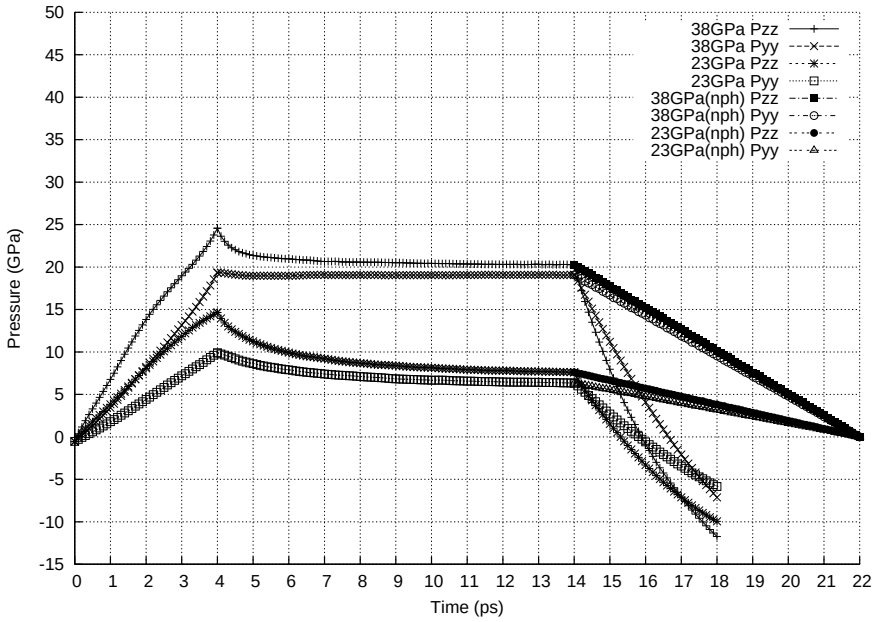


Figure 3.14: Plot of pressure against time for nanocrystal sample with voids.

Plastic deformation becomes more noticeable (after $\sim$ 7% strain) when we reach the second slope change, Fig. 3.12, is especially more pronounced in the 38 GPa run. Dislocations form and start to travel across grains at this point and continues until it has passed the target strain. Since we are holding strain constant, dislocation mechanisms now continue to work and try to equate σ_{xx} and σ_{yy} to σ_{zz} and vice versa.

Dislocations have reached a saturation point where all of possible local stresses are reduced (flat plateau before unloading in Fig. 3.11); this can be seen by the leveling of pressure in plot in Figure 3.11. The average pressure no longer changes. The amount of triaxiality becomes constant.

As soon as unloading happens dislocations pull back. For the case of unloading to zero strain, the number of dislocations was reduced very fast and only a few dislocations were left as residual deformation. For the case of unloading to zero pressure, the rate of reduction of dislocation is much slower and many more dislocations are left at the end.

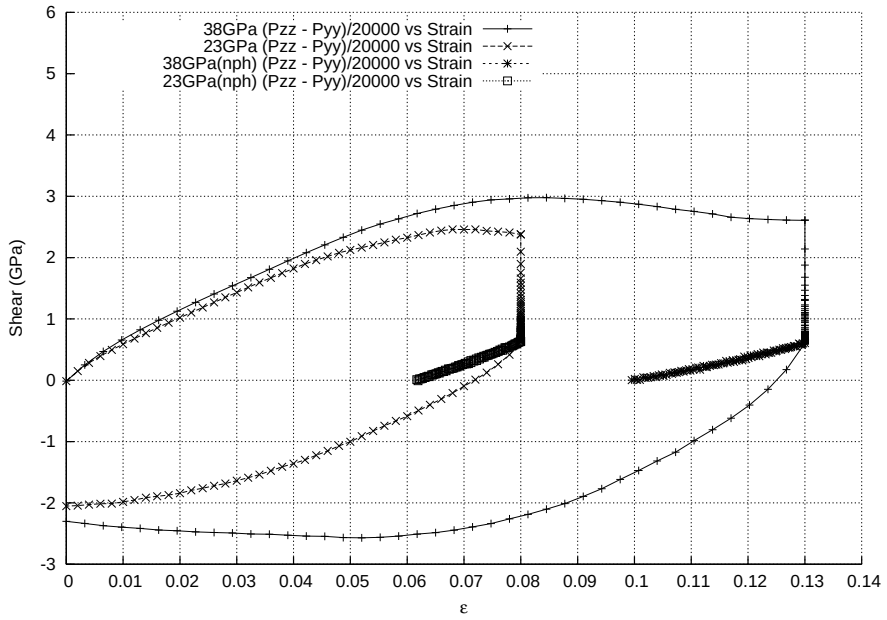
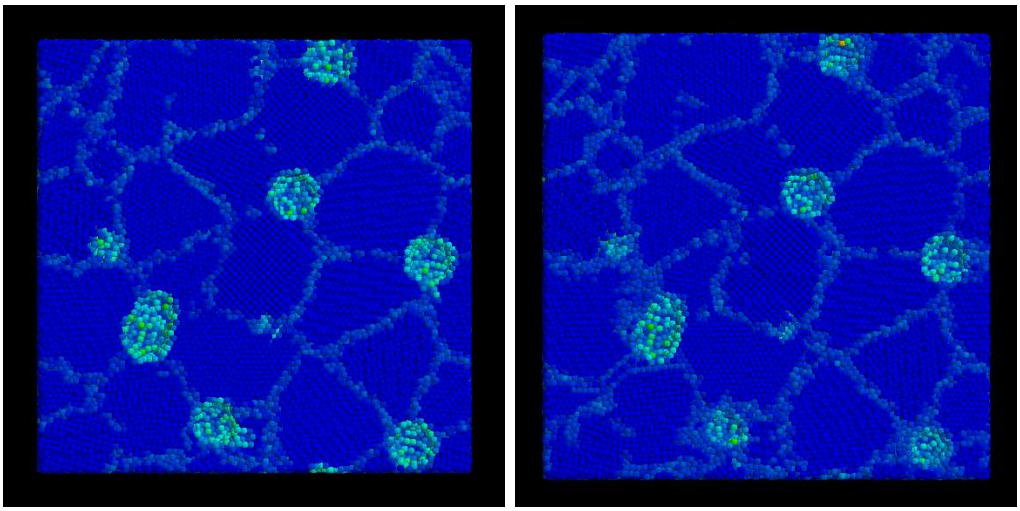


Figure 3.15: Plot of shear stress versus strain of nanocrystal sample with voids.

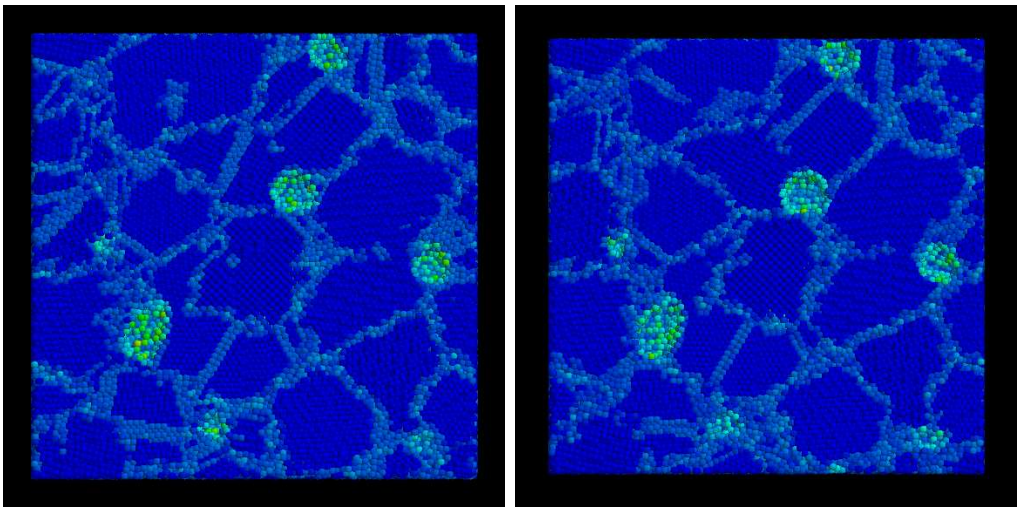
For the cases with voids, with 23 GPa loading, the voids shrank in size a little but did not fully close. This is also accompanied by grain sliding and rotation as found previously and also an additional small amount of dislocation activity. Jarmakani et al. [101] showed that most of the plastic deformation ($\sim 80\%$) in nanocrystalline nickel is accommodated by the grain boundaries. It was until the simulation had passed maximum strain that dislocations traveled further. When unloaded, some dislocations stay, but the majority of dislocations pull back, and the voids never grew larger.

With 38 GPa loading on the sample with voids, dislocations traveling from the void surface and grain boundaries as well as void shrinking can be observed. Voids collapsed completely before reaching maximum strain and extra dislocation activity followed that. When the sample was unloaded, dislocations pulled back slightly but the voids never reopened.



(a) Initial condition

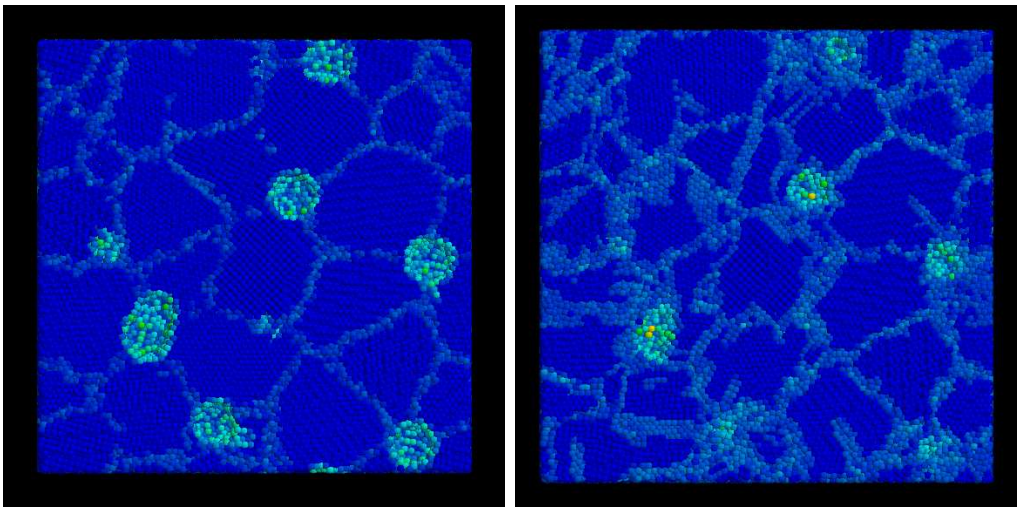
(b) Fully loaded



(c) End of holding at constant strain

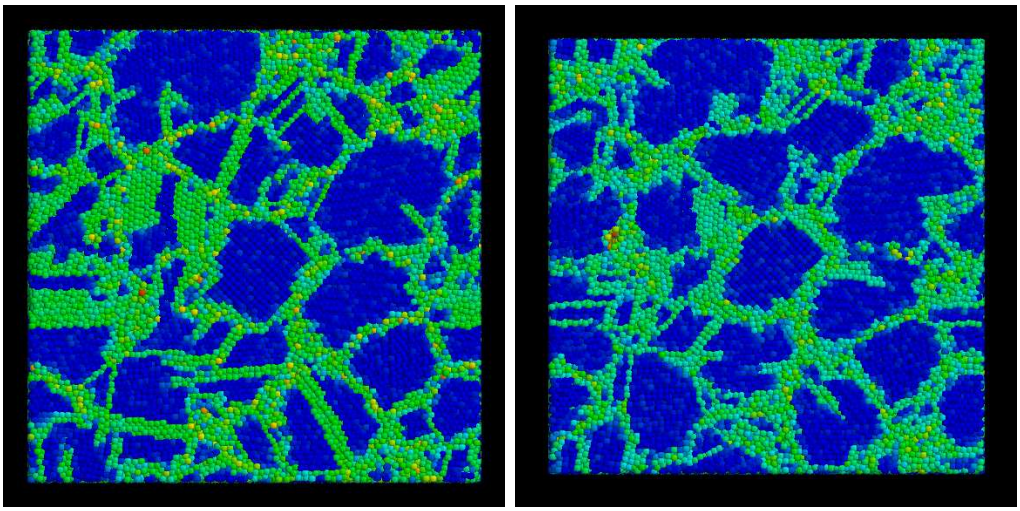
(d) Fully unloaded

Figure 3.16: Sequence of nanocrystal nickel under 23 GPa loading with sample containing voids.



(a) Initial condition

(b) Fully loaded



(c) End of holding at constant strain

(d) Fully unloaded

Figure 3.17: Sequence of nanocrystal nickel under 38 GPa loading with sample containing voids

3.3.3 Interpretation

With 23 GPa loading level, the voids did not all collapse. Rather, grain sliding dominated the plastic deformation. As the load (strain) grew larger for 38 GPa, dislocation activity played a dominant role in local stress relaxation. The voids that existed before compression never reopened after they were fully collapsed. There are several stages with different mechanisms of plastic deformation in combination during the same simulation. This will need further analysis and properly designed simulations that can separate different mechanisms during each individual step.

3.4 Void Growth in a Single-Crystal Structure - Uniaxial expansion

As we have learned that strain rates directly affect how dislocations behave, designing a new simulation become much easier. The simulations presented in this section (Figure 3.18) were designed to run at a fixed low strain rate of 10^8 s^{-1}, with a few additional runs with different strain rates and loading direction, for reference purposes. The following MD simulations are designed to mimic how finite element methods would be generally set up for a simulation with uniaxial loading conditions. The results of the simulations should therefore be easier to compare with their continuum counterparts. These molecular dynamics simulations are also motivated by the knowledge that continuum models for porous materials, mentioned in section 1.3.2, do not take into account the size scale of the void[100].

3.4.1 Simulation Setup

A cubic domain of single crystal copper was created such that all three principal directions of the crystal align with the global Cartesian coordinate system of the simulation domain. The size of each cubic domain side was 28 unit cells or 10.122 nanometers. A spherical void was cut at the center of the cube with a radius of 1

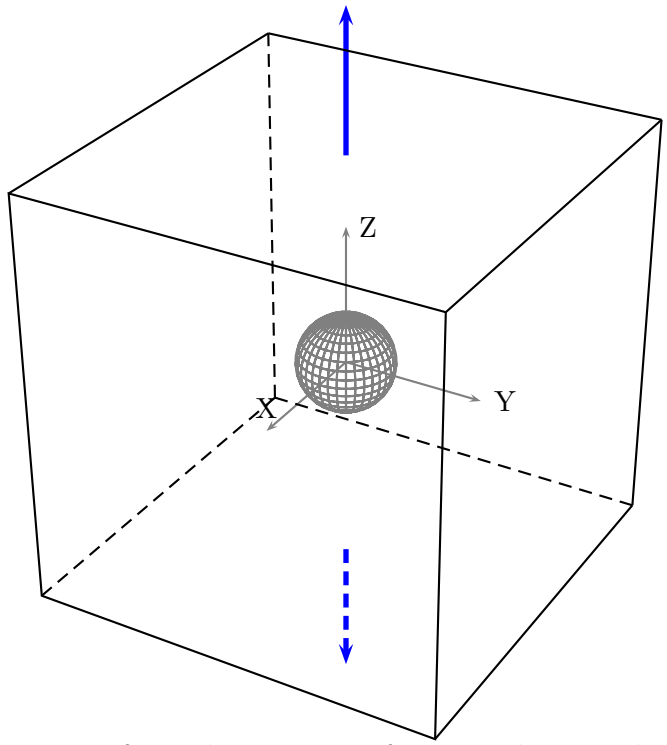


Figure 3.18: Diagram of simulation setup for a single crystal cubic domain and uniaxial strain.

Table 3.2: Table of simulation configurations for void growth under uniaxial expansion.

R_{void} (nm)	L/a_0	L (nm)	Void/box (%)	Total atom	Removed
0.3	28	10.122	0.0101	87,808	13
0.5	14	5.061	0.404	10976	43
1.0	28	10.122	0.404	87,808	369
2.0	56	20.244	0.404	702,464	2,899
4.0	112	40.488	0.404	5,619,712	22,663

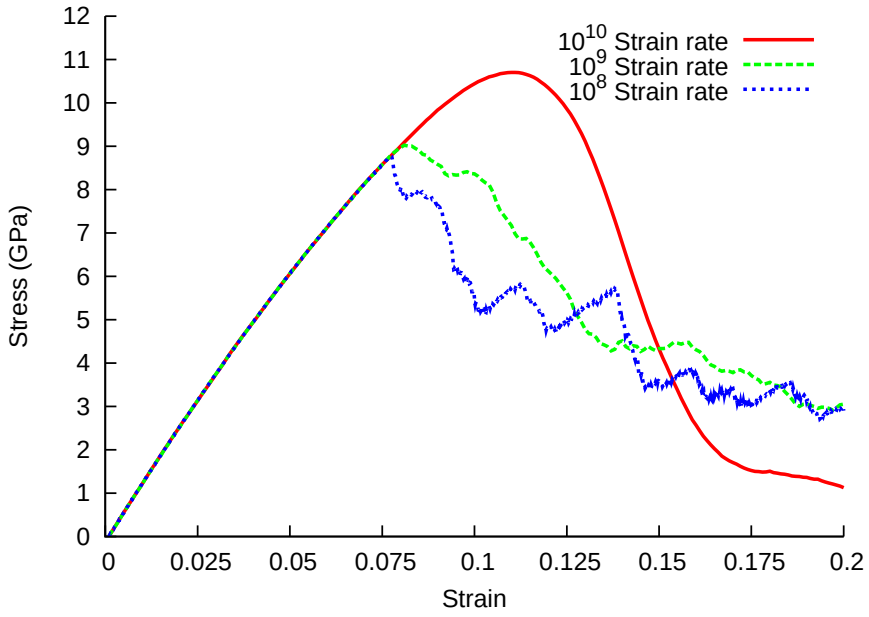
nanometer. This gives a void volume ratio of 0.404%. The total number of atoms in the simulation was 87439.

The domain with void was equilibrated for 2 to 3 picoseconds, using an nph ensemble (integration steps with a barostat, see detail in appendix A), to reach the zero pressure state. Then uniaxial expansion in the **z**-direction (aligned with [001] crystal direction) was applied with three different strain rates, 10^{10}, 10^{9} and 10^{8} s^{-1}. The target strain is 20% volume expansion in the **z**-direction. A smaller simulation domain (0.5 nm void radius) as well as two larger domains (2.0 nm and 4.0 nm void radius) were also prepared with the same exact void volume fraction of 0.404%. These small and larger models were run at strain rates of 10^{8} s^{-1} alone to compare the yield stress as a result of the difference in size scale. To compare with previous results from hydrostatic expansion strain, a domain sample with 1.0 nm void radius also underwent hydrostatic expansion strain.

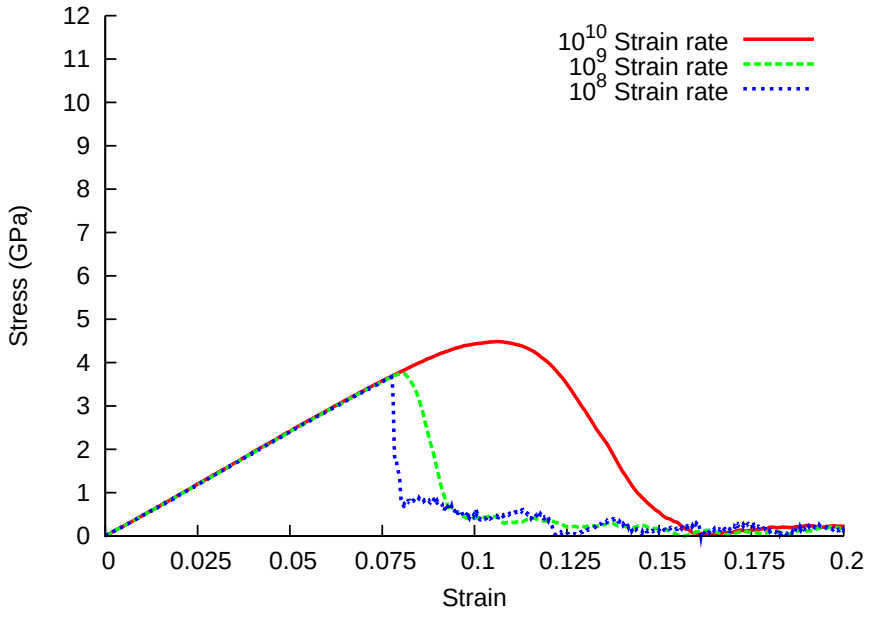
3.4.2 Result

Effect of Strain rate

With a high strain rate such as 10^{10} s^{-1}, the material yields at about 10.6 GPa. When the strain rate was lowered, the yield stress became lower to 9 GPa. This is similar to the scenario in Figure 3.2 for 2.88×10^{11} s^{-1} strain rate. The rate sensitivity is the result of dislocation lag, the behavior in which dislocations don't have enough time to relax stress concentrations. When dislocations have enough



(a) Mean stress



(b) von Mises stress

Figure 3.19: Plot of stresses against strain for different strain rates while fixing void size at 1.0 nm radius.

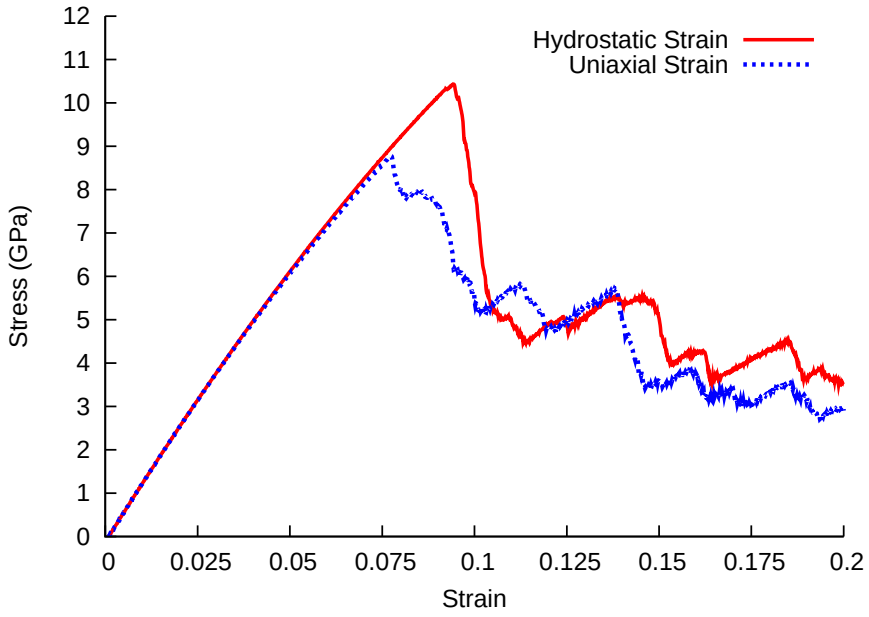
time to transport material and relax the stress in lower strain rate simulations, the drop in the yield stress becomes obvious. The stress drop indicates the failure of the material, and it is followed by a slight hardening behavior as dislocations reach the simulation boundary and interact with their periodic images. This behavior was only present in the two lower strain rates, as they allow dislocations enough time to nucleate and travel across the simulation domain. See Figure 3.19.

Difference Between Hydrostatic and Uniaxial Strain

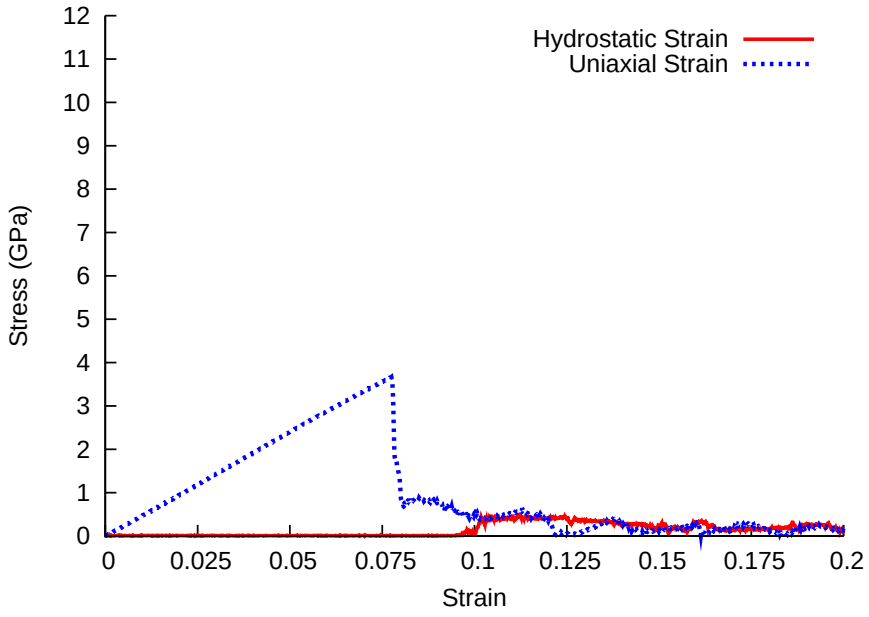
Hydrostatic and uniaxial strain simulations show a difference in the yield stress (the stress at which the voids start growing); the yield stress for uniaxial strain was lower, see Fig. 3.20. This difference is due to the fact that the shear stresses resulting from hydrostatic strain are considerably lower than in the case of uniaxial strain. An additional, but related, effect is that the formation of dislocations under hydrostatic strain was generally more complex.

Size Scale Difference

The size-scale dependence of the yield stress can be observed in uniaxial strain simulations with various sizes of voids, Fig. 3.21. The void fraction was kept constant at 0.404%. As the void size increases, the yield stress drops: it is 11 GPa for 0.5 nm and 7 GPa for 2 nm void. It should be mentioned that Potirniche et al. [61] have made similar calculations for nickel. Their void radius was varied from 0.75 to 4.5 nm whereas in the current calculations they were varied from 0.5 to 4 nm. Potirniche et al. [61] used a constant ratio of specimen to void dimensions. Other differences with the current calculations are the lateral boundary conditions; the boundaries were assumed free by Potirniche et al. [61] whereas here the state was assumed to be uniaxial strain with periodic boundary conditions. Free boundaries lead to necking of the sample. There is also a significant difference in strain rate: Potirniche et al. [61] used 10^{10}s^{-1}, whereas we used 10^{8}s^{-1}. This lower strain rate enabled individual observation of dislocations. In spite of the differences, the

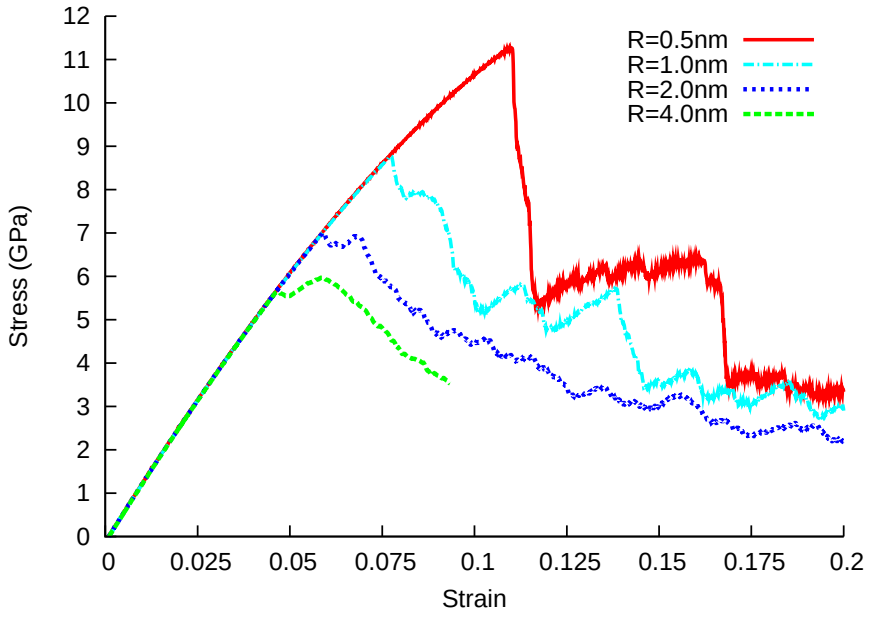


(a) Mean stress

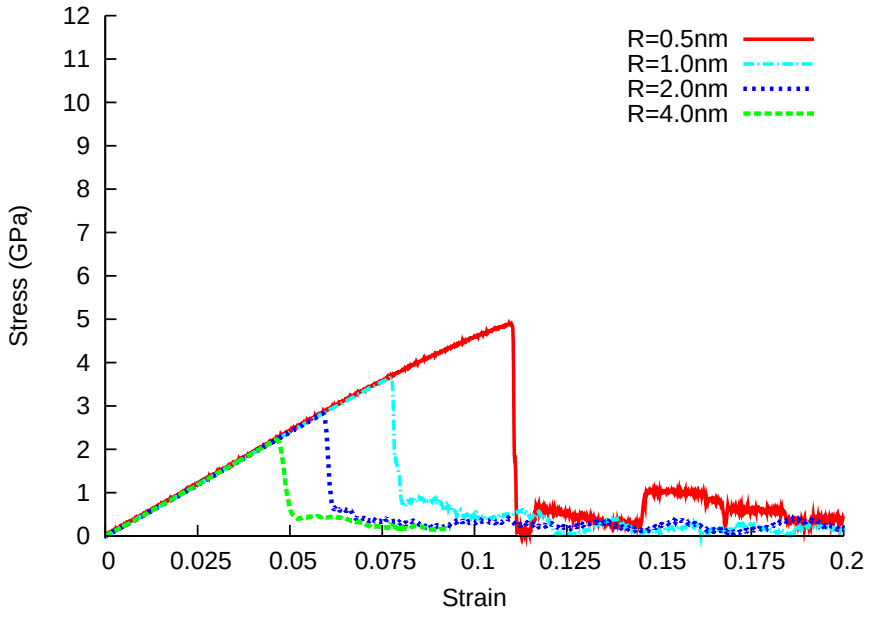


(b) von Mises stress

Figure 3.20: Plot of stresses against strain for different strain types while fixing void size at 1.0 nm radius and strain rate at 10^8 s^{-1}.



(a) Mean stress



(b) von Mises stress

Figure 3.21: Plot of stresses against strain for different void size at fixed strain rate of 10^8 s^{-1}.

stresses calculated herein are in good agreement with the ones by Potirniche et al. [61] for nickel; the ratios of stress to shear modulus vary from 0.12 to 0.22 in our calculations and from 0.17 to 0.26 in Figure 8 of Potirniche et al. [61]. Note that there are always error bars (on the order of ± 0.05 GPa) in the stress results from MD simulations, but it is small and can be negligible.

The void size dependence is in opposition to the Gurson criterion [1], which is size independent: $\sigma_y = g\left(f, \sigma_{kk}, \sigma_e\right)$. The stress at which the quasi-linear behavior is no longer obeyed is taken as the yield stress; it corresponds to the onset of dislocation activity. The stress drop is substantial for the smallest void radius (0.5nm); this is due to the fact that domain size is the smallest and therefore the dislocation density becomes the largest. For the larger voids, a greater extent of dislocation interaction takes place before the dislocations reach the boundaries of the "box". It is interesting that these results can also be interpreted in the framework of gradient plasticity [63, 64, 102, 103, 104, 105, 106], however, this transcends the goal of this report.

Figure 3.22(a) shows the yield stress (normalized to the shear modulus, G) plotted as a function of the normalized void radius, R/b, where b is the Burgers vector. The decrease of yield stress with increasing R/b is clear, and similar to Dávila et al. [60]. The von Mises stress was obtained from the three components of the principal stress in the uniaxial strain state, neglecting the cross terms, σ_{xy}, σ_{xz}, and σ_{yz} because there were much smaller than the diagonal terms.

$$\sigma_{vm} = \sqrt{\frac{((\sigma_{xx} - \sigma_{yy})^2 + (\sigma_{xx} - \sigma_{zz})^2 + (\sigma_{yy} - \sigma_{zz})^2)}{2}} \tag{3.1}$$

$$\sigma_m = \frac{(\sigma_{xx} + \sigma_{yy} + \sigma_{zz})}{3} \tag{3.2}$$

Note that the results of the mean stress will give a slightly different threshold at the yield point compared to von Mises stress. The atomistic results compare well with the analytical calculations by Lubarda et al. [41] which are obtained

from

$$\frac{\sigma_{cr}}{G} = \frac{b/R}{\sqrt{2}\pi(1-\nu)} \frac{(1+\sqrt{2}R_{core}/R)^4+1}{(1+\sqrt{2}R_{core}/R)^4-1} \tag{3.3}$$

where the radius of the core, R_{core}, was made equal to b, $2b$, and $4b$. The stresses calculated by Lubarda et al. [41] are local values at the surface of the void, whereas the current values are from the far field, and therefore a correction factor of 2 (stress concentration for the spherical void) was introduced.

By contrast, Gurson's formulation [1] is void size independent, since only the porosity, f, enters the expression

$$\Phi = \frac{\sigma_e^2}{\sigma_y^2} + 2f\cosh\left(\frac{\sigma_h}{2\sigma_y}\right) - 1 - f^2 \tag{3.4}$$

where σ_y is the uniaxial yield stress of the material, σ_e is the equivalent von Mises stress, σ' is the deviatoric stress and σ_h is the hydrostatic stress. The latter two are

$$\sigma_e = \left[\frac{3}{2}\sigma'_{ij}\sigma'_{ij}\right]^{\frac{1}{2}} \tag{3.5}$$

$$\sigma_h = \sigma_{kk} \tag{3.6}$$

The condition for plastic flow is

$$\Phi = 0. \tag{3.7}$$

The decomposition of the strain into hydrostatic and deviatoric parts is

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{3}\varepsilon_x & 0 & 0 \\ 0 & \frac{1}{3}\varepsilon_x & 0 \\ 0 & 0 & \frac{1}{3}\varepsilon_x \end{bmatrix} + \begin{bmatrix} \frac{2}{3}\varepsilon_x & 0 & 0 \\ 0 & -\frac{1}{3}\varepsilon_x & 0 \\ 0 & 0 & -\frac{1}{3}\varepsilon_x \end{bmatrix} \tag{3.8}$$

The corresponding stresses are

$$\boldsymbol{\sigma} = \sigma_{hydrostatic} + \sigma_{deviatoric} = K\begin{bmatrix} \frac{1}{3}\varepsilon_x & 0 & 0 \\ 0 & \frac{1}{3}\varepsilon_x & 0 \\ 0 & 0 & \frac{1}{3}\varepsilon_x \end{bmatrix} + 2G\begin{bmatrix} \frac{2}{3}\varepsilon_x & 0 & 0 \\ 0 & -\frac{1}{3}\varepsilon_x & 0 \\ 0 & 0 & -\frac{1}{3}\varepsilon_x \end{bmatrix} \tag{3.9}$$

where K is bulk modulus and G is shear modulus. From Eq. 3.6

$$\sigma_h = K\varepsilon_x \tag{3.10}$$

With deviatoric stress from Eq. 3.5 and Eq. 3.9

$$\sigma_e^2 = 4G^2\varepsilon_x^2 \tag{3.11}$$

And the Gurson yield function now becomes

$$\Phi = \frac{4G^2\varepsilon_x^2}{\sigma_y^2} + 2f\cosh\left(\frac{K\varepsilon_x}{2\sigma_y}\right) - 1 - f^2. \tag{3.12}$$

We can assume, to a first approximation, that strain rate imparted to the material (10^8 s^{-1}) is such that the theoretical shear stress is reached. Thus:

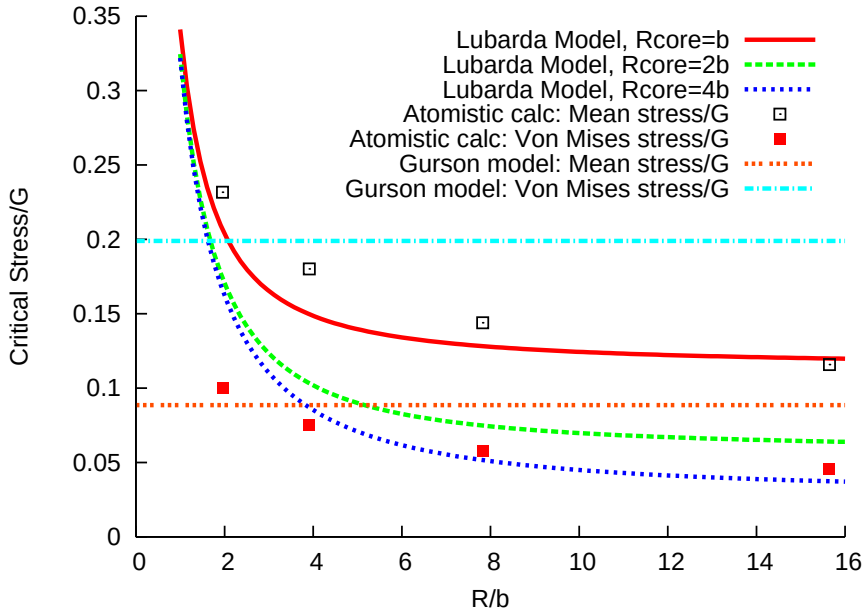
$$\tau = \frac{\sigma_y}{2} \simeq \frac{G}{10} \tag{3.13}$$

where G is 48.7 GPa [107], K is 130 GPa and σ_y is 9.74 GPa. With f=0.0042, one obtains $\varepsilon_x = 0.099484$ which results in a von Mises stress of 9.6897 GPa and a mean stress of 4.311 GPa. These values are introduced into Fig. 3.22(a) for comparison purposes. It can be seen that the Gurson model [1] is in reasonable agreement with the analytical Lubarda et al. [41] results and the MD calculations for larger void sizes. However, it does not have a void size dependence. Wen et al. [28] modified Gurson's model by incorporating the Taylor dislocation model. With the introduction of this scale dependent hardening component, the stress required to expand voids became scale dependent. This corresponds to the incorporation of gradient plasticity [63, 64, 102, 103, 104, 105, 106] into Gurson's model.

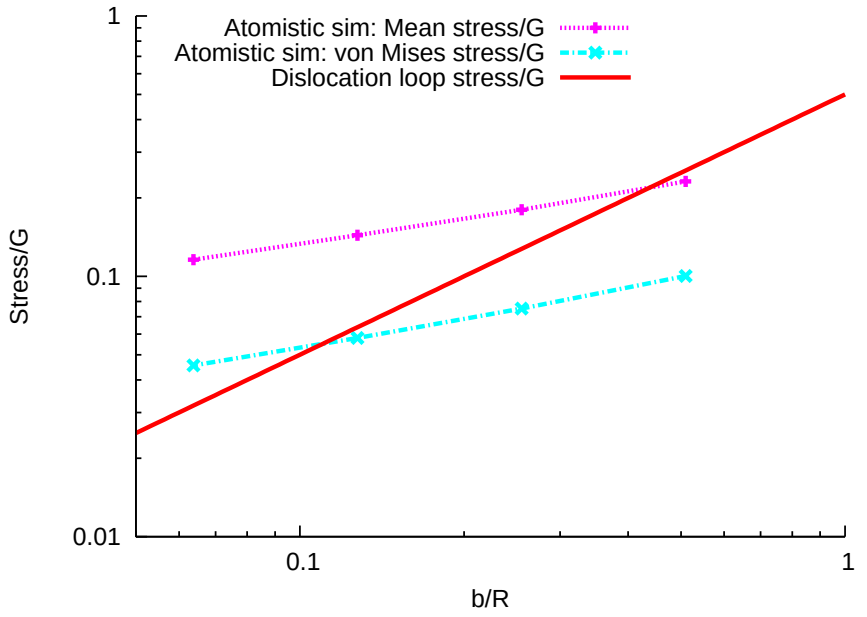
It is instructive to establish whether the void size dependence of the flow stress is directly linked to the stress required to bow dislocation loops into semi-circles (the stress minimum). Thus, the expression, e.g. [91], was used

$$\frac{\sigma}{G} = \alpha\frac{b}{R} \tag{3.14}$$

where α is a parameter equal to approximately 0.5 and R was taken as the void radius (assuming that loop and void radii are the same, to a first approximation).



(a)



(b)

Figure 3.22: Normalized critical stress against normalized void size; (a) with models; (b) on log-log scale with $\alpha = 0.5$.

The results are plotted in Fig. 3.22(b). In the log-log scale, both stresses obtained from atomistic simulations are linear and have expressions:

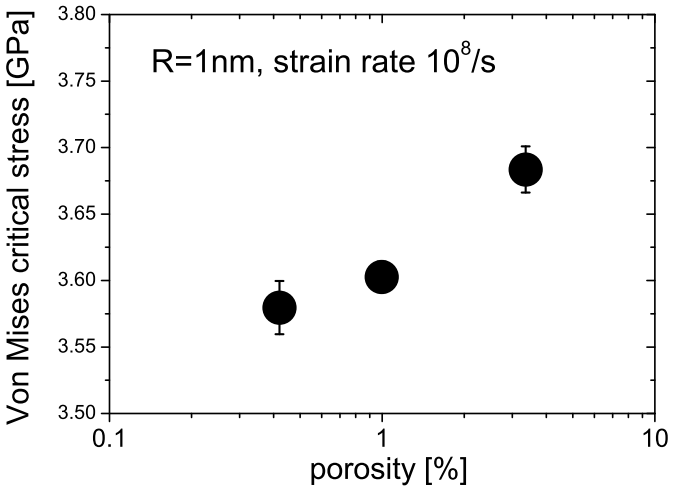
$$\begin{aligned} \frac{\sigma_m}{G} &= 0.2884 \left(\frac{b}{R}\right)^{0.3323} \\ \frac{\sigma_e}{G} &= 0.1288 \left(\frac{b}{R}\right)^{0.3798} \end{aligned} \tag{3.15}$$

The exponents and pre-exponential factors in the atomistic calculation are 0.33-0.38 and 0.13-0.29, in contrast with Eq. 3.14, in which they are 1 and 0.5, respectively. Nevertheless, the compatibility of the results is strong evidence that loop expansion beyond a semi-circle is an important contributing mechanism.

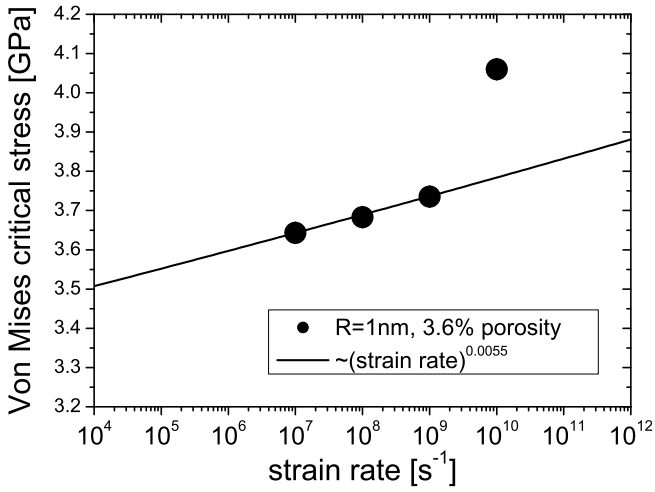
A few additional simulations for the uniaxial expansion at the strain rate of 10^8 s^{-1} on a single crystal copper with void radius of 1 nm were performed at several varying porosities (by changing the size of the overall simulation domain). The result is shown in Figure 3.23(a). The difference on the critical stress between the porosity of 0.404% and 1% is small, while it is larger for the case of 3.6% porosity. Furthermore, the simulations of 3.6% porosity single crystal copper under the uniaxial expansion for void radius of 1 nm were performed for the various strain rates of 10^7, 10^8, 10^9, and 10^{10} s^{-1}. The result is shown in Figure 3.23(b). The von Mises critical stresses are similar for the simulations with the strain rates of 10^7 to 10^9 s^{-1} and there is a large difference for the case of 10^{10} s^{-1} This shows that the strain rate of 10^8 s^{-1} which was adopted almost throughout this work is a reasonable trade off between the accuracy of the results and the computational resources [100].

3.4.3 Interpretation and Calculation

Figures 3.24 and 3.25 show both the MD simulations (left) and models (right) for the initiation and propagation of dislocations. Shear loops on different {111} planes, making 45° with the void, connect at the ⟨110⟩ intersection (Fig. 3.24(b)). Figures 3.24(a) and 3.24(c) show two views of a biplanar dislocation loop starting to form. This is more clear in the schematic of Fig. 3.24(d) showing the leading partials fully formed. As the leading partials expand, the trailing partials



(a) von Mises critical stress versus porosity



(b) von Mises critical stress versus strain rate

Figure 3.23: Plots of von Mises critical stress (yield stress) from additional simulations in collaboration with Eduardo M. Bringa [100].

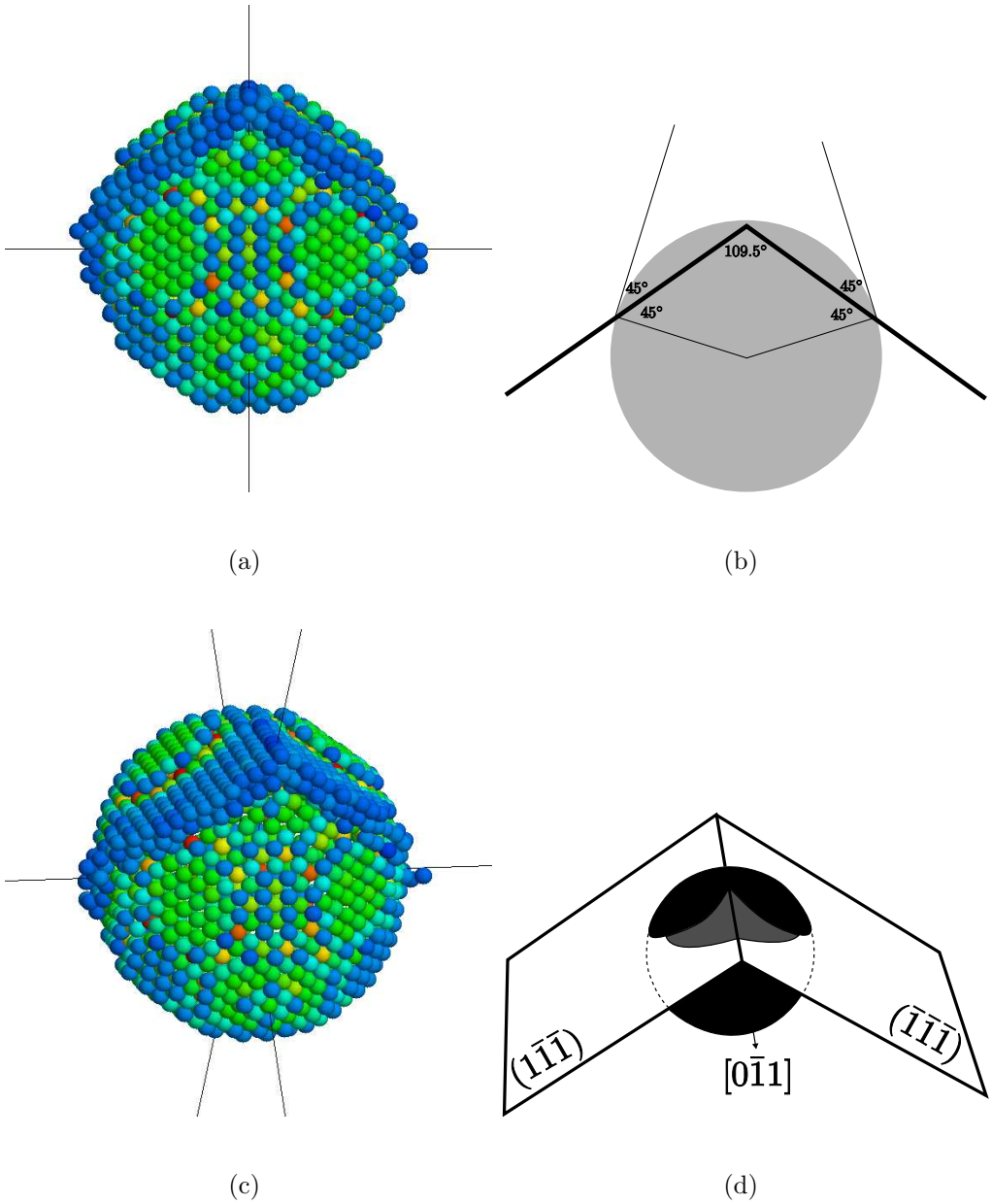
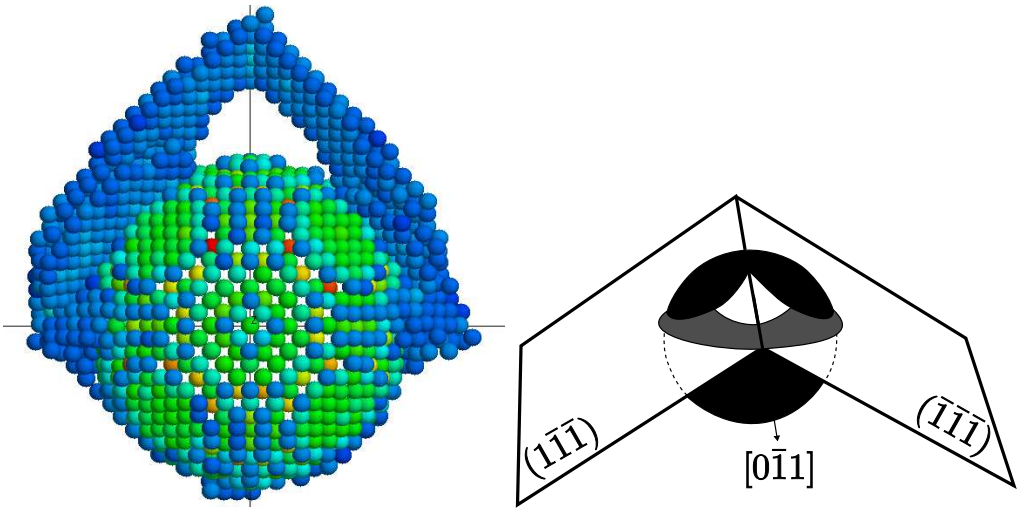
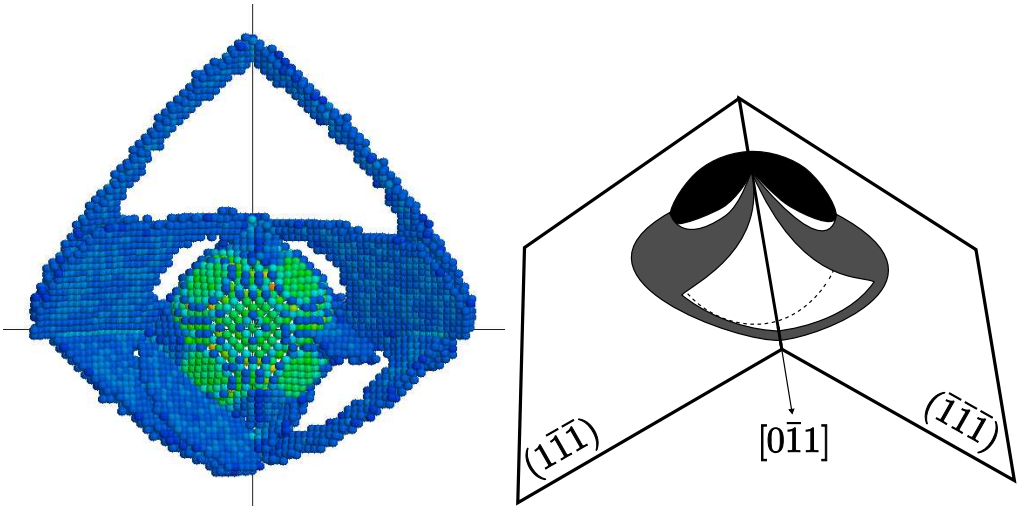


Figure 3.24: Initiation of plastic flow at void surface (at 590ps); (a) rendered atoms from MD; (b) diagram of $(1\bar{1}\bar{1})$ and $(\bar{1}\bar{1}\bar{1})$ slip planes intersecting sphere surface at 45°; (c) rendered atoms from (a), rotated to show two loops; (d) diagram showing leading partial dislocations.

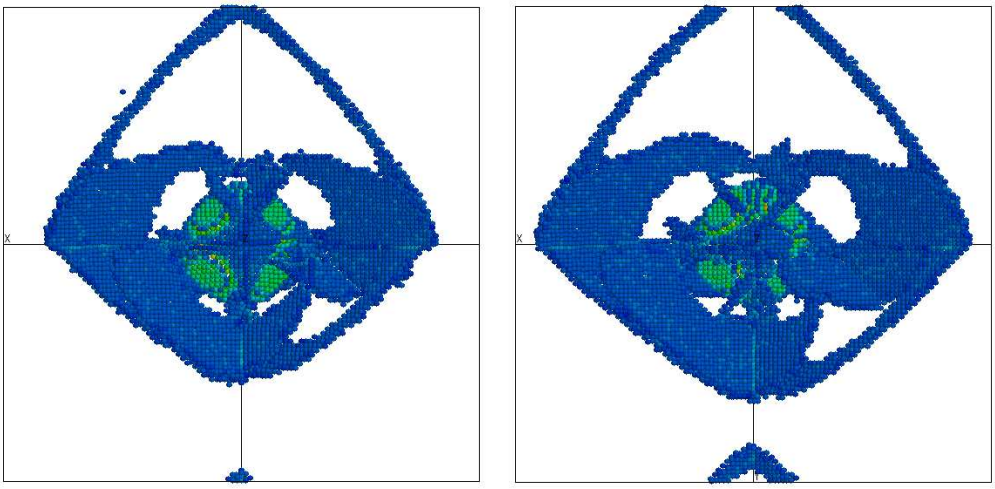


(a) (b)

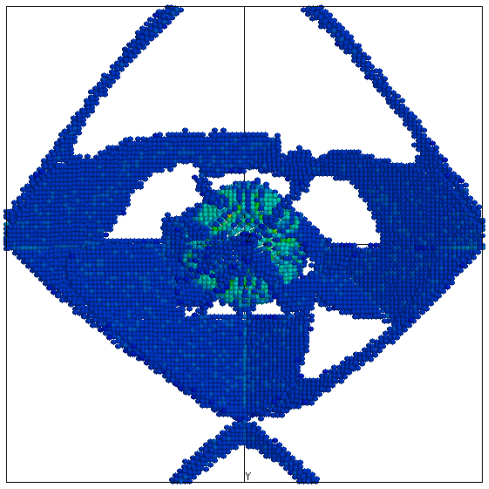


(c) (d)

Figure 3.25: Continued loop expansion; (a) rendered atoms from MD (591ps); (b) corresponding sketched diagram; (c) rendered atoms from MD (595ps); (d) corresponding sketched diagram.



(a) (b)



(c)

Figure 3.26: Later growth and interaction of shear loops emanating from void: (a) 597 ps; (b) 598 ps; (c) 599 ps.

follow them. More complex dislocation interactions take place as the shear loops propagate outwards (Figs.3.26).

In general, the stacking fault consists of two layers of atoms composing the plane of the dislocation. This is an artifact of filtering using centrosymmetry parameter (the range of centrosymmetry parameter for dislocation partial is $\sim$ 0.2-0.5). Figure 3.25 shows the continued expansion of the biplanar loops. When an additional layer forms on top of these two layers, the plane opens up with trailing edge of dislocation closing the stacking fault. At this point, it becomes a shear loop with a narrow stacking fault band. Shear loops continue to travel outwards from the void surface as they transport material, accommodating the growth of void. Dislocations travel at a very high velocity in the same level of sound speed in the material.

Dislocation planes are of the family {111}. Partial dislocations on two planes interact at the initiation stage, forming an angle in which they tie and travel together as they expand. This interaction happens for any two partial dislocation planes that nucleate next to each other, Fig. 3.24.

3.5 Calculation of Dislocation Interactions

For the shear loops postulated by Lubarda et al. [41] to undergo continued expansion, they have to intersect, if they form in the same (111). Six loops with edge dislocations at the center create the uniform expansion of the void segment (calota) if they can expand uniformly. In this section the energetics of the process are analyzed. Figure 3.27 shows three intersecting dislocations ($[0\bar{1}1]$, $[\bar{1}01]$ and $[\bar{1}10]$) in the (111) plane that intersects the void at 45°. These are the planes (shown in Fig. 1.8(b)) that maximize the shear stress. Three nascent loops are shown in Fig. 3.27(a); as they expand (Fig. 3.27(b)), their extremities touch and this would encourage a reaction. We analyse this for perfect and partial dislocations in the next subsection.

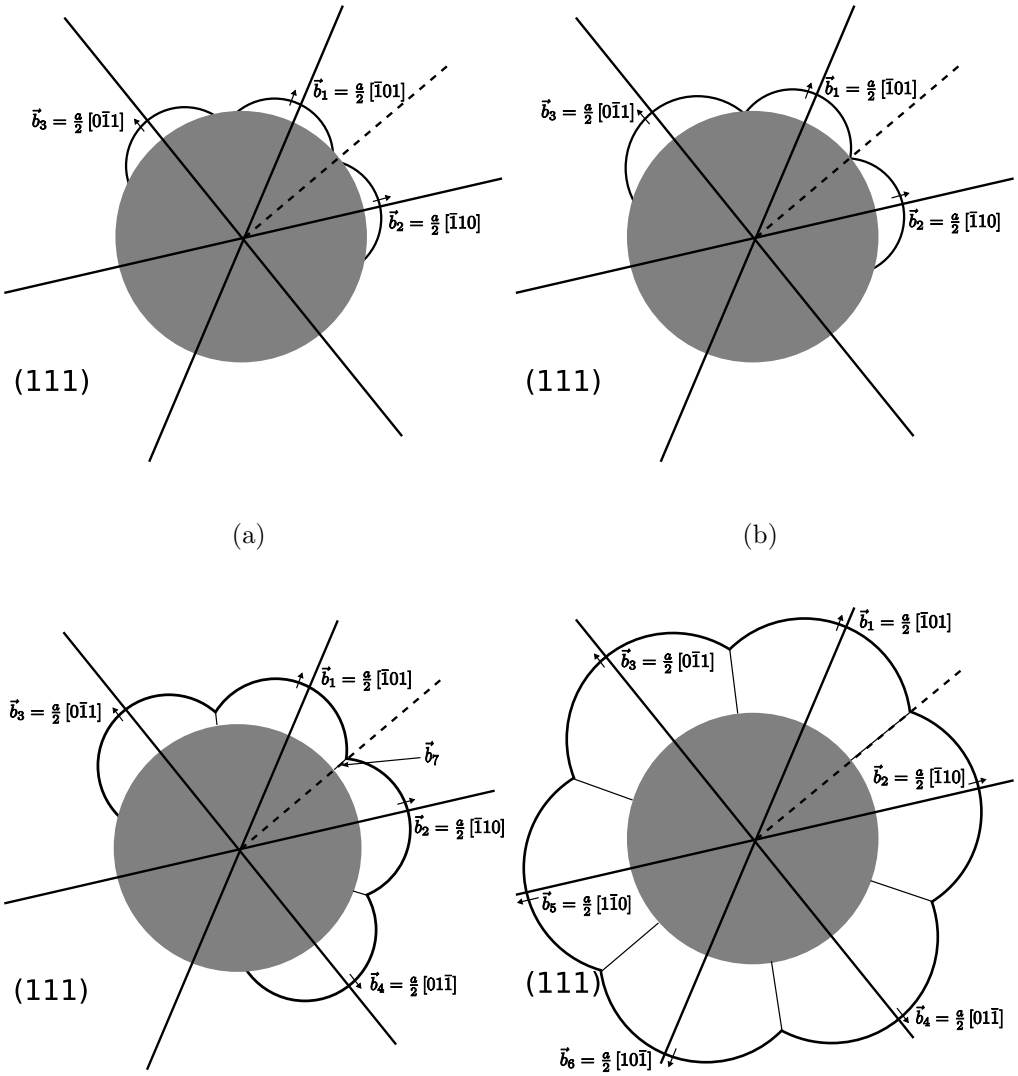


Figure 3.27: In-plane dislocation interactions; (a) before interaction; (b) onset of interaction; (c) interactions and reaction; (d) uniform expansion of loops leaving dislocation segments behind.

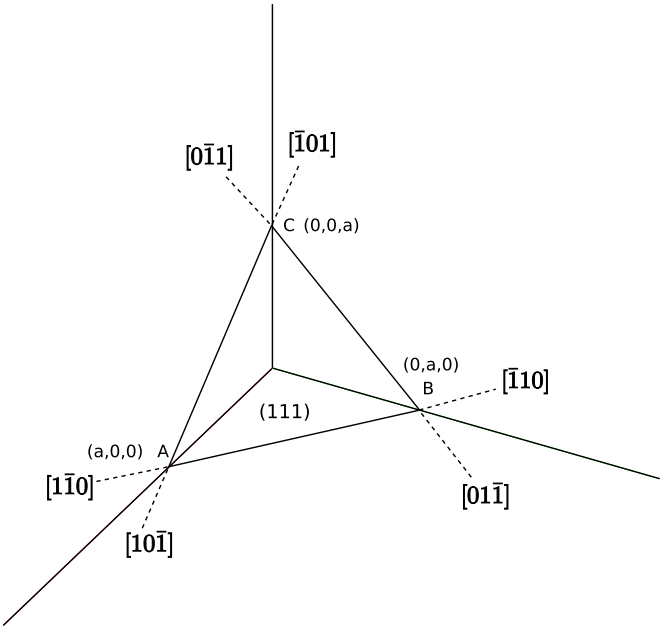


Figure 3.28: Plane (111) on principal directions.

3.5.1 In Plane Dislocation and Interactions

The following shows equations for a dislocation loop reaction in plane (111), Fig. 3.28.

For Perfect Dislocations

Figure 3.27(b) shows three dislocation loops with Burgers vectors $\vec{b}_1$, $\vec{b}_2$ and $\vec{b}_3$. The resulting reaction will lead to (Fig. 3.27(c))

$$\vec{b}_1 + \vec{b}_2 = \vec{b}_7. \tag{3.16}$$

The Burgers vector of $\vec{b}_7$ is (the Burgers vectors have to be subtracted in order to account for the dislocation lines; this is analogous to the interaction in a Frank-Reed source)

$$\frac{a}{2}[\bar{1}01] + (-)\frac{a}{2}[\bar{1}10] = \frac{a}{2}[0\bar{1}1]. \tag{3.17}$$

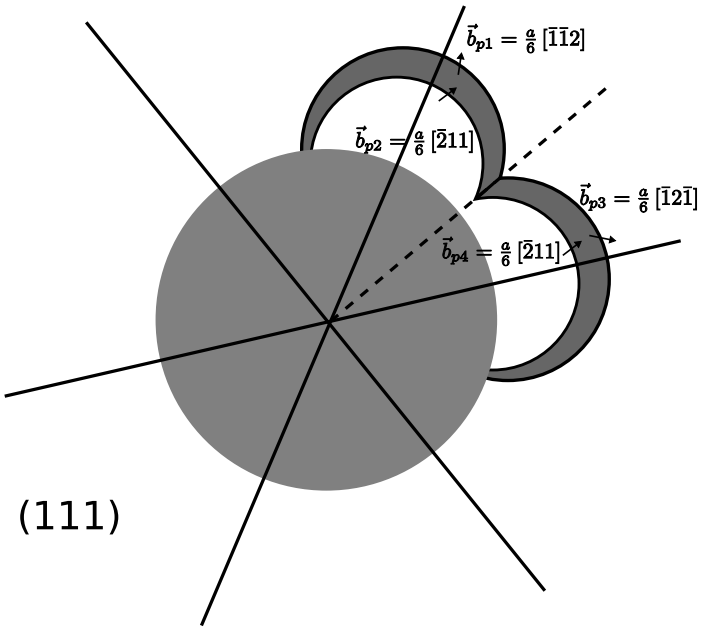


Figure 3.29: Plane (111) with in-plane partial dislocations interactions.

The simple energy reduction criterion is obeyed ($E = G\frac{b^2}{2}$) and the reaction takes place

$$G\frac{a^2}{2} + G\frac{a^2}{2} > G\frac{a^2}{2}. \tag{3.18}$$

Additionally, there is a change in overall dislocation length when a reaction occurs as shown in Fig. 3.27(c). One can estimate the equilibrium of the dislocation configuration by using the energy equation incorporating the lengths.

Thus, the configuration seen in Fig. 3.27(d) can be envisaged: six dislocation loops expanding uniformly in the same (111) plane, creating six segments through reactions. These segments are not mobile but are not sessile, since they have Burgers vectors in (111). They have edge character, with the Burgers vector perpendicular to the line. This configuration is slightly different from the one described by Marian et al. [56, 57].

For Partial Dislocations

The partial dislocations corresponding to $\vec{b}_1$ in plane (111) are (Fig. 3.29)

$$\vec{b}_1 = \frac{a}{2}[\bar{1}01] \Rightarrow \vec{b}_{p1} = \frac{a}{6}[\bar{1}\bar{1}2]\,; \vec{b}_{p2} = \frac{a}{6}[\bar{2}11] \tag{3.19}$$

and the partial dislocations corresponding to $\vec{b}_2$ in plane (111) are

$$\vec{b}_2 = \frac{a}{2}[\bar{1}10] \Rightarrow \vec{b}_{p3} = \frac{a}{6}[\bar{1}2\bar{1}]\,; \vec{b}_{p4} = \frac{a}{6}[\bar{2}11]\,. \tag{3.20}$$

The leading partials react as (again, we have to subtract $\vec{b}_{p3}$ from $\vec{b}_{p1}$ to account for dislocation line direction normalization)

$$\vec{b}_{p1} + \vec{b}_{p3} = \frac{a}{6}[\bar{1}\bar{1}2] + (-)\frac{a}{6}[\bar{1}2\bar{1}] = \frac{a}{2}[0\bar{1}1]\,. \tag{3.21}$$

The trailing reaction produces

$$\vec{b}_{p2} + \vec{b}_{p4} = \frac{a}{6}[\bar{2}11] + (-)\frac{a}{6}[\bar{2}11] = 0. \tag{3.22}$$

This is the same solution than for perfect dislocations, as expected. It is interesting to note that the leading partials react, creating a perfect dislocation and the trailing partials cancel.

3.5.2 Biplanar Dislocation and Interactions

A more detailed dislocation analysis is shown in Figs. 3.30 (perfect dislocations) and 3.31 (partial dislocations). In Fig. 3.30, two perfect dislocation loops, $\vec{b}_1$ and $\vec{b}_2$, forming on $(1\bar{1}\bar{1})$ and $(\bar{1}\bar{1}\bar{1})$, respectively, interact from the early state of dislocation formation. They have parallel Burgers vectors but are on different planes. The intersection line is also aligned with $[0\bar{1}1]$ which allows the two dislocations to glide without forming sessile segments. Actually, they cancel each other at the $\langle 110 \rangle$ intersection. An analysis analogous to the one made for in-plane dislocation interactions was carried out with the different that we now use $(1\bar{1}\bar{1})$ and $(\bar{1}\bar{1}\bar{1})$.

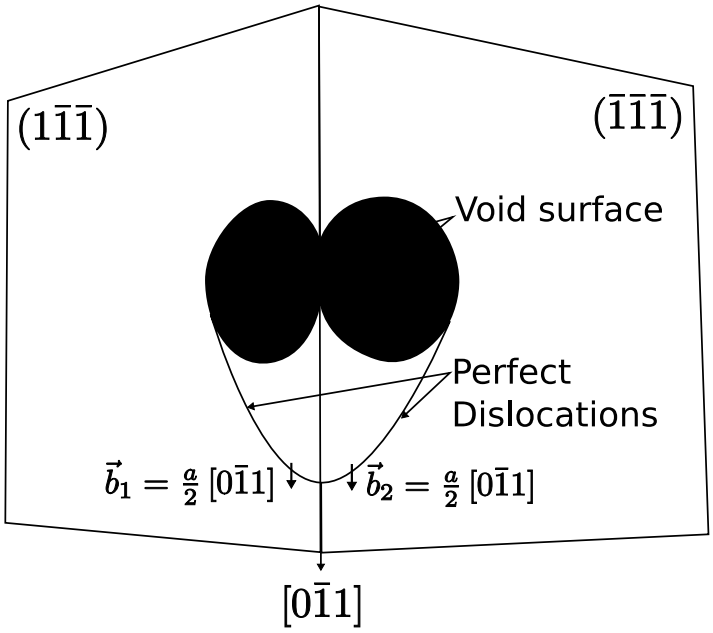


Figure 3.30: Top view of two perfect dislocations.

For Perfect Dislocations

$$\vec{b}_1 + \vec{b}_2 = \vec{b}_7 \tag{3.23}$$

$$\frac{a}{2}[0\bar{1}1] + (-)\frac{a}{2}[0\bar{1}1] = 0 \tag{3.24}$$

The energy becomes zero at the intersection line because the two perfect dislocations cancel each other. Thus, the biplanar loop does not require the creation of a radial dislocation. This is an energetic advantage over the planar loop emission mechanism.

For Partial Dislocations

The interaction of perfect dislocations can be extended to partial dislocations (Fig. 3.31). The leading partials $\vec{b}_{p1}$ and $\vec{b}_{p3}$ react and form a sessile dislocation $\frac{a}{3}[\bar{1}00]$; the trailing partials form an opposite dislocation $\frac{a}{3}[100]$. They cancel each

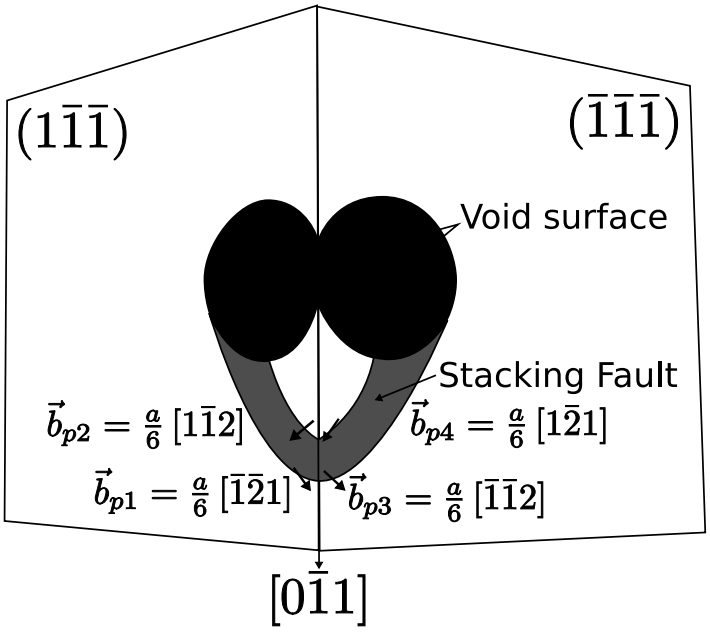


Figure 3.31: Top view of partial dislocations.

other as the partials move, leaving the crystalline structure undistorted behind. The $\frac{a}{3}[\bar{1}00]$ dislocation is sessile and has an energy below the $\vec{b}_{p1} + \vec{b}_{p3}$ sum. It constricts the loop at the slip-plane intersection. Hence, the biplanar shear loop mechanism is applicable to the case where perfect dislocations decompose into partials. This is also clearly seen in the simulation of Fig. 3.25. It should be noted that dislocation reactions should be $\vec{b}_1 - \vec{b}_2$ (and, accordingly, $\vec{b}_{p1} - \vec{b}_{p3}$ and $\vec{b}_{p2} - \vec{b}_{p4}$) because of the dislocation loop line vectors in the $(1\bar{1}\bar{1})$ and $(\bar{1}\bar{1}\bar{1})$ which are opposed.

The decomposition of a perfect dislocation $\vec{b}_1$ in $(1\bar{1}\bar{1})$ leads to

$$\vec{b}_1 = \frac{a}{2}[0\bar{1}1] \Rightarrow \vec{b}_{p1} = \frac{a}{6}[\bar{1}\bar{2}1]\,; \vec{b}_{p2} = \frac{a}{6}[1\bar{1}2]\,. \tag{3.25}$$

The energy criterion is ($\propto \vec{b}^2$)

$$\frac{a^2}{2} > \frac{a^2}{6} + \frac{a^2}{6} = \frac{a^2}{3} \tag{3.26}$$

The decomposition of a perfect dislocation $\vec{b}_2$ in $(\bar{1}\bar{1}\bar{1})$ is

$$\vec{b}_2 = \frac{a}{2}[0\bar{1}1] \Rightarrow \vec{b}_{p3} = \frac{a}{6}[\bar{1}\bar{1}2]\,; \vec{b}_{p4} = \frac{a}{6}[1\bar{2}1]\,. \tag{3.27}$$

The energy criterion is ($\propto \vec{b}^2$)

$$\frac{a^2}{2} > \frac{a^2}{6} + \frac{a^2}{6} = \frac{a^2}{3}. \tag{3.28}$$

The reaction between the leading partials is (note the sign change required for normalization of the dislocation line direction)

$$\vec{b}_{p1} + \vec{b}_{p3} : \frac{a}{6}[\bar{1}\bar{2}1] + (-)\frac{a}{6}[\bar{1}\bar{1}2] = \frac{a}{6}[0\bar{1}\bar{1}]\,. \tag{3.29}$$

The energy criterion is ($\propto \vec{b}^2$)

$$\frac{a^2}{6} + \frac{a^2}{6} = \frac{a^2}{3} > \frac{a^2}{18} \tag{3.30}$$

This reaction reduces energy. For the trailing partials

$$\vec{b}_{p2} + \vec{b}_{p4} : \frac{a}{6}[1\bar{1}2] + (-)\frac{a}{6}[1\bar{2}1] = \frac{a}{6}[011] \tag{3.31}$$

The energy criterion is ($\propto \vec{b}^2$)

$$\frac{a^2}{6} + \frac{a^2}{6} = \frac{a^2}{3} > \frac{a^2}{18}. \tag{3.32}$$

This reaction also reduces energy. The sum of the two reaction products is, as expected, zero. This is consistent with the calculations conducted on biplanar perfect dislocations. Thus the leading partials create a Lomer-Cottrell sessile dislocation $\frac{a}{6}[0\bar{1}\bar{1}]$; the trailing partials react similarly and create another Lomer-Cottrell sessile dislocation $\frac{a}{6}[011]$, which cancels the one created by the leading partials. This sessile dislocation constricts the loop at the slip-plane intersection. Hence, the biplanar shear loop mechanism is applicable to the case where perfect dislocations decompose into partials. This is also clearly seen in the simulation of Fig.3.25. The formation of sessile dislocations was successfully observed (molecular dynamics and quasi-continuum computational approaches) by Marian et al. [56, 57] and is confirmed here, although there are differences in the details of the reaction.

3.6 Void Growth Kinetics

The Cocks-Ashby [108, 109] model for void growth is, *strictu sensu*, only applicable to creep; the mechanisms of matter transfer are not dislocations but flow of vacancies along boundaries, surfaces, or dislocations (the latter is the power-law creep, vacancies promoting the climb of dislocation segments). However, its form is such that it can be used for an ideally plastic material with strain rate sensitivity. The constitutive equation for power-law creep is

$$\dot{\varepsilon}_{ss} = \dot{\varepsilon}_0 \left(\frac{\sigma_e}{\sigma_0} \right)^n , \tag{3.33}$$

where $\dot{\varepsilon}_{ss}$ is the equivalent strain rate, σ_e is the equivalent stress, and $\dot{\varepsilon}_0$, σ_0 and n are parameters. The "power-law creep" exponent is n and it is the inverse of the strain rate sensitivity. Cocks and Ashby [108, 109] applied continuity conditions to it in the presence of a void and obtained the following equation for the evolution of damage, D

$$\frac{dD}{dt} = \beta \dot{\varepsilon}_0 \left[\frac{1}{(1-D)^n} - (1-D) \right] \left(\frac{\sigma_e}{\sigma_0} \right)^n , \tag{3.34}$$

This constitutive equation was implemented by Bamman et al. [110] into FEM codes to predict the failure of metals. The evolution of damage predicted by Cocks and Ashby [108, 109] is dependent on the parameter n. In creep, it has a value between 1 and 10 (with $n = 5$ being the most quoted value [111]), but in defining the strain rate sensitivity of plastic flow, the value of $(1/n)$ is much lower, on the order of 0.01, corresponding to $n = 100$. Integration of the Equation 3.34 yields the closed form solution:

$$\ln|(1-D)^{n+1} - 1| - \ln|(1-D_0)^{n+1} - 1| = (n+1)\dot{\varepsilon}\beta t \tag{3.35}$$

The damage evolution in the MD calculations was estimated by considering the radius increase as a function of time. The initial damage D_0 was evaluated for a void radius $r = 2$ nm that gave an initial damage level $D_0 = 0.004$. Figure 3.32(a) shows the evolution of damage for three crystalline orientations: [001], [110] and [111]. The computation was carried out for a longer time for [001]; nevertheless,

the results from the three orientations are compatible. The predictions of the Cocks-Ashby model are shown in the same plot and good agreement is obtained, the shape of the curves being similar. A significant difference is that there is an incubation time of $\sim$ 400 ps for the MD computations. This is the result of the time required to nucleate a dislocation (shear) loop. The match with Cocks-Ashby is best for a value of $n = 30$. This corresponds to a strain-rate sensitivity of 0.033. This value is somewhat higher than the strain rate sensitivity often used for Ni at lower strain rates: 0.01. The higher strain rate sensitivity can be justified by the exceedingly high strain rate used in the present MD simulations: $10^8 s^{-1}$. The Cocks-Ashby prediction can be easily extended to larger damages and this is shown in Figure 3.32(b) for the values of n used in Figure 3.32(a). There is a gradual increase in the rate until $D = 0.2$. Beyond this value, damage proceeds essentially instantly. Note that there was an incubation time for MD simulation where void did not grow until there were dislocation activities. Therefore, MD results in Figure 3.32(a) should be shifted in order to have zero time at initial growth of void (this might result in a match of $n = 60 \sim 70$).

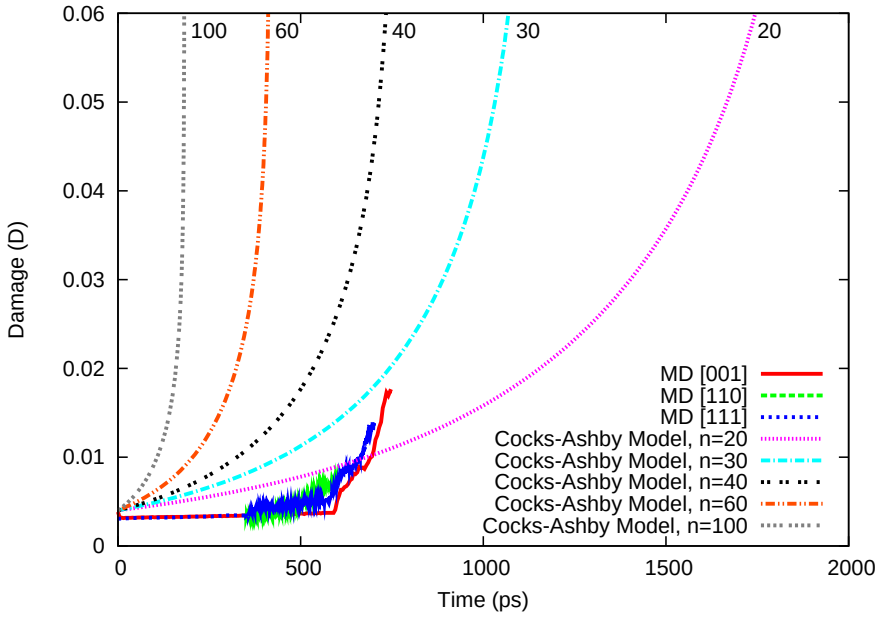
3.7 Density of Geometrically Necessary Dislocations

It is possible to estimate the total dislocation length around the expanding void using Ashby's [112, 113, 40] concept of geometrically-necessary dislocations. This can be done in an approximate manner by assuming that the dislocation loops transport matter outside.

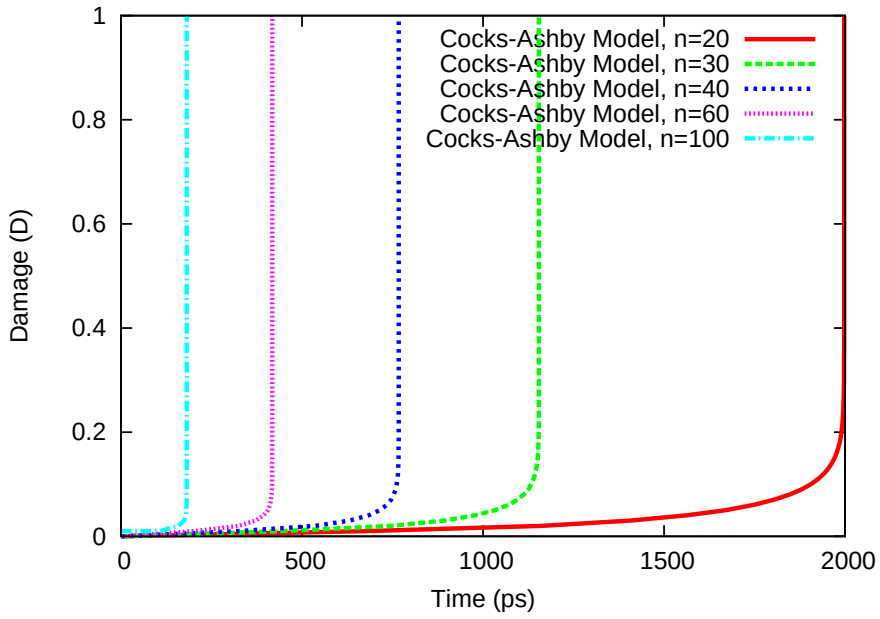
The length of a circumnavigating loops (each composed of six initial loops whose ends react) is, at an angle of 45° with the surface

$$\triangle L = 2\left(\frac{2\pi r}{\sqrt{2}}\right)k + 6(kr - r), \tag{3.36}$$

where k is the extension ratio of the loop from its original value ($k = R/r$).



(a)



(b)

Figure 3.32: Damage vs Time; (a) comparison of MD simulations and Cocks-Ashby equation (b) predictions from Cocks-Ashby equation (n is exponent in power-law constitutive equation).

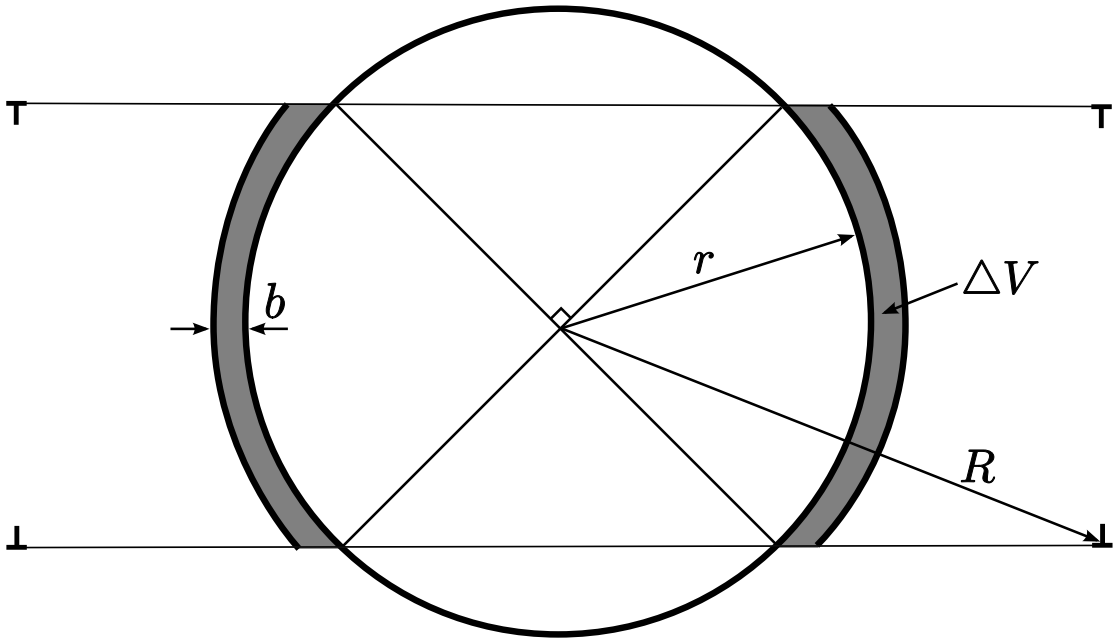


Figure 3.33: Volume increment generated in void by the expansion of two shear loop rings.

The distance that the dislocations travel outwards determines the radius R of the work-hardened layer (Fig. 3.33). The two terms represent the circular loop and six radii resulting from the reactions of loops extremities. The formation of two loops expands the void by a volume $\triangle V$ (Fig. 3.33)

$$\triangle V \simeq 2\sqrt{2}\pi r^2 b \tag{3.37}$$

The corresponding average increase in void radius, $\triangle r$, ignoring the distortion and other effects, is

$$\triangle r = (\triangle V)^{\frac{1}{3}} = (2\sqrt{2}\pi r^2 b)^{\frac{1}{3}}. \tag{3.38}$$

The ratio of equations gives

$$\frac{dL}{dr} = \frac{\left[\frac{4\pi k}{\sqrt{2}} + 6(k-1)\right] r}{\left(2\sqrt{2}\pi r^2 b\right)^{\frac{1}{3}}}. \tag{3.39}$$

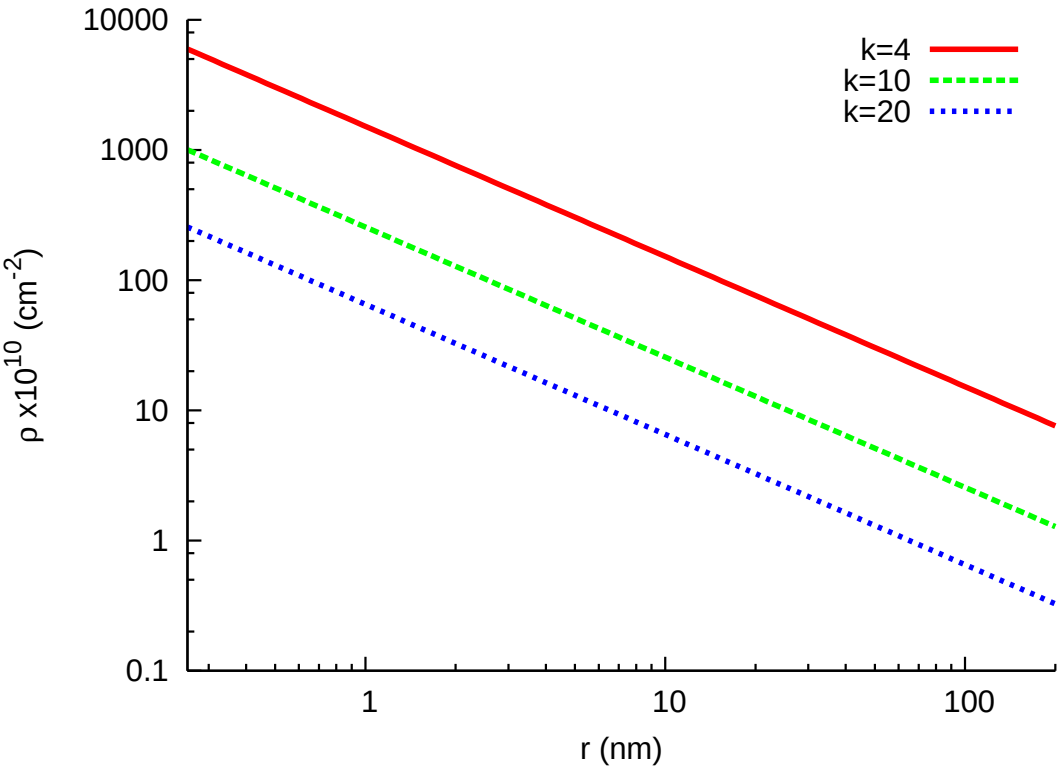


Figure 3.34: Calculated density of geometrically-necessary dislocations as a function of the ratio $k = \frac{R}{r}$.

Integrating gives

$$L = \frac{\left[\frac{4\pi k}{\sqrt{2}} + 6(k-1)\right]}{\left(2\sqrt{2}\pi b\right)^{\frac{1}{3}}} \int_{r_0}^{r_f} r^{\frac{1}{3}} dr. \tag{3.40}$$

If we make $r_0 = 0$

$$L = \frac{3\left[\frac{4\pi k}{\sqrt{2}} + 6(k-1)\right]}{4\left(2\sqrt{2}\pi b\right)^{\frac{1}{3}}} r^{\frac{4}{3}}. \tag{3.41}$$

The work-hardened volume is equal to

$$V_{wh} = \frac{4}{3}\pi R^3 - \frac{4}{3}\pi r^3 = \frac{4}{3}\pi\left(k^3 - 1\right) r^3. \tag{3.42}$$

The dislocation density is defined as

$$\rho = \frac{L}{V_{wh}} = \frac{9\left[\frac{4\pi k}{\sqrt{2}} + 6(k-1)\right]}{16\left(2\sqrt{2}\pi b\right)^{\frac{1}{3}} \pi\left(k^3 - 1\right)} r^{-\frac{5}{3}} = P(k) r^{-\frac{5}{3}}. \tag{3.43}$$

The densities are plotted for values of k varying from 4 to 20, in Fig. 3.34. These values are consistent with dislocation densities in highly work-hardened metals. It is evident that k has to be larger for smaller voids, consistent with the mean free path of dislocations. As the void expands, the dislocation density can be accommodated with relatively a smaller work-hardened region. The prediction from Eq. 3.43 applies only to the geometrically-necessary dislocations and does not incorporate dislocation interaction effects that contribute to the statistically-stored dislocation density.

3.8 Void Growth in Single Crystals with Different Loading Orientations

Following the simulations of void growth in single crystal copper under uniaxial expansion aligned with [001], the result from the void radius of 2 nm shows that the size of the simulation domain is not too large to compute repeatedly and not too small to produce evidents of dislocation interactions for detailed studies. We use

the setup of the 2 nm void radius for two more simulations of uniaxial expansion along [110] and [111] of single crystal copper. The void/volume ratio was kept constant at 0.404% with the uniform expansion by 10^8 s^{-1} strain rate.

3.8.1 Simulation Setup

The molecular dynamics LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [78] code was used in this investigation. For the FCC copper structure, an EAM [80] Mishin et al. [83] potential was used. The number of atoms was varied from 10^5 to 10^7, and calculations were performed on parallel PCs and on the supercomputer at San Diego Super Computer Center.

The single crystal copper domain was a cube with a spherical void at the center, Figure 3.18. Periodic boundaries were used perpendicular to the expansion plane, providing a uniaxial strain state. The different domains were subjected to uniaxial strain along [100], [110] and [111]. All simulations were done at an initial temperature of 150 K and strain rate of $10^8 s^{-1}$ for times up to 2000 picoseconds, corresponding to 20% volume strain. Visualization of stacking faults representing dislocations was conducted with a filter using a centrosymmetry parameter [90].

3.8.2 Results

The calculations (for voids with 2 nm radius) were performed for three orientations of the tensile axis; the Schmid factors were shown in Table 3.3, they have the following number of slip systems with the highest magnitude of Schmid factor:

[100] eight slip systems
[110] four slip systems
[111] six slip systems

Loops emission occurs, as postulated by Lubarda et al. [41], at the line corresponding to the intersection of the slip plane making an angle of 45° with the surface of the void and the void surface, which maximizes the shear stress. The

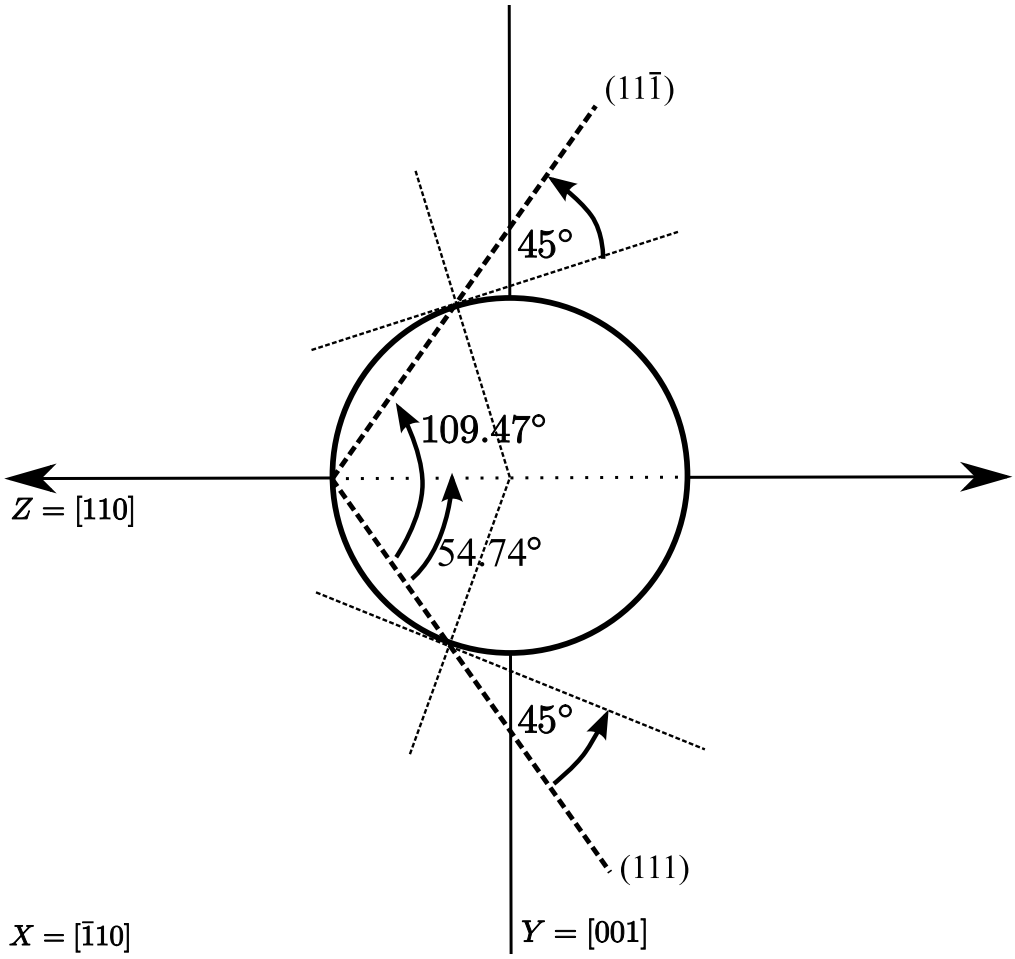


Figure 3.35: Schematic showing traces of two slip planes intersecting void at 45°: loading axis ([110]) marked by arrows.

Table 3.3: Table of Schmid factor calculation for three loading directions.

Slip plane	Slip direction	Load [001] S.F.	Load [110] S.F.	Load [111] S.F.
(111)	$[0\bar{1}1]$	0.408248	-0.408248	0
	$[\bar{1}01]$	0.408248	-0.408248	0
	$[\bar{1}10]$	0	0	0
$(\bar{1}1\bar{1})$	$[0\bar{1}\bar{1}]$	0.408248	0	0.272166
	$[\bar{1}01]$	-0.408248	0	0
	$[\bar{1}\bar{1}0]$	0	0	0.272166
$(\bar{1}\bar{1}1)$	$[0\bar{1}\bar{1}]$	-0.408248	0.408248	0.272166
	[101]	0.408248	-0.408248	-0.272166
	$[1\bar{1}0]$	0	0	0
$(1\bar{1}\bar{1})$	$[01\bar{1}]$	0.408248	0	0
	[101]	-0.408248	0	-0.272166
	$[\bar{1}\bar{1}0]$	0	0	0.272166

traces of two slip planes are illustrated in Figure 3.35 for a [110] loading direction (marked by arrows). The 45° angles with the surface are marked, and they make an angle of 109.47°. The sequence of shear loop initiation and expansion is demonstrated here for the three loading orientations, [110], [100], and [111], in order of complexity.

Loading Application Direction [110]

Figure 3.36 shows the sequence of expansion of a shear loop for [110] loading. The atoms on the surface of the voids are green, yellow, orange and red. The atoms in the stacking fault, indentified through the centrosymmetry parameter [90], are light blue. Only the leading partial dislocation is emitted during the computational time (limited by the domain size), and the (111) slip plane, as expected, makes an angle of 45° with the surface of the spherical void (radius = 2nm). This angle is shown in Figure 3.36(d). The expansion of the loop proceeds as shown in Figures 3.36(a-c). The loop acquires a heart shape (Fig. 3.36(c)). In Figure 3.36(d) a second loop is shown. For this direction of loading, the application of Schmid equation predicts four slip systems with highest Schmid factors (=0.408):

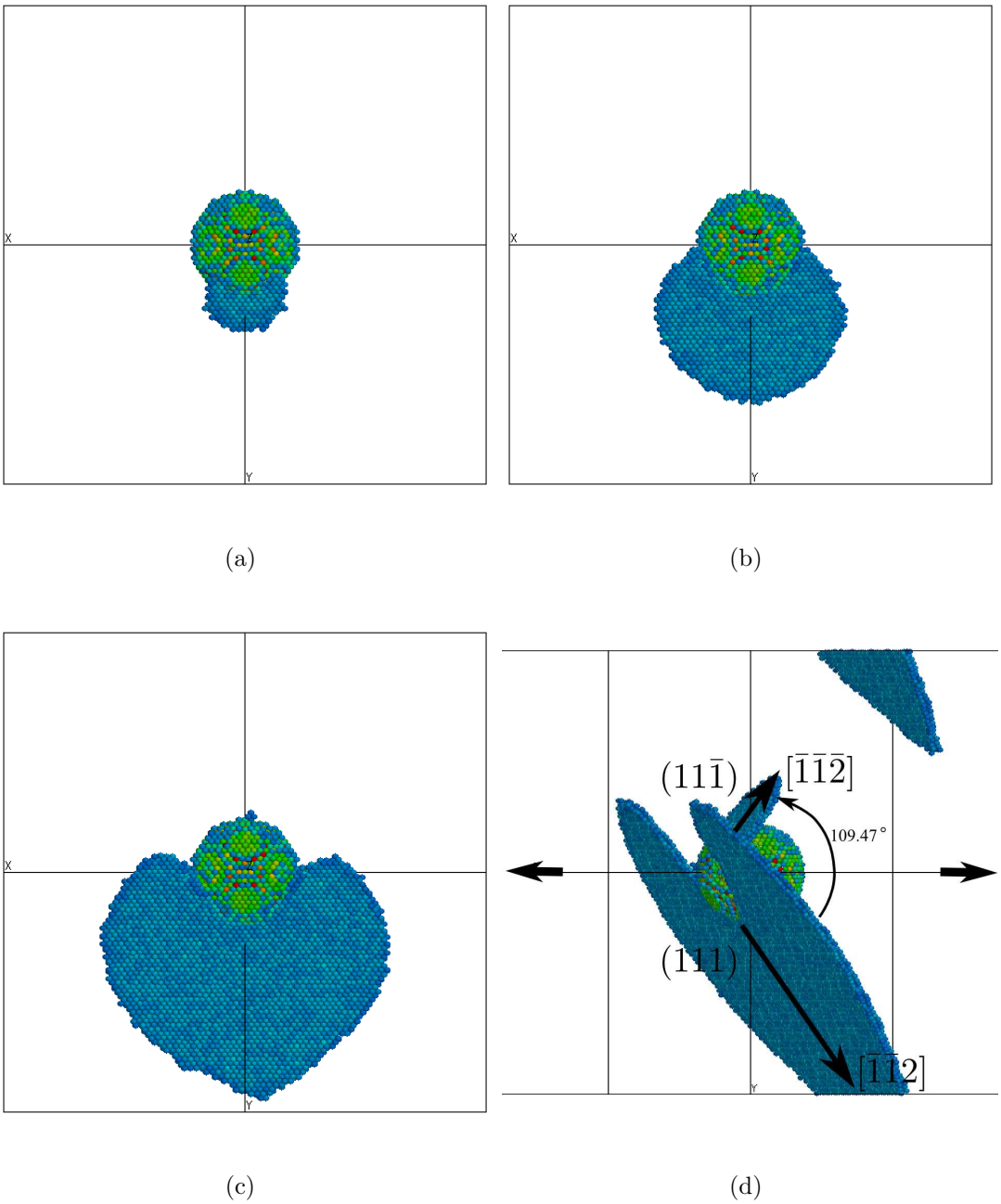


Figure 3.36: Sequence of shear loop nucleation and growth for [110] loading direction. Note directions and planes in (d) as well as second loop forming (loading direction perpendicular to plane of paper for Figs. 3(a-c) and marked in Fig. 3(d)).

$$\begin{array}{rcl} (111) & \rightarrow & [0\bar{1}1] \\ & \rightarrow & [\bar{1}01] \\ (\bar{1}\bar{1}1) & \rightarrow & [0\bar{1}\bar{1}] \\ & \rightarrow & [101] \end{array}$$

Figure 3.36(d) shows that indeed the two slip planes ((111) and $(\bar{1}\bar{1}1)$) make an angle of 54.7° with the loading axis.

Loading Application Direction [100]

For this orientation (sequence shown in Fig. 3.37), a cooperative growth of partial dislocation loops is observed. This mechanism is analyzed in detail by Traiviratana et al. [114] in a forthcoming paper. A biplanar shear loop emerges from the surface of the void, on planes $(1\bar{1}\bar{1})$ and $(\bar{1}\bar{1}\bar{1})$, marked in Figure 3.37(b). These planes define the maximum Schmid factor orientations. The two leading partial dislocations advance, moving away from the void. The trailing partials are subsequently formed, and the dislocation that is formed by the reaction of the leading partials ($\frac{a}{6}[\bar{1}\bar{2}1] - \frac{a}{6}[\bar{1}\bar{1}2] = \frac{a}{6}[0\bar{1}\bar{1}]$) is cancelled by the one forming by the reaction of the trailing partials ($\frac{a}{6}[1\bar{1}2] - \frac{a}{6}[1\bar{2}1] = \frac{a}{6}[011]$). Upon further loading, additional loops form on other planes (Figs. 3.37(c) and (d)).

Loading Application Direction [111]

The growth sequence is shown in Figure 3.38. Similar to [110] and [100], the loading axis is perpendicular to the plane of the paper. Three loops are simultaneously generated on different planes. As the shear loops expand by the glide of the leading partial dislocations, the trailing partials form (Fig. 3.38(c)). Figure 3.39(a) shows the schematic of the three slip planes with the leading and trailing partial dislocations as well as the stacking faults shown. A more detailed MD view of the expansion of the triplanar loops, shaped like a parachute, is seen in Figure 3.39(b). The three slip planes are $(1\bar{1}\bar{1})$, $(\bar{1}\bar{1}1)$, and $(\bar{1}1\bar{1})$, and the slip directions within each plane are indicated. Schmid factor computations predict the following six slip systems with highest Schmid factors (=0.272) involving the three planes:

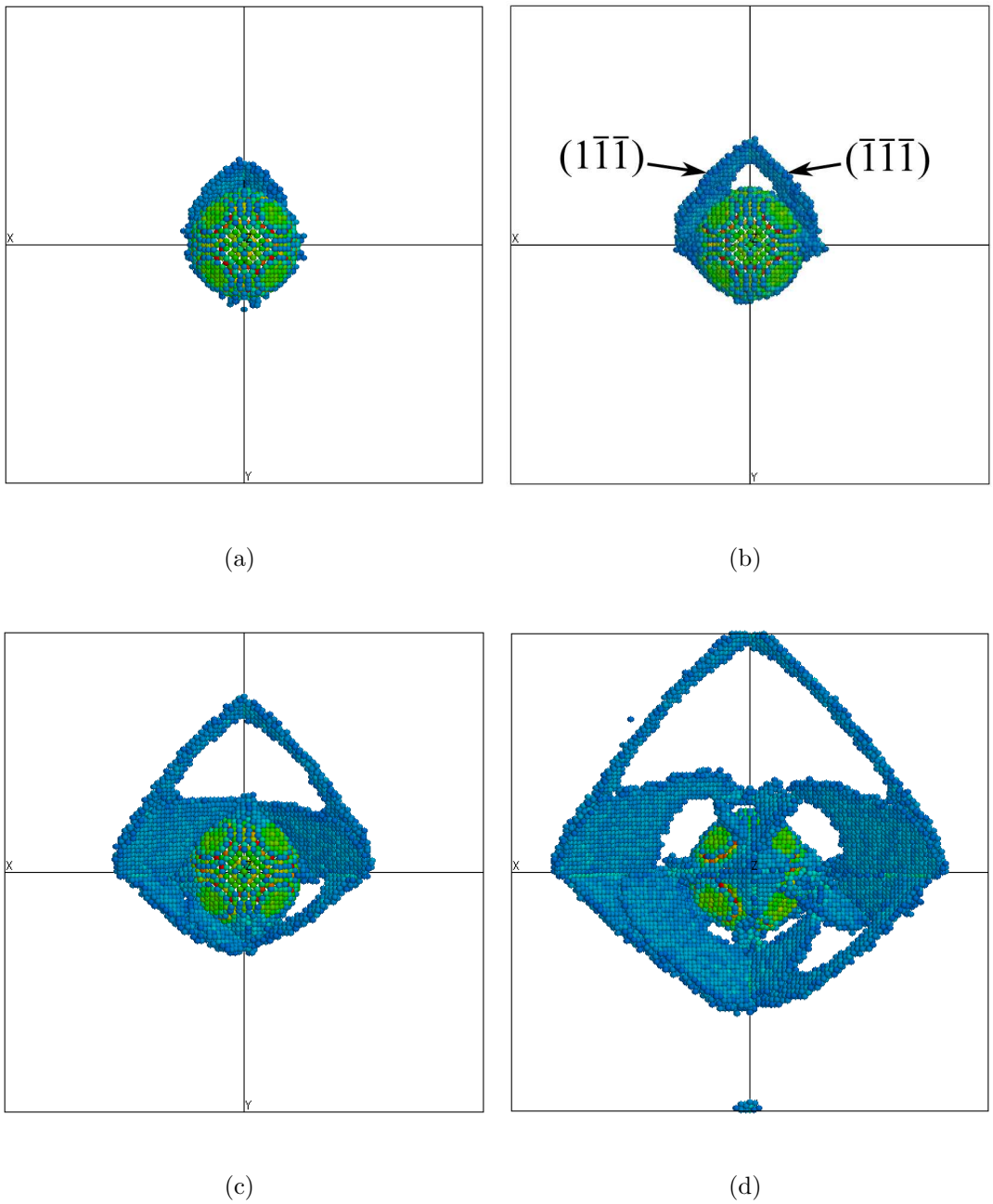
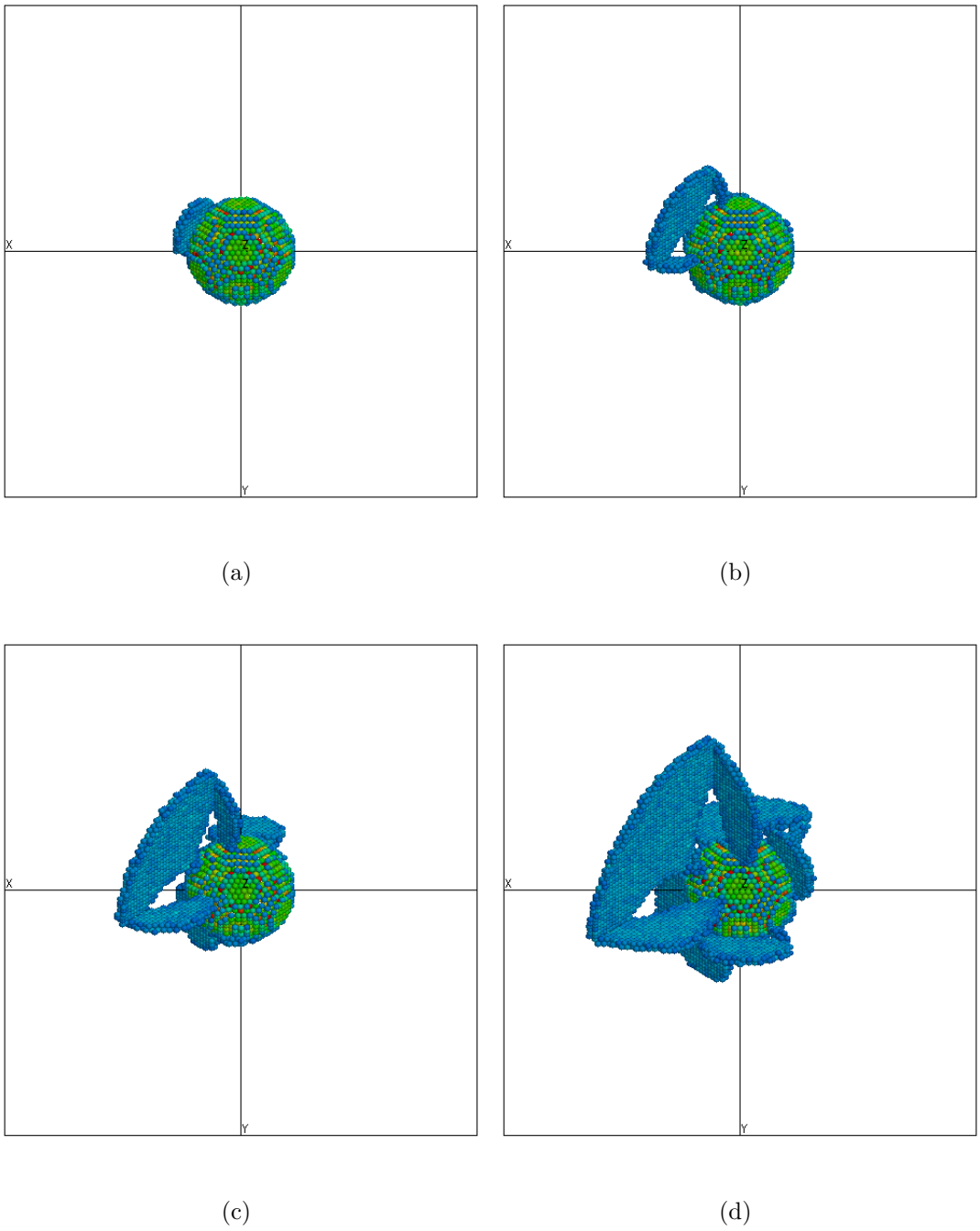


Figure 3.37: Sequence of loop nucleation and growth for loading along [100] (loading direction perpendicular to the plane of paper). Note two loops reacting and forming biplanar loop.



(a) (b)

(c) (d)

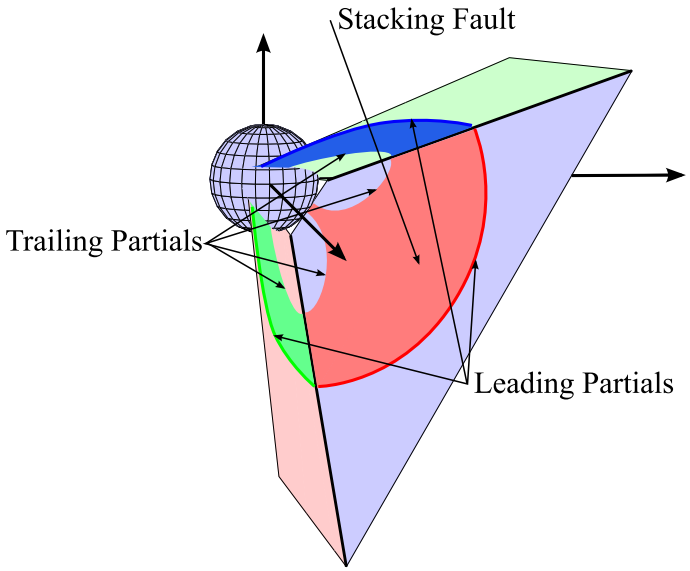
Figure 3.38: Sequence of loop nucleation and growth for loading along [111] (loading direction perpendicular to the plane of paper). Note the formation of loops on three planes in (b).

$$
\begin{array}{rcl}
(\bar{1}1\bar{1}) & \rightarrow & [0\bar{1}\bar{1}] \\
 & \rightarrow & [\bar{1}\bar{1}0] \\
(\bar{1}\bar{1}1) & \rightarrow & [0\bar{1}\bar{1}] \\
 & \rightarrow & [101] \\
(1\bar{1}\bar{1}) & \rightarrow & [101] \\
 & \rightarrow & [\bar{1}\bar{1}0]
\end{array}
$$

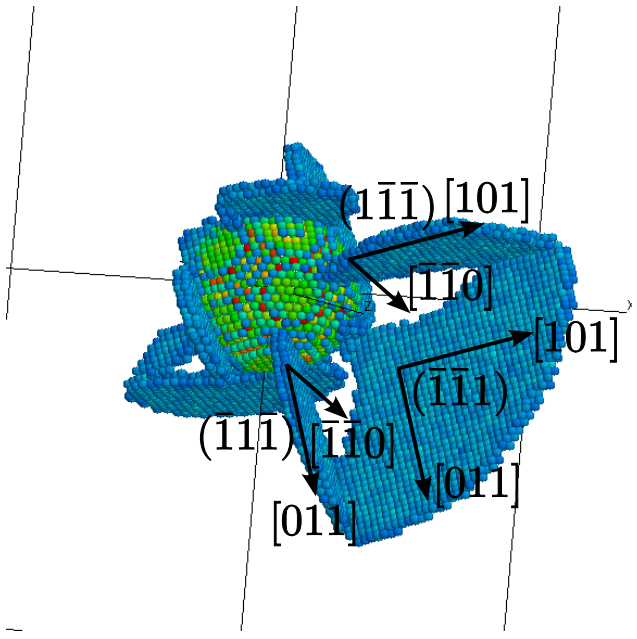
3.8.3 Interpretation

When we consider the plot of mean stresses of the three loading orientations, Figure 3.40(a), we see that the slopes during the elastic deformation are about the same and the yielding stress are not clear enough to distinguish the yield point. The plot of the von Mises stresses is an alternative plot that we can use to study the stress behaviors, Figure 3.40(b). It is much clearer that the crystal under the loading orientation of [110] and [111] yield at the earlier strain of about 0.035 compare to about 0.06 for the case of [001] loading orientation. However the yielding stress is the highest in the case of [111] loading orientation, at 4.75 GPa, and this indicates that the copper crystal is stronger in the direction of [111]. The softer direction is the [001] orientation with the yielding stress of 2.8 GPa.

We also notice the sharp drop of the von Mises stress after it passes the yield point consistently for all loading orientations. Since von Mises is a representation of shear stress in the system, we can say that the shear stress accumulated in the system by the applied uniaxial uniform expansion was relaxed by the activity of the shear loop dislocation emission from the void surface. It should be noted here that the stress values passed the yield point can be useful up to the start of hardening because the hardening is a result of the collision between dislocations (passing through the boundary) and their periodic images.

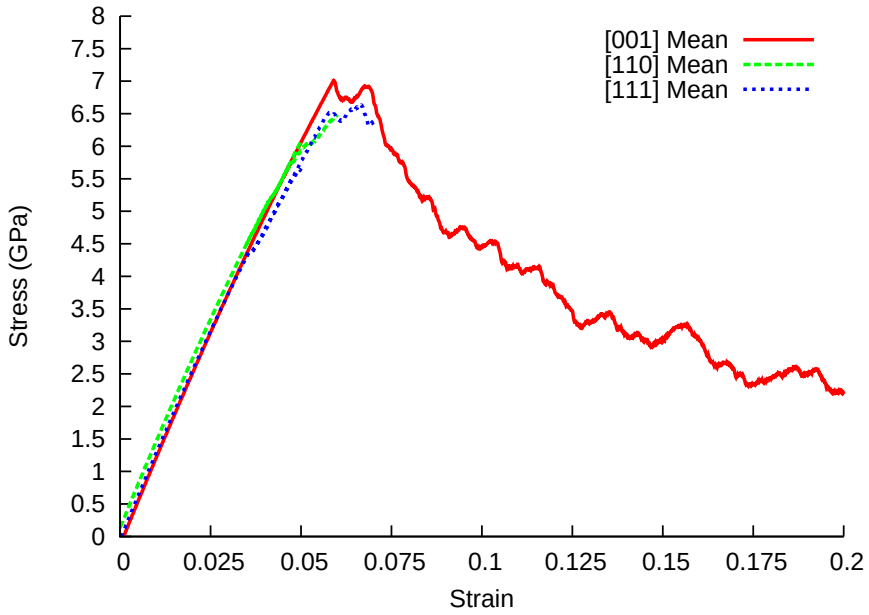


(a)

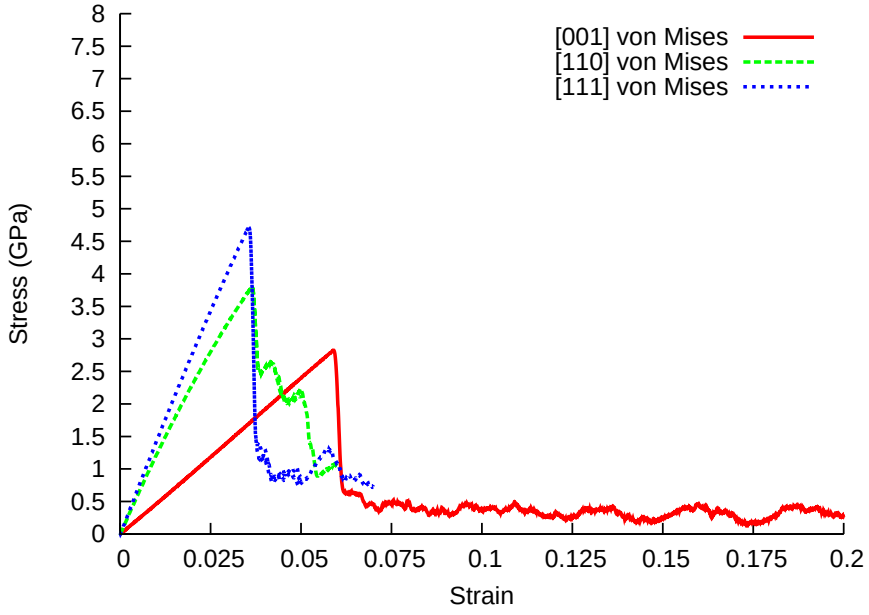


(b)

Figure 3.39: Plane and directions labeled for [111] loading direction; (a), schematic illustration and (b), MD simulation.



(a) Mean stress



(b) von Mises stress

Figure 3.40: Plot of stresses against strain for different loading orientations for R = 2 nm.

3.9 Acknowledgement

Some figures and materials in sections 3.2 and 3.4 to 3.7 also appear in S. Traiviratana, E. M. Bringa, D. J. Benson, and M. A. Meyers. "Void growth in metals: Atomistic calculations." *Acta Materialia*, vol 56: page 3874-3886, 2008.

Some figures and materials in sections 3.8 are in preparation for publication.

4

Algorithm for dislocation information extraction

4.1 Motivation

As it is natural for Molecular Dynamics calculations, each atom is considered individually and has at least 6 properties, namely a location vector ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ coordinate) and a velocity vector (in $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ components). The velocity vectors can be used to calculate kinetic energy and temperature. When we combine these basic parameters of a group of atoms in a given volume, we can calculate their pressure, stress components and so forth. These values can be easily output from LAMMPS during the run time as often as needed. This can result in an extremely large amount of information. To retrieve the plane, the Burgers vector, the velocity and the length information of an individual dislocation, one could use a rendering application to visualize the organization of atoms in the crystal.

Our results from LAMMPS Molecular Dynamics simulations have shown that one can see dislocation activity as a result of the material dynamics. With the centrosymmetry parameter [90], one can study how dislocations propagate or travel within material. To take this further, one can filter out, using the centrosymmetry parameter, the particles that are still in their perfect crystal configuration, leaving

only the particles in the dislocations. In general, particles which belong to stacking faults align themselves in a way that one can notice as two contiguous planes or disks.

4.2 Ideas

Once we observe atoms in dislocations, we need to separate them into groups of particles which belong only to a particular dislocation plane. Visually, we can define the leading and trailing edges for each dislocation. The normal to the plane can then be approximated. The length of leading and trailing edges can be measured or calculated to obtain the dislocation density. All of these can be done by hand calculation in principle, measuring and counting, but since we have a large amount of data from different simulations, hand calculations can only be done in a very limited ranges of simple simulations, in practice.

To be able to understand the interactions of dislocations in large numbers, we must take an advantage of the rich source information we have in MD calculations and applied statistical analysis to that information. A large amount of information must be processed, interpreted and summarized into what we can actually use to adjust current continuum models or even constructing a new and better model.

4.3 Algorithm

The above ideas can be summarized into the procedure described below;

4.3.1 Filtering out non-defect particles

With the help of centrosymmetry parameter [90], one can filter out particles that are in perfect FCC configuration, leaving only defects such as dislocations, stacking faults, grain boundaries and void surfaces, as seen in Figure 4.1.

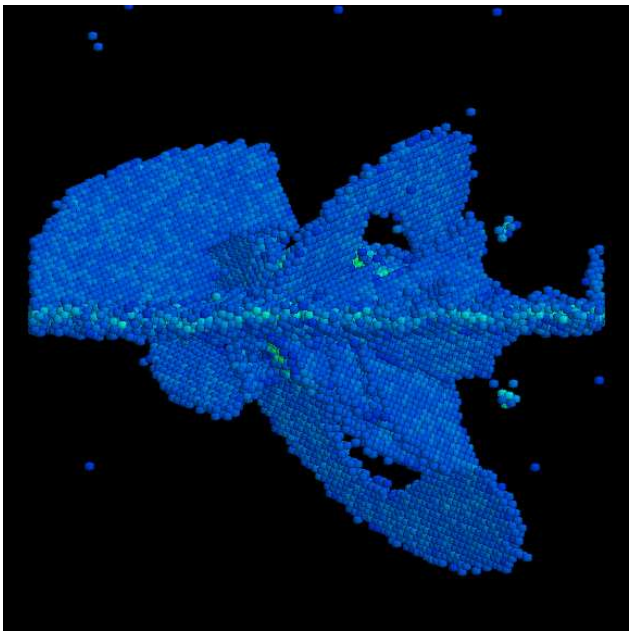


Figure 4.1: Atom rendered with a centrosymmetry parameter filter showing dislocation planes and grain boundary.

4.3.2 Search boxes generation

The simulation domain is divided into smaller boxes which various number of particles. Each box will have a list of particles that reside within the box. To conserve memory, each box will be represented by a linked-list (C programming and dynamic dimensioning, see Appendix B) containing only particle IDs. This is the preferred approach because each box can contain different number of particles, so a fixed-size array of integers would waste too much memory. Members of each search box are found by looping though all of the particles and comparing their location vector to check whether it is in a given box or not.

In one dimension, the box number n is $\mathrm{Int}(x/\Delta x)+1$, where x is the coordinate of the particle, and Δx is the box width. Spatial bucket sorting was pioneered for finite element contact by Benson and Hallquist [115], and it was found to be equally efficient for searching for dislocations.

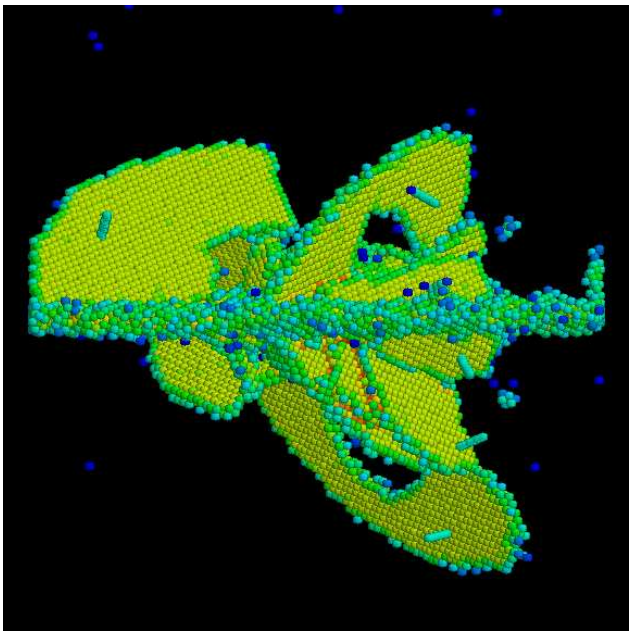


Figure 4.2: Atom colored by number of neighbors showing outward normal vector.

4.3.3 Neighbor particles search

Once we have the lists of particle numbers that belong to the search boxes, we search within each box for particles that are close to each other($r < 1.1(2r_0)$, where r_0 is the copper atom radius) and count them as neighbors. The same comparison is also done for neighboring boxes. This is done because particles along the edge of the search box may have neighbors within neighboring boxes.

4.3.4 Filter for the Dislocation Planes

As the search for neighbors is run, the number of neighbors of a given particles is counted and stored. Vectors that point from neighbor particles to the main particle are added up to construct a normal for this particle. Since it is an artifact of centrosymmetry parameter filtering that exposes dislocation planes with two adjacent planes of atoms, vectors from neighbor atoms on the same plane will cancel each other, while the vectors from neighbor atoms on the adjacent plane

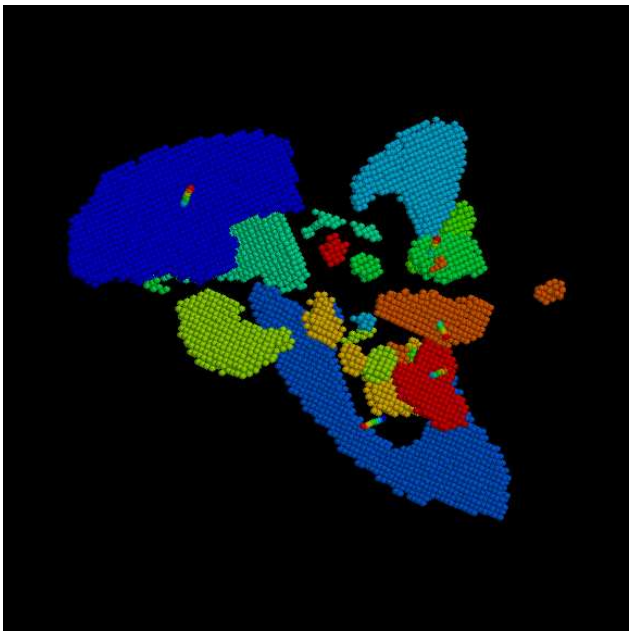


Figure 4.3: Separated dislocation planes and average plane normals, each with a distinct color.

point toward the normal to the dislocation plane. If the particle is a member of a dislocation plane, its normal will be close to parallel to the normal from neighbor particles that also belong to the same dislocation plane, see Figure 4.2. There are many types of data we can gather while we are searching for neighbor particle:

- The number of neighbors.
- The list that contains number of neighbors of neighbors.
- The average number of neighbors from list above.
- The standard deviation of the list above.
- The "normal" vector of an atom from adding up vectors pointing from neighbors.
- The Average distance between neighbors.

There are many ways to use the above parameters and combinations of them to search for particles that belong to a dislocation plane, and at the same time,

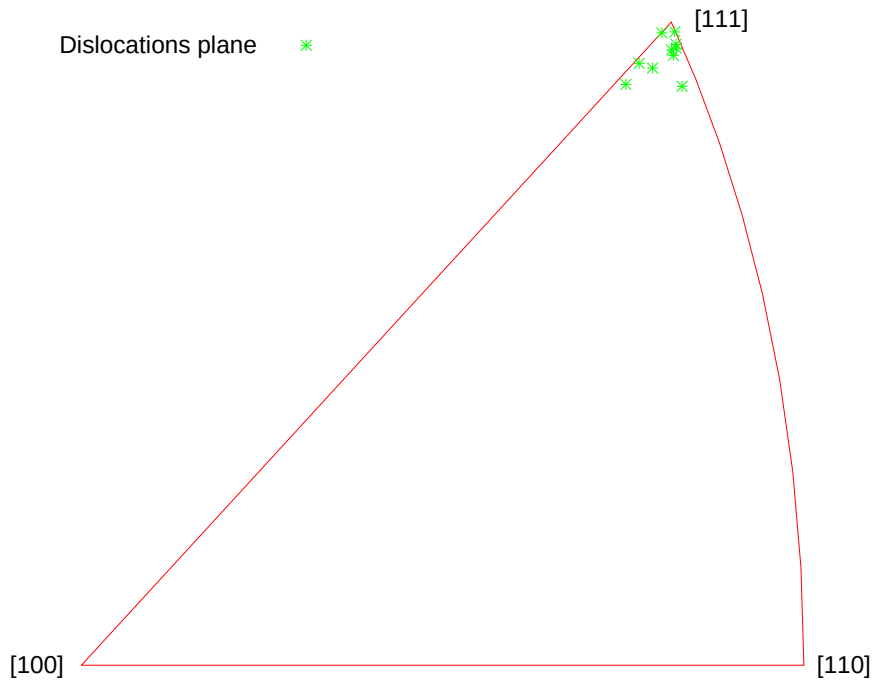


Figure 4.4: Stereographic triangle showing normal directions of dislocation planes from a simulation of a void between two crystals of copper undergoes uniaxial expansion. See section 3.2, spherical void configuration.

separate them from particles belonging to different dislocation planes, see Figure 4.3.

4.3.5 Normal of plane of dislocation

As it is mentioned above, that normal of a particle came from adding up vectors that point from neighboring particles, when we include the information that shows that this group of particle belong to a dislocation plane, the approximate normal to the plane can be calculated by averaging the normals of all the member particles. One can then compare this normal with the reference crystal directions and then be able to represent the normal of a plane on a stereographic triangles, see Figure 4.4.

4.3.6 Leading and Trailing Edge of Dislocation

After the particles that belong to a plane are determined, and the normal to the plane is calculated, one can create two axes spanning the local **x**-**y** plane of dislocation. The of dislocation can be calculated by averaging all the location vectors of member particle. The final local coordinate system consists of an origin which is at the centroid of the plane, the local **x** and **y** axes on the plane and the normal to the plane as the local **z**-axis. The **x**-axis can be identified by drawing a line from the farthest atom on plane to the centroid. Once the local **x** and **z** axes are created the local **y** axis follows from the right hand rule. The atoms in the plane are calculated in the local coordinate system. The local **z** component is close to zero and not used. By using the local **x** and **y** components, together with the local **x** axis (that separates the leading edge from the trailing edge), the search for maximum and minimum **y** values within every small interval Δx along **x**-axis gives the location of the leading edge and trailing edges, respectively, see Figure 4.5.

4.3.7 The Length of Dislocation and Dislocation Density

The search for the leading edge and trailing edge also produces, by connecting points that belong to each edge, two lines that define the leading and trailing edges. The line of the two edges can be full of discontinuities and jumps. With this information in mind, a mapping function is needed. A least square fit can be used to smooth the edges. At the same time, one can calculate the length of each line, which can later be used to find dislocation density, using the following equation

$$\rho = \frac{L}{V} \tag{4.1}$$

where ρ is the dislocation density, L is the sum of the dislocation lengths and V is the volume of the material.

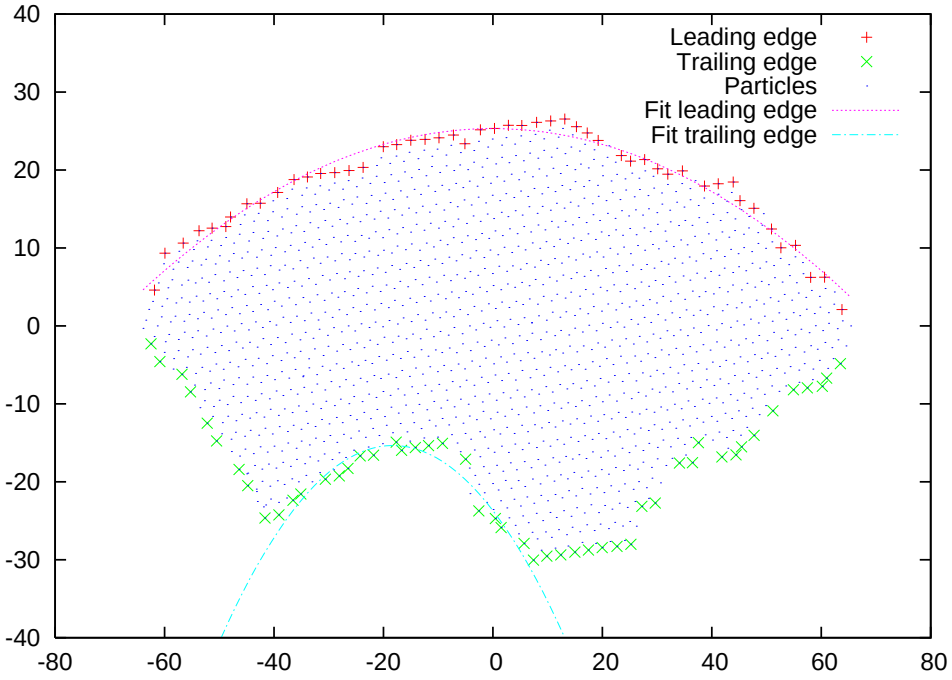


Figure 4.5: Leading and trailing edge extraction and curve fitting. Note: the centroid is at (0,0), the local **x**-axis goes from -65 to 65.

4.3.8 The Velocity of Dislocation

For each timestep, the locations of the leading and trailing edges can be calculated and smoothed. Comparing the locations from different timesteps allows us to calculate the dislocation velocity. Similarly, the acceleration (or deceleration) of a dislocation due to dislocation interactions can be calculated by differencing the velocities at two time steps.

4.4 Difficulties

The centrosymmetry parameter filter alone only roughly distinguishes between particles in dislocations and particles which are in a normal FCC crystal. This is the case, for example, when we have voids, surface particles, and grain boundary particles present in the simulation box. Those particles have range of CSP values from those of normal FCC particle to particles on dislocations and beyond. Once one sets a filter value for the CSP for the dislocation particles, the void surface particles and the grain boundary particles will also appear as well. To get a more accurate filter for dislocation particles, one must come up with a more complex combination of filters.

To by-pass this filter-centric idea and move on to working with an already filtered array of dislocation particles, I have hand picked some particles that are seen on planes of dislocations and start from those to find the rest of the member particles on the dislocation planes. Once that was done, the calculation of the planes, the normal directions of the planes, the leading edges and trailing edges were subsequently computed with much less complexity. Completing work on this algorithm has been left for future research.

5

Future Work

5.1 Algorithms for Dislocation Information Extraction

The algorithms for dislocation detection described in the previous chapter have to be further developed to be able to extract accurate information about defects. These algorithms should be able to give useful information such as dislocation length and velocity, dislocation interaction, and annihilation. Statistical analysis will help relate dislocation activity with stress relaxation and concentration, the yielding and the strength of the material. With the above mentioned development, information from MD simulations can be transfromed into information that can be compared with continuum analysis.

5.2 Atomistic Model for Void Growth Compared to Current Continuum Models

With a robust, accurate dislocation extraction algorithm, yield drop and hardening extracted from the stress-strain results could be directly linked with how dislocations move and interact. The probability of such interactions could be stud-

ied with statistical analysis, ultimately resulting in equations governing the yield process of the material. Multiple voids simulations can be performed in order to study the mechanisms of void growth and coalescence. As Gurson's model [1] and several modified ones [22, 23, 24, 25, 28, 29, 30] have predicted how voids should influence the strength of material, direct results from MD simulations will be able to test and refine the models. These results can then be used in continuum finite element codes.

5.3 Bicrystal simulations

Grain boundaries are considered one of the most interesting features in materials because they can be sources and sinks of dislocations. There can be a large number of configurations of grain boundaries between two crystals. Mathematically, there can be countless number of configuration, but, energetically, only a few ones are preferred. Several selected grain boundary configurations can be generated for tensile strain simulation. For instance, we can use tilt boundaries and show how the angle of tilt can influence dislocation interactions and the strength of the bi-crystals. Void can be created at the boundary to compare dislocation activities from void surface and grain boundary. With a dislocation detection algorithm, one could further analyze the dislocation interactions and obtain mechanisms which could translate into continuum understanding.

5.4 Molecular Dynamics Simulation of Nanocrystal Copper

Since the we can generate nanocrystal structures using the ideas provided in Chapter 2, nanocrystalline structures can be generated for a FCC material of choice with preset average grain sizes and random grain orientations. The size of the simulation domain can be as large as the computational capacity allows. It

would be interesting to see dislocation interactions and behaviors in nanocrystal microstructures with various grain sizes. We could then compare them with what has been postulated in Meyers et al. [3], regarding the strength of nanocrystalline materials. There are already many publications on MD simulations of nanocrystalline materials at different given sizes, for examples, Schiøtz and Jacobsen [99] and Kumar et al. [116]

5.4.1 Base Crystalline Structure, BCC, FCC

It would be interesting to see the behavior of dislocations on different basic crystalline structures, like BCC and FCC. The grain sizes and grain orientations can be fixed leaving only the basic crystalline structure as the variable. As this will be a comparison of two or more materials, the loading conditions must be the same with the amount of load normalized by the strength of each material.

5.4.2 Grain Size

Grain size is known to significantly affect the mechanical behavior of a material, especially its yield stress. It is well established in the micron grain size range that the yield stress, σ_y, varies with grain size, d, according to the Hall-Petch relation [4, 5].

$$\sigma_y = \sigma_0 + Kd^{\frac{1}{2}} \tag{5.1}$$

where σ_0 is the friction stress and K is a constant. When the grain size is reduced into the nanometer regime, the Hall-Petch relationship breaks down, as the pile-up of dislocation cannot be a hardening mechanism because the grains are too small. This leaves grain boundary sliding as a candidate for plastic deformation in nanocrystalline metals [3]. Various grain sizes can be generated while having same random grain orientation. MD simulations can be performed on different grain size samples, as it has been done in previous work, for example, Vo et al. [117].

5.4.3 Strain Rate

Strain rates are found to affect the yield stress in single crystals because they directly interfere with the dislocation velocity. When applying tension at large strain rates($\sim 10^{10}s^{-1}$), the single crystals deform elastically with the nucleation of voids. When the strain rate is reduced, the dislocation mechanisms take a leading role in the plastic deformation of the single crystal. It would be interesting to see the behavior of nanocrystals at different strain rates, as dislocations might not be the main mechanism in plastic deformation. Some preliminary results have been presented by Schiøtz and Jacobsen [99].

5.4.4 Voids in Polycrystalline Structure

Many attempts to study mechanical properties of nanocrystalline metals have found that the synthesized nanocrystalline structures contain large amounts of porosity or voids within the structure, most of them between grains [3]. These voids are sometimes thought to be the reason why the nanocrystalline structures lose their strength under tension [72, 95, 96]. These voids can be source of dislocations as well as weak spots for fracture within the material. A comparison of MD simulations with the same grain size and orientation, with voids and without voids, can provide insight into the influence of voids on the strength of nanocrystalline materials.

6

Conclusions

The most widely adopted continuum models of void growth have no size scale dependent input in their failure models and yield criterion, however, it was pointed out by several MD studies [61, 62, 114, 118] that size scale can indeed affect the yield strength of a material.

Several molecular dynamics simulations revealed previously unknown dislocation interaction mechanisms. Further analyses are necessary to fully understand all the mechanism of void growth at the atomistic level, and to be able to link these mechanisms to the continuum scale.

It has been shown that shear loops are the main mechanism for void growth under high strain rates, as opposed to the diffusion mechanisms proposed by others. The expansion of the loops and their cross slip leads to the severely work hardened layer surrounding a growing void. The size scale dependence of the void on the yield strength of material was confirmed. Calculations were also carried out for a void at the interface between two grains sharing a tilt boundary. The results show similar dislocation behaviors.

The growth of voids simulated by MD is compared with the Cocks-Ashby constitutive model and significant agreement is found. The density of geometrically-necessary dislocations as a function of void size is calculated based on the emission of shear loops and their outward propagation.

The details of the dislocation interactions, and the examples of bi-planar and tri-planar dislocation loop interactions were shown. These results should help improve future studies in void growth and coalescence in ductile materials.

A code was written in C employing Voronoi tessellation to construct nanocrystalline material microstructures for MD simulations. Implemented in this code is a method for generating random crystal orientations using Euler's parameters, which, produces equally distributed random direction on a unit sphere. This is an improvement on Euler's angles which tend to concentrate a large fraction of the random distribution near the **x**, **y** and **z** axes on the unit sphere. In addition, a fast and effective method for searching for atoms that have the same or similar locations, which can produce high repulsive forces, was implemented using the pointer and linked-list capability in C to further reduce storage and increase robustness.

A preliminary code that can be used to extract useful information from dislocation activities from molecular dynamics results was created and tested successfully, giving the initial steps for a post-processing code that can autonomously retrieve useful data from the vast amount of rich information produced by MD simulations.

7

Appendix A

This appendix describes molecular dynamics tools which I used for this study from pre-processing to post-processing. The purpose of including this information is to give some recommendations/guidance for future researchers who are going to use MD for solid materials. This will include the tutorial of LAMMPS [78] and tools I used for nanocrystal construction and to extract information from the computation. There may be other possible ways to execute and run LAMMPS simulations which I have not included in this section, but it is for the user to find out what is suited to each situation.

7.1 LAMMPS Input File

This section will go through line-by-line the LAMMPS instructions inside of a LAMMPS input file. The LAMMPS code is compatible with LAMMPS version 24Sept07. Since LAMMPS can be updated several times in a single year, the input code and its statements might changed. It is recommended that the user read the LAMMPS documentation (http://lammps.sandia.gov/doc/Manual.html) for further details and troubleshooting.

Listing 7.1: Relaxation input code

```
1  units           metal
2  boundary        p p p
3  atom_style      atomic
4  lattice         fcc 3.6150 origin 0 0 0 orient x 1 0 0 orient y 0 1 0
       orient z 0 0 1
5  region          box block -14 14 -14 14 -14 14
6  region          void sphere 0 0 0 10 units box
7  create_box      1 box
8  create_atoms    1
9  group           bulk region box
10 group           void region void
11 delete_atoms    group void
12 pair_style      eam/alloy
13 pair_coeff      * * cuMishinAlloy Cu
14 neighbor        2.0 bin
15 neigh_modify    every 1 delay 5 check yes
16 compute         new3d all temp
17 velocity        all create 300 14285736 temp new3d
18 thermo          1
19 thermo_style    custom step temp pe ke etotal press vol lx ly lz pxx
      pyy pzz pxy pxz pyz
20 thermo_modify   temp new3d norm yes
21 timestep        0.001
22 fix             1 all nve
23 dump            1 all custom 5 dump.relax1.* x y z centro vx vy vz
      sxx syy szz sxy sxz syz tag epair ke
24 restart         10 restart.relax1.*
25 run             10
```

The first line tells LAMMPS to use metal units, for example, length in Angstroms, time in picoseconds, mass in gram/mole, energy in eV, temperature in K and pressure in bars. The second line set the boundary of the simulation box to be parallel in all three dimensions, **x**, **y** and **z**. The third line sets the atom type to atom particle and not charged particle, or something else.

The 4th line describes how LAMMPS should generate the crystalline structure of atom particles. In this case, it is set to the FCC structure with the origin of the simulation box at $(x, y, z) = (0, 0, 0)$. The x-axis of the simulation is aligned with the crystal direction [100], while the **y**-axis and **z**-axis are aligned with [010] and [001], respectively.

The 5th line tells LAMMPS to make a region (named box) with block region type with each side contains 28 unit cells and to locate the center of simulation box at the global origin. The 6th line also creates a spherical region (named void) with the center at the origin and has the radius of 10 Angstroms (note the command units box). The 7th and 8th lines tell LAMMPS to generate the simulation domain and fill it with the atom of predefined particles.

The 9th and 10th lines name the two groups, bulk and void, from the two preset regions, box and void. This group naming facilitates the application of commands to the whole group, for example, the delete_atom command, in line 11th, operating on the group void which results in creating a spherical void where the region is defined.

The 12th and 13th lines define the type of potential of the created atoms, in this case, it is EAM alloy potential reading from the file cuMishinAlloy. Lines 14th and 15th tell LAMMPS to manage the neighbor partitioning. Line 16th defines the name of a computation, type Temperature, and name it new3d. The 17th assign the temperature of 300K with a random seed 14285736 to the preset computation name new3d, which is equal to assigning the kinetic energy to the system.

The 18th, 19th and 20th lines tell LAMMPS to output the information of step number, temperature, potential energy, kinetic energy, total energy, pressure, volume, simulation box sizes in **x**, **y** and **z**, pressure or stress of the whole simulation domain as components of stress tensor every step of the run into the log file. Timestep size is define in line 21st as 1 femtosecond(0.001 picosecond).

The 22nd line applies no loading constrain to the system and only allow the computation to progress with the constant volume and internal energy. This is the most basic instruction for a simulation. The dump command in line 23rd

tells LAMMPS to output the following information of every atom into the file dump.relax1 every 5 running steps;

- Location vector including the component **x**, **y** and **z**.
- Centrosymmetry parameter
- Velocities in **x**, **y** and **z** directions.
- Components of stress tensor
- Atom number
- Potential energy
- Kinetic energy

The 24th line tells LAMMPS to write a restart file into the name restart.relax1 for every 10 running steps. The last line tells LAMMPS to run the computation for 10 steps.

The example from Listing 7.1 only run for ten steps without applying any load to the simulation system. The Listing 7.2 shows the relaxation using "nph" ensemble, which is used to bring down the pressure of the simulation system to the 1 atm level.

Listing 7.2: NPH Relaxation input code

```
1  units           metal
2  boundary        p p p
3  atom_style      atomic
4  read_restart    restart.relax1.10
5  pair_style      eam/alloy
6  pair_coeff      * * cuMishinAlloy Cu
7  neighbor        2.0 bin
8  neigh_modify    every 1 delay 5 check yes
9  compute         new3d all temp
10 thermo          20
11 thermo_style    custom step temp pe ke etotal press vol lx ly lz pxx
       pyy pzz pxy pxz pyz
12 thermo_modify   temp new3d norm yes
```

```
13 timestep          0.001
14 reset_timestep    0
15 fix               1 all nph aniso 4712.2526 0.0 4723.4878 0.0 4721.2104
       0.0 0.3
16 dump              1 all custom 1000 dump.relax3.* x y z centro vx vy vz
       sxx syy szz sxy sxz syz tag epair
17 restart           1000 restart.relax3.*
18 run               2000
```

Most of the command instructions in Listing 7.2 are similar to ones from Listing 7.1 with the differences in the following lines. The 4th line tells LAMMPS to read a restart file from the file restart.relax1.10 which is created from the first run, Listing 7.1. The frequency of outputting the thermo information in the log file is changed to every 20 running steps as shown in line 10th. The 13th line resets the timestep number to zero. The dump and restart instructions also change the frequency of writing dump and restart files to every 1000 running steps, as seen in lines 15th and 16th. This relaxation runs for 2000 timesteps.

The most important instruction is in the 15th line. It tells LAMMPS to update the position and velocities for each timestep with Nose/Hoover pressure barostat. This will bring the pressure from each **x**, **y** and **z** directions to the zero level. The numbers 4712.2526, 4723.4878 and 4721.2104 are from the pressure pxx, pyy, and pzz read from the last timestep within the log file of the previous run.

7.2 Running LAMMPS

This section will give an example of a running script (input file) for LAMMPS in case of the uniaxial expansion in **z** direction. See Listing 7.3.

Listing 7.3: Uniaxial expansion input code

```
1  units            metal
2  boundary         p p p
3  atom_style       atomic
4  read_restart     restart.relax3.2000
5  pair_style       eam/alloy
6  pair_coeff       * * cuMishinAlloy Cu
7  neighbor         2.0 bin
8  neigh_modify     every 1 delay 5 check yes
9  compute          new3d all temp
10 thermo           500
11 thermo_style     custom step temp pe ke etotal press vol lx ly lz pxx
       pyy pzz pxy pxz pyz
12 thermo_modify    temp new3d norm yes
13 timestep         0.001
14 reset_timestep   0
15 fix              1 all nve
16 fix              2 all volume/rescale 1 z -106.4784 106.4784
17 dump             1 all custom 5000 dump.expand1.* x y z centro vx vy
       vz sxx syy szz sxy sxz syz tag epair
18 restart          50000 restart.expand1.*
19 run              500000
```

The most of the command instructions are similar to the Listing 7.2, except for line 16th. It tells LAMMPS to rescale the simulation box in **z** direction for every running step until it reaches the specified size (in this example from -106.4784 to 106.4784), which is 5% larger than the original size (can be found in the log file from the previous run in the lz column). This input file tells LAMMPS to run for 500000 timesteps, which will give the strain rate of 10^8 s^{-1}.

$$\dot{\varepsilon} = \frac{\varepsilon}{t} = \frac{\varepsilon}{N\Delta t} = \frac{0.05}{(500000)(0.001 \times 10^{-12} s)} = 10^8 s^{-1} \tag{7.1}$$

For some computational systems, it takes a long time to complete a full 2 million timesteps for a 20% of strain. We can take an advantage of restart file by

separate the whole simulation into 4 parts. This 500000 timesteps is equal to a half of a million timesteps or a quarter of an entire simulation.

7.3 Nanocrystalline Structure Simulation

The code for the construction of nanocrystalline structures is provided in appendix B. It is employed to generate nanocrystalline copper samples as LAMMPS data files which can be included into the LAMMPS input file. The following listing shows how the data file can be imported in to LAMMPS and the special relaxation scheme for nanocrystal structure.

Listing 7.4: Relaxation of nanocrystal copper input code

```
1  units           metal
2  boundary        p p p
3  atom_style      atomic
4  pair_style      eam/alloy
5  read_data       Nc5datNi-1.dat
6  pair_coeff      * * niyuri.eamalloy Ni
7  mass            1 58.693
8  neighbor        2.0 bin
9  neigh_modify    every 1 delay 5 check yes
10 compute         new3d all temp
11 velocity        all create 0.01 5812775 temp new3d
12 fix             1 all nve
13 fix             2 all temp/rescale 7 0.01 0.01 0.001 1.
14 thermo          100
15 thermo_style    custom step temp pe ke etotal press vol lx ly lz pxx
       pyy pzz pxy pxz pyz
16 thermo_modify   temp new3d norm yes
17 timestep        0.0001
18 dump            1 all custom 2500 dump.relax.* x y z centro
19 run             2500
20 unfix           2
21 velocity        all create 300 5812775 temp new3d
```

```
22 fix              3 all temp/rescale 7 300 300 0.1 1
23 thermo           100
24 thermo_modify    temp new3d norm yes
25 timestep         0.0005
26 run              2500
27 unfix            3
28 velocity         all create 5 5812775 temp new3d
29 fix              4  all temp/rescale 7 5 5 0.1 1
30 thermo           100
31 thermo_modify    temp new3d norm yes
32 timestep         0.001
33 run              2500
34 unfix            4
35 thermo           100
36 thermo_modify    temp new3d norm yes
37 timestep         0.001
38 restart          10000 relax.restart
39 run              2500
```

This relaxation scheme (lines 11th to 39th) is described in detail in section 3.3. The import of the generated nanocrystal structure is in line 5th. The resulting nanocrystal structure is output as a restart file in line 38th. The dump command in this input file is optional.

7.4 LAMMPS Dump File Processing

The dump files from LAMMPS can be customised to output several desired data from the simulation. The most frequency used style in this work can be seen in Listing 7.3 at the 17th line. This dump style gives the LAMMPS dump file which is partially shown in Listing 7.5.

Listing 7.5: Part of a dump file

```
1  ITEM: TIMESTEP
2  0
3  ITEM: NUMBER OF ATOMS
4  699565
5  ITEM: BOX BOUNDS
6  -101.411 101.411
7  -101.413 101.413
8  -105.971 105.971
9  ITEM: ATOMS
10 -65.2733 -86.8902 -52.9935 0.185195 1.29046 -1.08892 0.823471 726609
       1.08786e+06 1.01682e+06 233523 -222733 -113605 22217 -3.5146
11 -86.9162 -83.334 -53.0245 0.182712 1.47628 -2.17631 -1.88452 790687
       900497 939671 134009 -130235 42192 22249 -3.53088
12 -97.8432 -99.6685 -104.113 0.172806 -1.36554 0.177734 -2.02157 562354
        839236 826231 -26384.6 135577 161043 8 -3.51977
13 -101.344 -77.8583 -73.8101 0.207106 -0.103727 -1.31566 1.08847 592293
        130014 679869 -161252 -178117 110780 12884 -3.49943
14 -94.2013 -99.5969 -104.213 0.37063 1.41066 1.5959 1.86263 788915
       634517 1.09649e+06 23762.3 -87344.9 215950 12 -3.48832
```

There are 699565 atoms in this simulation and there will be the same number of lines following the key word "ITEM: ATOMS" listing the parameter of each atom. This dump file can be large and if we tell LAMMPS to write output dump file frequently, we can have a large number of dump files. Processing these files one-by-one is a very time consuming task. A script which I use to transform a dump file into a Rasmol compatible (xyz) format is in Listing 7.6.

Listing 7.6: A bash shell script for the transformation of dump file into an xyz file

```
1 #!/bin/bash
2 if [ $# -lt 2 ]
3 then
4   echo "Usage␣$0␣dump.file␣column"
5   exit 0
6 fi
```

```
7  timestep=`head -2 $1 | tail -1`
8  step=`printf "%07d" $timestep`
9  echo -n $step
10 tmp1=.tmp1
11 tmp2=.tmp2
12 target="step_"$step"_C"$2".xyz"
13 echo "{print \"Cu\",\$1,\$2,\$3,$"$2"}" > $tmp1
14 csplit -s $1 10 && echo -n "_csplit"
15 awk -f $tmp1 xx01 > $tmp2 && echo -n "_awk"
16 wc -l $tmp2 | cut -d " " -f1 > $target && echo -n "_wc"
17 echo "Cu" >> $target
18 cat $tmp2 >> $target && echo "_cat = done!"
19 rm $tmp1 $tmp2 xx00 xx01
```

The resulting xyz file can be read by atomic rendering application called "Rasmol" [76].

8

Appendix B

The code for construction of nanocrystal copper will be shown here. The requirement is that you have qhull application install in the system. There are five listing showing the file needed to compile the code.

Listing 8.1: Makefile

```
1  CC = gcc
2  FL = -g
3  LB = -lm
4  OJ = -c
5  rev := $(shell date +%y%m%d)
6
7  all: fillatoms
8
9  fillatoms: fillatoms.c functions.o
10         ${CC} ${FL} ${LB} -o fillatoms fillatoms.c functions.o
11
12 functions.o: functions.c
13         ${CC} ${FL} ${OJ} functions.c
14
15 clean:
16         rm -f fillatoms *.o
```

Listing 8.2: fillatom.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5
6  #include "fillatoms.h"
7  #include "global.h"
8
9  int main(int *argc, char **argv)
10 {
11    int i,j,k,l,m,n,a,nop,noj,nos,nof,nopoly,noppp,noppf,extra,pfcatom;
12    double x,y,z,boxvolume,unitcellvolume;
13    double r0 = 0.25*sqrt(2.0*lattice*lattice);
14    double penalty,tol;
15    //double tol = 0.0; //FIXME this will not check for interference
16    double majorlat = 2.0*sqrt(2.0)*r0;
17    double minorlat = 0.5*majorlat;
18    double pi=4.0*atan2(1.0,1.0);
19    double offset[3] = {-1.0,0.0,1.0};
20    Vector r,*p,*site;
21    int *junction,*sitecenter;
22    Orientation Or;
23    Polyhedron *ph;
24    Face *f;
25    FILE *fi,*fj;
26    int *filledlist,fill=0,bno=1;
27    time_t start_time;
28    unsigned int seed;
29
30 //  srand(time(0));
31    srand(14285736);
32
33    printf("Start reading input files.\n");
34
```

```
35    fj = fopen("box","r");
36    fscanf(fj,"%lf %lf %lf %lf %lf %lf",&gmx,&gmy,&gmz,&px,&py,&pz);
37    fscanf(fj,"%d %lf %d",&nos,&penalty,&seed);
38    fclose(fj);
39    if (seed==0) {srand(time(0));printf("Time seed\n");}
40    else if (seed==1) {srand(1);printf("Default seed\n");}
41    else {srand(seed);printf("User seed\n");}
42
43    site = (Vector *)malloc(nos*sizeof(Vector));
44    printf("  X: %8.2f %8.2f\n  Y: %8.2f %8.2f\n  Z: %8.2f %8.2f\n",gmx
          ,px,gmy,py,gmz,pz);
45    printf("%d grains with ",nos);
46    tol = 2*r0 - penalty * (2*r0);
47
48    fi = fopen("siteq","w");
49    fj = fopen("site","w");
50    fprintf(fi,"3\n");
51    fprintf(fi,"%d\n",nos*27);
52
53    for (i=0;i<nos;i++) {
54      site[i].x = RandomWithIn(gmx,px);
55      site[i].y = RandomWithIn(gmy,py);
56      site[i].z = RandomWithIn(gmz,pz);
57    }
58
59    px = px - gmx; py = py - gmy; pz = pz - gmz;
60    boxvolume = px*py*pz;
61    unitcellvolume = majorlat*majorlat*majorlat;
62    printf("grain size = %f Angstrom.\n",
63         2.0*pow((3.0*boxvolume/4.0/(double)nos/pi),(1.0/3.0)));
64    printf("Allow %f of 2*radius distance (%f Angstrom)\n",penalty,tol)
          ;
65
66    for (i=0;i<3;i++) {
67      for (j=0;j<3;j++) {
```

```
68        for (k=0;k<3;k++) {
69          for (n=0;n<nos;n++) {
70            fprintf(fi,"%15.10f %15.10f %15.10f\n",
71                site[n].x+offset[i]*px,site[n].y+offset[j]*py,site[n].z
                  +offset[k]*pz);
72            fprintf(fj,"%15.10f %15.10f %15.10f\n",
73                site[n].x+offset[i]*px,site[n].y+offset[j]*py,site[n].z
                  +offset[k]*pz);
74          }
75        }
76      }
77    }
78    fclose(fi);fclose(fj);
79    system("cat siteq | qhull v o s Fv TO input");
80
81    fi = fopen("input","r");
82    fj = fopen("site","r");
83    fscanf(fi,"%d %d %d %d",&a,&nop,&nopoly,&extra);extra=0; //FIXME
        extra=verbose
84    if (extra) printf("Nop     :%d\n",nop);
85
86    p = (Vector *)malloc(nop*sizeof(Vector));
87    ph = (Polyhedron *)malloc(nopoly*sizeof(Polyhedron));
88    site = (Vector *)malloc(nopoly*sizeof(Vector));
89    junction = (int *)malloc(nop*sizeof(int));
90    sitecenter = (int *)malloc(nopoly*sizeof(int));
91
92    nos = 0;
93    for (i=0;i<nopoly;i++) {
94      fscanf(fj,"%lf %lf %lf",&x,&y,&z);
95      site[i].x = x;
96      site[i].y = y;
97      site[i].z = z;
98      if (inthebox(&site[i])) { sitecenter[nos] = i; nos++; }
99    }
```

```
100   fclose(fj);
101
102   maxatom = (int)(boxvolume / unitcellvolume * 4.01);
103   pfcatom = (int)(boxvolume / unitcellvolume * 4.0);
104   atom = (Particle *)malloc(maxatom*sizeof(Particle));
105   printf("Allocated %d particles (perfect crystal %d particles).\n",
          maxatom,pfcatom);
106
107   noj = 0;
108   for (i=0;i<nop;i++) {
109     fscanf(fi,"%lf %lf %lf",&x,&y,&z);
110     p[i].x = x;
111     p[i].y = y;
112     p[i].z = z;
113     if (inthebox(&p[i])) { junction[noj] = i; noj++; }
114     if (extra)
115       printf("P:%d %14.10f %14.10f %14.10f\n",i,p[i].x,p[i].y,p[i].z)
              ;
116   }
117
118   if (extra) printf("Nopoly   :%d\n",nopoly);
119
120   for (j=0;j<nopoly;j++) {
121     ph[j].blank = 1;
122     ph[j].p0 = &p[0];
123     fscanf(fi,"%d",&noppp);
124     if (extra) printf("Noppp    :%d\n",noppp);
125     ph[j].nop = noppp;
126     for (i=0;i<noppp;i++) {
127       fscanf(fi,"%d",&a);
128       ph[j].p[i] = a; if (extra) printf("%d ",a);
129     }
130   }
131
132   fscanf(fi,"%d",&nof);
```

```
133   f = (Face *)malloc(nof*sizeof(Face));
134   for (i=0;i<nof;i++) {
135     if (extra) printf("Face:%d\n",i);
136     fscanf(fi,"%d",&noppf);if (extra) printf("%d ",noppf);
137     f[i].noppf = noppf;
138     for (k=0;k<noppf;k++) {
139       fscanf(fi,"%d",&a);
140       if (extra) printf("%d ",a);
141       f[i].p[k] = a;
142     }
143     if (extra) printf("\n");
144   }
145   if (extra) printf("\n");
146   fclose(fi);
147
148   printf("\nFinish reading, counting neighbors of each polyhedron.\n"
        );
149
150   for (j=0;j<nopoly;j++) {
151     ph[j].nof = 0;
152     ph[j].com = site[j];
153     for (i=0;i<nof;i++) {
154       if ((f[i].p[0]==j)||(f[i].p[1]==j)) {
155         if (extra) {
156           printf("Polyhedron:%d has Face:%d -> ",j,i);
157           for (k=0;k<f[i].noppf;k++) printf("%d ",f[i].p[k]);
158           printf("\n");
159         }
160         if (f[i].p[0]==j) {
161           r = va2b(&site[f[i].p[0]],&site[f[i].p[1]]);
162           ph[j].n[ph[j].nof] = unit(&r);
163           ph[j].ngh[ph[j].nof] = f[i].p[1];
164         }
165         else {
166           r = va2b(&site[f[i].p[1]],&site[f[i].p[0]]);
```

```
167             ph[j].n[ph[j].nof] = unit(&r);
168             ph[j].ngh[ph[j].nof] = f[i].p[0];
169           }
170           ph[j].c[ph[j].nof] = center2(&site[f[i].p[1]],&site[f[i].p
                  [0]]);
171           ph[j].nof++;
172         }
173       }
174       if (extra) printf("Nof:%d \n",ph[j].nof);
175     }
176
177     printf("\nSearch Box generating.\n");
178
179     searchbox = (struct node**)malloc(nopoly*sizeof(struct node*));
180     gb = (struct node**)malloc(nopoly*sizeof(struct node*));
181     printf("Allocated %d searchboxes(polyhedron).\n",nopoly);
182     for (j=0;j<nopoly;j++) searchbox[j] = gb[j] = NULL;
183
184
185     printf("\nStart filling up polyhedrons.\n");
186
187     ano=0;
188     filledlist = (int *)malloc(nopoly*sizeof(int));
189
190     time(&start_time);
191     for (i=0;i<nopoly;i++) {
192       if ((inthebox(&site[i]))&&(ph[i].blank)) {
193         Or = MakeOrientation(0);
194         i = FillAt(&ph[0],nopoly,i,&Or,&ph[i].com);
195         for (j=0;j<ph[i].pr_count;j++) {
196           k = FillAt(&ph[0],nopoly,0,&Or,&ph[i].periodic[j]);
197         }
198         printpercent(start_time,100.*(double)ano/(double)maxatom);
199       }
200     }
```

```
201     fill = UpdateFilledList(&ph[0],nopoly,&filledlist[0]);
202     printf("Filled %d particles (%d different)\n",ano,ano-pfcatom);
203
204
205     printf("\nFilter atoms, too close within neighbors, tol = %f\n",tol
            );
206
207     time(&start_time);n = 0;
208     for (i=0;i<fill;i++) {
209       for (j=0;j<ph[filledlist[i]].nof;j++) {
210         a = Search2Boxes(filledlist[i],ph[filledlist[i]].ngh[j],tol);
211         n = n + a;
212       }
213     }
214     printf("Total %d to be delete\n",n);
215     printpercent(start_time,100.0);
216
217
218     printf("\nPrint out atoms into files.\n");
219
220     time(&start_time);a = 1;
221     for (i=0;i<fill;i++) {
222       a = printbox(filledlist[i],a);
223       if (((i+1)*100/fill) % 10 == 0)
224         printpercent(start_time,100.*(double)(i+1)/(double)fill);
225     }
226
227     free(atom);          atom = NULL;
228     free(searchbox);     searchbox = NULL;
229     free(gb);            gb = NULL;
230
231     return 0;
232 }
```

Listing 8.3: functions.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5
6  #include "fillatoms.h"
7  #include "global.h"
8
9  int UpdateFilledList(Polyhedron *ph,int nopoly,int *filledlist)
10 {
11   int i,fill=0;
12
13   for (i=0;i<nopoly;i++) {
14     if (!(ph->blank)) {
15       *(filledlist+fill) = i;
16       fill++;
17     }
18     ph++;
19   }
20   return fill;
21 }
22
23 int NotInThisList(int *list,int size,int check)
24 {
25   int i,go=0;
26
27   for (i=0;i<size;i++) {
28     if (check == *list) go++;
29     list++;
30   }
31   if (go == 0) return 1;
32   else return 0;
33 }
34
```

```
35 int PolyhedronHasThis(Vector *r,Polyhedron *ph,int nopoly)
36 {
37   int i,j;
38
39   for (i=0;i<nopoly;i++) {
40     if (inpolyhedron(r,ph) < 1) {
41       j = i;
42       break;
43     }
44     ph++;
45   }
46   return j;
47 }
48
49 Vector latsize(Vector *a,Vector *r,int mm)
50 {
51   double r0=0.25*sqrt(2.0*lattice*lattice);
52   double majorlat=2.0*sqrt(2.0)*r0;
53   Vector v = *r;
54
55   if (mm == 0) {
56     while (v.x > a->x) {
57       v.x -= majorlat;
58     }
59     while (v.y > a->y) {
60       v.y -= majorlat;
61     }
62     while (v.z > a->z) {
63       v.z -= majorlat;
64     }
65   } else if (mm == 1) {
66     while (v.x < a->x) {
67       v.x += majorlat;
68     }
69     while (v.y < a->y) {
```

```
70        v.y += majorlat;
71      }
72      while (v.z < a->z) {
73        v.z += majorlat;
74      }
75    }
76    return v;
77  }
78
79  int SaveAtom(Vector *rt,int j,int or,int noa,int tno)
80  {
81    atom[ano].r = *rt;
82    atom[ano].type = tno+2; //GB = 1, internal = 2
83    atom[ano].ph = j;
84    atom[ano].or = or;
85
86    Push(&searchbox[j],ano);
87    if (tno==-1) Push(&gb[j],ano);
88
89    ano++;
90    noa++;
91
92    return noa;
93  }
94
95  void bound(Vector *min,Vector *max, Vector *p)
96  {
97    min->x = min(min->x,p->x);
98    min->y = min(min->y,p->y);
99    min->z = min(min->z,p->z);
100   max->x = max(max->x,p->x);
101   max->y = max(max->y,p->y);
102   max->z = max(max->z,p->z);
103   return;
104 }
```

```
105
106 int FindNeighbor(Polyhedron *ph,int nopoly,int j,int *tobefilledlist,
        int tobefilled)
107 {
108   Polyhedron *phj, *phk;
109   int i,k,count;
110   Vector *r;
111
112   phj=ph+j;
113
114   //printf("Polyhedron %d of %d with %d faces %d points\n",j,nopoly,
        phj->nof,phj->nop);
115   //printf("Neighbor:\n");
116   for (i=0;i<phj->nof;i++) {
117     phk=ph + phj->ngh[i];
118     //printf("%d contain %d points ",phj->ngh[i],phk->nop);
119     count=0;
120     for (k=0;k<phk->nop;k++) {
121       r=phk->p0 + phk->p[k]; // r is the point that bound this
            polyhedron
122       //printvector(k,r);
123       if (inthebox(r)) {
124         count++;
125       }
126     }
127     if (count>0) {
128       //printf("In The Box %d / %d\n",count,phk->nop);
129       if (phk->blank) {
130         *(tobefilledlist + tobefilled) = phj->ngh[i];
131         tobefilled++;
132       }
133     } else {
134
135       //printf("\n");
136     }
```

```
137     }
138
139
140     return tobefilled;
141 }
142
143 int FillAt(Polyhedron *ph,int nopoly,int j,Orientation *or,Vector *r)
144 {
145     Polyhedron box,*phh,*phk,*phi;
146     Vector xyz,tmp,rt,rr,pr,*p0,p1,dummy;
147     int noa=0,tno;
148     double *rotM = &(or->rotM[0][0]);
149     double r0 = 0.25*sqrt(2.0*lattice*lattice);
150     double majorlat = 2.0*sqrt(2.0)*r0;
151     double minorlat = 0.5*majorlat;
152     int i,k,extend[8],ext=0,inph;
153
154     phk = ph;
155     rr = rotation(*r,rotM);
156
157     if (j==0) { //Search for polyhedron[j] to fill
158       for (i=0;i<nopoly;i++) {
159         if (inpolyhedron(r,ph) < 1) {
160           j = i;
161           phh = ph;
162           break;
163         }
164         ph++;
165       }
166     } else { //Already know which to fill
167       phh = ph + j;
168     }
169
170     if (phh->blank) { //If it's blank then fill
171       phh->start = ano;
```

```
172     p0 = phh->p0;
173     for (i=0;i<phh->nop;i++) { //Figuring out bound of a polyhedron
174       p1 = rotation(*(p0+phh->p[i]),rotM);
175       bound(&(phh->min),&(phh->max),&p1);
176     }
177     box = MakeBox(rotM);
178     bound(&dummy,&(phh->min),&box.min);
179     bound(&(phh->max),&dummy,&box.max);
180     phh->min = latsize(&(phh->min),&rr,0); //Make bound multiple of
181     phh->max = latsize(&(phh->max),&rr,1); //Lattice size
182
183     xyz.z = phh->min.z;
184     while (xyz.z <= phh->max.z) {
185       xyz.y = phh->min.y;
186       while (xyz.y <= phh->max.y) {
187         xyz.x = phh->min.x;
188         while (xyz.x <= phh->max.x) {
189
190           rt = rotationT(xyz,rotM);
191           inph = inpolyhedron(&rt,phh);
192           if (inph < 1) { // if in this polyhedron
193             //i = inpolyhedron(&rt,&box);
194             if (inpolyhedron(&rt,&box) < 1) { // if in working domain
195               noa = SaveAtom(&rt,j,or->or,noa,inph);
196             } else { // in polyhedron but not in domain (periodic)
197               pr = periodic(&rt); // this point on the other side
198               k = PolyhedronHasThis(&pr,phk,nopoly); // is in which
                      ph
199               phi = phk + k; // it's in this ph
200               if ((NotInThisList(&extend[0],ext,k))&&(phi->blank)) {
201                 // not already stored, not filled, then save it
202                 extend[ext] = k; phh->periodic[ext] = pr; ext++;
203               }
204             }
205           }
```

```
206
207             tmp.x = xyz.x + minorlat;
208             tmp.y = xyz.y + minorlat;
209             tmp.z = xyz.z;
210             rt = rotationT(tmp,rotM);
211             inph = inpolyhedron(&rt,phh);
212             if ((inph<1)&&(inpolyhedron(&rt,&box) < 1)) {
213               noa = SaveAtom(&rt,j,or->or,noa,inph);
214             }
215
216             tmp.x = xyz.x + minorlat;
217             tmp.y = xyz.y;
218             tmp.z = xyz.z + minorlat;
219             rt = rotationT(tmp,rotM);
220             inph = inpolyhedron(&rt,phh);
221             if ((inph<1)&&(inpolyhedron(&rt,&box) < 1)) {
222               noa = SaveAtom(&rt,j,or->or,noa,inph);
223             }
224
225             tmp.x = xyz.x;
226             tmp.y = xyz.y + minorlat;
227             tmp.z = xyz.z + minorlat;
228             rt = rotationT(tmp,rotM);
229             inph = inpolyhedron(&rt,phh);
230             if ((inph<1)&&(inpolyhedron(&rt,&box) < 1)) {
231               noa = SaveAtom(&rt,j,or->or,noa,inph);
232             }
233
234             xyz.x += majorlat;
235           }
236           xyz.y += majorlat;
237         }
238         xyz.z += majorlat;
239       } // END while
240       if (noa==0) {
```

```
241         phh->blank = 1;
242       } else {
243         phh->noa = noa;
244         phh->pr_count = ext;
245         phh->blank = 0;
246         // printf("Ph:%5d Ano = %10d\n",j,ano);
247       }
248     }
249     return j;
250 }
251
252 Polyhedron MakeBox(double *rotM)
253 {
254     double minx=gmx, miny=gmy, minz=gmz;
255     double maxx=px+gmx, maxy=py+gmy, maxz=pz+gmz;
256     int i;
257     Vector pb[8],r;
258     Face fb[6];
259     Polyhedron box;
260
261     pb[0].x = minx-0.01;  pb[0].y = miny-0.01;  pb[0].z = minz-0.01;
262     pb[6].x = maxx-0.01;  pb[6].y = maxy-0.01;  pb[6].z = maxz-0.01;
263     box.com.x = (minx+maxx)/2.0;
264     box.com.y = (miny+maxy)/2.0;
265     box.com.z = (minz+maxz)/2.0;
266     pb[1].x = pb[6].x;  pb[1].y = pb[0].y;  pb[1].z = pb[0].z;
267     pb[2].x = pb[6].x;  pb[2].y = pb[6].y;  pb[2].z = pb[0].z;
268     pb[3].x = pb[0].x;  pb[3].y = pb[6].y;  pb[3].z = pb[0].z;
269     pb[4].x = pb[0].x;  pb[4].y = pb[0].y;  pb[4].z = pb[6].z;
270     pb[5].x = pb[6].x;  pb[5].y = pb[0].y;  pb[5].z = pb[6].z;
271     pb[7].x = pb[0].x;  pb[7].y = pb[6].y;  pb[7].z = pb[6].z;
272     minx = maxx = miny = maxy = minz = maxz = 0.0;
273     for (i=0;i<8;i++) {
274       r = rotation(pb[i],rotM);
275       minx = min(minx,r.x);
```

```
276       maxx = max(maxx,r.x);
277       miny = min(miny,r.y);
278       maxy = max(maxy,r.y);
279       minz = min(minz,r.z);
280       maxz = max(maxz,r.z);
281     }
282     box.min.x = minx;
283     box.max.x = maxx;
284     box.min.y = miny;
285     box.max.y = maxy;
286     box.min.z = minz;
287     box.max.z = maxz;
288     fb[0].center = center4(&pb[1],&pb[2],&pb[5],&pb[6]);
289     fb[1].center = center4(&pb[0],&pb[3],&pb[4],&pb[7]);
290     fb[2].center = center4(&pb[2],&pb[3],&pb[6],&pb[7]);
291     fb[3].center = center4(&pb[0],&pb[1],&pb[4],&pb[5]);
292     fb[4].center = center4(&pb[4],&pb[5],&pb[6],&pb[7]);
293     fb[5].center = center4(&pb[0],&pb[1],&pb[2],&pb[3]);
294     fb[0].normal = Vnormal(&pb[1],&pb[2],&pb[5],&box,&fb[0]);
295     fb[1].normal = Vnormal(&pb[0],&pb[3],&pb[4],&box,&fb[1]);
296     fb[2].normal = Vnormal(&pb[2],&pb[3],&pb[6],&box,&fb[2]);
297     fb[3].normal = Vnormal(&pb[0],&pb[1],&pb[4],&box,&fb[3]);
298     fb[4].normal = Vnormal(&pb[4],&pb[5],&pb[6],&box,&fb[4]);
299     fb[5].normal = Vnormal(&pb[0],&pb[1],&pb[2],&box,&fb[5]);
300
301     for (i=0;i<6;i++) {
302       box.c[i] = fb[i].center;
303       box.n[i] = fb[i].normal;
304     }
305     box.nof = 6;
306
307     return box;
308 }
309
310 double dcos(double v)
```

```
311 {
312   double pi=4.0*atan2(1.0,1.0);
313   double rad=(v/180.)*pi;
314 
315   return cos(rad);
316 }
317 
318 Vector rotation(Vector p,double *r)
319 {
320   Vector u;
321 
322   u.x = *(r)*p.x + *(r+1)*p.y + *(r+2)*p.z;
323   u.y = *(r+3)*p.x + *(r+4)*p.y + *(r+5)*p.z;
324   u.z = *(r+6)*p.x + *(r+7)*p.y + *(r+8)*p.z;
325 
326   return u;
327 }
328 
329 Vector rotationT(Vector p,double *r)
330 {
331   Vector u;
332 
333   u.x = *(r)*p.x + *(r+3)*p.y + *(r+6)*p.z;
334   /* time(&start_time);
335   for (a=0;a<ano;a++) {
336     if (atom[a].ph > -1) {
337       printatom(&atom[a].r,atom[a].or,atom[a].ph);
338       printlammps(&atom[a].r,bno,atom[a].type,atom[a].ph);
339       bno++;
340     } else if (atom[a].ph == -1) {
341       printatom(&atom[a].r,4,999999);
342     } else {
343       printatom(&atom[a].r,7,999996);
344     }
```

```
345       for (i=1;i<10;i++) if (a == i*ano/10) printpercent(start_time,i
              *10.);
346     }
347     printpercent(start_time,100);
348     // FIXME
349     for (a=1;a<noj;a++) {
350       printatom(&p[junction[a]],5,999998);
351     }
352     for (a=0;a<nos;a++) {
353       printatom(&site[sitecenter[a]],6,999997);
354     } */
355     // FIXME
356     u.y = *(r+1)*p.x + *(r+4)*p.y + *(r+7)*p.z;
357     u.z = *(r+2)*p.x + *(r+5)*p.y + *(r+8)*p.z;
358
359     return u;
360 }
361
362 void printvector(int i,Vector *r)
363 {
364     printf("%3d %14.7f %14.7f %14.7f\n",i,r->x,r->y,r->z);
365     return;
366 }
367
368 void printatom(Vector *a, int value, int no)
369 {
370     FILE *fj;
371     char fname[20];
372
373     if (no < 10) sprintf(fname,"polygon-000%d",no);
374     else if (no < 100) sprintf(fname,"polygon-00%d",no);
375     else if (no < 1000) sprintf(fname,"polygon-0%d",no);
376     else sprintf(fname,"polygon-%d",no);
377     fj = fopen(fname,"a");
378
```

```
379    fprintf(fj,"%14.10f %14.10f %14.10f %d\n",a->x,a->y,a->z,value);
380    fclose(fj);
381    return;
382  }
383
384  void printlammps(Vector *a, int atomno, int tno, int fno)
385  {
386    FILE *fj;
387    char fname[20];
388
389    if (fno < 10) sprintf(fname,"Lpolygon-000%d",fno);
390    else if (fno < 100) sprintf(fname,"Lpolygon-00%d",fno);
391    else if (fno < 1000) sprintf(fname,"Lpolygon-0%d",fno);
392    else sprintf(fname,"Lpolygon-%d",fno);
393    fj = fopen(fname,"a");
394
395    fprintf(fj,"%7d%7d%14.8f%14.8f%14.8f\n",atomno,tno,a->x,a->y,a->z);
396    fclose(fj);
397    return;
398  }
399
400  /*int inSearchBox(Vector *v)
401  {
402    int i,j,k,r=0;
403    double minx,miny,minz,maxx,maxy,maxz;
404
405    for (i=0;i<nx;i++) {
406      for (j=0;j<ny;j++) {
407        for (k=0;k<nz;k++) {
408          minx = gmx+i*bx;
409          miny = gmy+j*by;
410          minz = gmz+k*bz;
411          maxx = minx + bx;
412          maxy = miny + by;
413          maxz = minz + bz;
```

```
414             if ((v->x >= minx)&&(v->x <= maxx)&&
415                 (v->y >= miny)&&(v->y <= maxy)&&
416                 (v->z >= minz)&&(v->z <= maxz)) {
417               r = i*ny*nz+j*nz+k;
418               i = nx;
419               j = ny;
420               k = nz;
421             }
422           }
423         }
424       }
425       return r;
426     }*/
427
428     int inpolyhedron(Vector *x, Polyhedron *p) //FIXME
429     {
430       Vector a2x;
431       int i,sum=0;
432       double val,v=-0.3;
433
434       for (i=0;i<p->nof;i++) {
435         a2x = va2b(&p->c[i],x);
436         val = dot(&a2x,&p->n[i]);
437         if (val > 0.0) sum=i+1;
438         else v=max(v,val);
439       }
440       if ((v > -0.3)&&(sum==0)) {sum=-1;};
441       return sum;
442     }
443
444     int inthebox(Vector *r)
445     {
446       if ((r->x > gmx)&&(r->x < gmx+px))
447         if ((r->y > gmy)&&(r->y < gmy+py))
448           if ((r->z > gmz)&&(r->z < gmz+pz)) return 1;
```

```
449       else return 0;
450     else return 0;
451   else return 0;
452 }
453
454 Vector center2(Vector *r, Vector *s)
455 {
456   Vector c;
457
458   c.x = (r->x + s->x)/2.0;
459   c.y = (r->y + s->y)/2.0;
460   c.z = (r->z + s->z)/2.0;
461   return c;
462 }
463
464 Vector shift(Vector *center, Vector *normal) //FIXME
465 {
466   Vector c;
467
468   c = unit(normal);
469   c = vscale(&c,1.2781*0.5);
470   c = va2b(&c,center);
471   return c;
472 } //FIXME
473
474 Vector center(Vector *r, Vector *s, Vector *t)
475 {
476   Vector c;
477
478   c.x = (r->x + s->x + t->x)/3.0;
479   c.y = (r->y + s->y + t->y)/3.0;
480   c.z = (r->z + s->z + t->z)/3.0;
481   return c;
482 }
483
```

```
484 Vector center4(Vector *r, Vector *s, Vector *t, Vector *u)
485 {
486   Vector c;
487 
488   c.x = (r->x + s->x + t->x + u->x)/4.0;
489   c.y = (r->y + s->y + t->y + u->y)/4.0;
490   c.z = (r->z + s->z + t->z + u->z)/4.0;
491   return c;
492 }
493 
494 Vector Vnormal(Vector *r, Vector *s, Vector *t, Polyhedron *ph, Face
      *fc)
495 {
496   Vector u,v,c,n,o;
497   double d;
498 
499   u = va2b(r,s);
500   v = va2b(s,t);
501   n = cross(&u,&v);
502   o = va2b(&ph->com,&fc->center);
503   d = dot(&o,&n);
504   if (d<0.0) n = cross(&v,&u);
505   return n;
506 }
507 
508 Vector va2b(Vector *r, Vector *s)
509 {
510   Vector u;
511 
512   u.x = s->x - r->x;
513   u.y = s->y - r->y;
514   u.z = s->z - r->z;
515   return u;
516 }
517 
```

```
518 double len(Vector *a)
519 {
520   double l;
521
522   l = a->x * a->x + a->y * a->y + a->z * a->z;
523   l = sqrt(l);
524   return l;
525 }
526
527 double lenA2B(Vector *r, Vector *s)
528 {
529   Vector u;
530   double l;
531
532   u.x = s->x - r->x;
533   u.y = s->y - r->y;
534   u.z = s->z - r->z;
535   l = u.x * u.x + u.y * u.y + u.z * u.z;
536   l = sqrt(l);
537
538   return l;
539 }
540
541 Vector unit(Vector *a)
542 {
543   Vector u;
544   double l=len(a);
545
546   u.x = a->x / l;
547   u.y = a->y / l;
548   u.z = a->z / l;
549   return u;
550 }
551
552 Vector unit3(double x,double y,double z)
```

```
553 {
554   Vector u;
555   double l=sqrt(x*x + y*y + z*z);
556 
557   u.x = x / l;
558   u.y = y / l;
559   u.z = z / l;
560   return u;
561 }
562 
563 Vector vscale(Vector *a,double value)
564 {
565   Vector s;
566 
567   s.x = a->x * value;
568   s.y = a->y * value;
569   s.z = a->z * value;
570   return s;
571 }
572 
573 Vector vmodify(Vector *a,double value,int idx)
574 {
575   Vector u;
576 
577   if ((idx>=0)&&(idx<=2)) {
578     if (idx==0) {
579       u.x = a->x + value;
580       u.y = a->y;
581       u.z = a->z;
582     } else if (idx==1) {
583       u.x = a->x;
584       u.y = a->y + value;
585       u.z = a->z;
586     } else if (idx==2) {
587       u.x = a->x;
```

```
588         u.y = a->y;
589         u.z = a->z + value;
590       }
591     }
592
593     return u;
594 }
595
596 Vector cross(Vector *r, Vector *s )
597 {
598     Vector u;
599
600     u.x = r->y * s->z - r->z * s->y;
601     u.y = r->z * s->x - r->x * s->z;
602     u.z = r->x * s->y - r->y * s->x;
603     return u;
604 }
605
606 double dot(Vector *a, Vector *b)
607 {
608     double d;
609
610     d = a->x * b->x + a->y * b->y + a->z * b->z;
611     return d;
612 }
613
614 Vector periodic(Vector *r)
615 {
616     Vector u = *r;
617
618     if (r->x < gmx) u.x = r->x + px;
619     else if (r->x > (gmx+px)) u.x = r->x - px;
620
621     if (r->y < gmy) u.y = r->y + py;
622     else if (r->y > (gmy+py)) u.y = r->y - py;
```

```
623
624   if (r->z < gmz) u.z = r->z + pz;
625   else if (r->z > (gmz+pz)) u.z = r->z - pz;
626
627   return u;
628 }
629
630 int Triangle2Color(Vector *r)
631 {
632   int color;
633   Vector d100,d110,d111;
634   int l1,l2,l3;
635
636   d100 = unit3(1.0,0.0,0.0);
637   d110 = unit3(1.0,1.0,0.0);
638   d111 = unit3(1.0,1.0,1.0);
639   l1 = (int)(15.0*(1.0-lenA2B(r,&d100)));
640   l2 = (int)(15.0*(1.0-lenA2B(r,&d110)));
641   l3 = (int)(15.0*(1.0-lenA2B(r,&d111)));
642   color=l1*16*16 + l2*16 + l3;
643
644   return(color);
645 }
646
647 Orientation MakeOrientation(int k)
648 {
649   Orientation or;
650   double phi,theta,psi,a,b,c,d;
651   FILE *fi;
652   Vector Z,u,p;
653   int i;
654
655   if (k==1) { // <100> || Z-axis
656     //or.or = k;
```

```
657     EulerAngles(&or.rotM[0][0],radian(90.0),radian(90.0),radian(0.0))
            ;
658   } else if (k==2) { // <110> || Z-axis
659     //or.or = k;
660     EulerAngles(&or.rotM[0][0],radian(135.0),radian(90.0),radian
            (270.0));
661   } else if (k==3) { // <111> || Z-axis
662     //or.or = k;
663     theta = radian(54.735610317245);
664     EulerAngles(&or.rotM[0][0],radian(135.0),theta,radian(270.0));
665   } else {
666     a = RandomWithIn(-1.0,1.0);
667     phi  = acos(a);
668     b = RandomWithIn(-1.0,1.0); u.x = b;
669     c = RandomWithIn(-1.0,1.0); u.y = c;
670     d = RandomWithIn(-1.0,1.0); u.z = d;
671     u = unit(&u);
672     b = u.x * sin(phi);
673     c = u.y * sin(phi);
674     d = u.z * sin(phi);
675     EulerParameters(&or.rotM[0][0],a,b,c,d);
676     //or.or = (int)((a+1.)*5000 + (b+1.)*500 + (c+1.)*50 + (d+1.)*5);
677   }
678
679   Z.x = Z.y = 0.0; Z.z = 1.0;
680   Z = rotationT(Z,&or.rotM[0][0]);  // <001> in crystal coordinate
681
682   p = unit(&Z);
683   p = vpositive(&p);
684   i=0; while ( ( !(intriangle(&p)) ) && (i<5) ) i = Vreorder(&p,i);
685   or.or = Triangle2Color(&p);
686
687   fi = fopen("euler_parameters","a");
688   fprintf(fi,"%5d%14.7f%14.7f%14.7f%14.7f%14.7f%14.7f%14.7f
          %14.7f%14.7f\n",
```

```
689       or.or,a,b,c,d,Z.x,Z.y,Z.z,p.x,p.y,p.z);
690     fclose(fi);
691
692     return or;
693   }
694
695   void EulerAngles(double *rotM,double phi,double theta,double psi)
696   { // phi = [0,2*pi] : theta = [0,pi] : psi = [0,2*pi]
697     *(rotM+0) = cos(psi)*cos(phi) - cos(theta)*sin(phi)*sin(psi);
698     *(rotM+1) = cos(psi)*sin(phi) + cos(theta)*cos(phi)*sin(psi);
699     *(rotM+2) = sin(psi)*sin(theta);
700     *(rotM+3) = -sin(psi)*cos(phi) - cos(theta)*sin(phi)*cos(psi);
701     *(rotM+4) = -sin(psi)*sin(phi) + cos(theta)*cos(phi)*cos(psi);
702     *(rotM+5) = cos(psi)*sin(theta);
703     *(rotM+6) = sin(theta)*sin(phi);
704     *(rotM+7) = -sin(theta)*cos(phi);
705     *(rotM+8) = cos(theta);
706     return;
707   }
708
709   void EulerParameters(double *rotM,double a,double b,double c,double d
        )
710   { // a = e0 = cos(phi/2) : bi+cj+dk = e1i+e2j+e3k => normalize * sin(
        phi/2)
711     *(rotM+0) = a*a + b*b - c*c - d*d;
712     *(rotM+1) = 2*(b*c + a*d);
713     *(rotM+2) = 2*(b*d - a*c);
714     *(rotM+3) = 2*(b*c - a*d);
715     *(rotM+4) = a*a - b*b + c*c - d*d;
716     *(rotM+5) = 2*(c*d + a*b);
717     *(rotM+6) = 2*(b*d + a*c);
718     *(rotM+7) = 2*(c*d - a*b);
719     *(rotM+8) = a*a - b*b - c*c + d*d;
720     return;
721   }
```

```
722
723 double RandomWithIn(double a,double b)
724 {
725   double r;
726
727   r = (double)rand() / ((double)(RAND_MAX) + (double)(1));
728   return r*( max(a,b) - min(a,b) ) + min(a,b);
729 }
730
731 double degree(double radian)
732 {
733   double pi=4.0*atan2(1.0,1.0);
734   return (radian/pi)*180.0;
735 }
736
737 double radian(double degree)
738 {
739   double pi=4.0*atan2(1.0,1.0);
740   return (degree/180.)*pi;
741 }
742
743 void Push(struct node** headref,int no)
744 {
745   struct node* newnode = malloc(sizeof(struct node));
746   newnode->no = no;
747   newnode->next = *headref;
748   *headref = newnode;
749 }
750
751 /*void makeVoid(Vector *r,double vradius)
752 {
753   int boxno = inSearchBox(r);
754   int a,b,c,i,j,k;
755   struct node* ca;
756   struct node* lista;
```

```
757     double dist;
758
759     n2i(boxno,&a,&b,&c);
760     for (i=a-1;i<a+2;i++)
761       for (j=b-1;j<b+2;j++)
762         for (k=c-1;k<c+2;k++) {
763           lista = searchbox[i2n(i,j,k)];
764           for (ca=lista;ca!=NULL;ca=ca->next) {
765             dist = lenA2B(&atom[ca->no].r,r);
766             if (dist < vradius) {
767               atom[ca->no].ph = -2;
768             }
769           }
770         }
771     return;
772 }*/
773
774 int printbox(int boxno,int count)
775 {
776     struct node* ca;
777     Vector r;
778     FILE *fi,*fj;
779     char fname[20],lname[20];
780
781     sprintf(fname,"polygon-%04d",boxno);
782     sprintf(lname,"Lpolygon-%04d",boxno);
783     fi = fopen(fname,"w");
784     fj = fopen(lname,"w");
785
786     for (ca=searchbox[boxno];ca!=NULL;ca=ca->next) {
787       r = atom[ca->no].r;
788       fprintf(fi,"%14.10f␣%14.10f␣%14.10f␣%d\n",r.x,r.y,r.z,atom[ca->no
              ].or);
789       if (atom[ca->no].ph > 0) {
```

```
790         fprintf(fj,"%7d%7d%14.8f%14.8f%14.8f\n",count,atom[ca->no].type
                ,r.x,r.y,r.z);
791         count++;
792       }
793     }
794     fclose(fi);
795     fclose(fj);
796
797     return(count);
798   }
799
800   int Search2Boxes(int boxA,int boxB,double tol)
801   {
802     struct node* ca;
803     struct node* cb;
804     double dist;
805     int count=0; //FIXME
806
807     for (ca=gb[boxA];ca!=NULL;ca=ca->next) {
808       for (cb=gb[boxB];cb!=NULL;cb=cb->next) {
809         dist = lenA2B(&atom[ca->no].r,&atom[cb->no].r);
810         //val=min(val,dist);
811         if (dist < tol) {
812           count++;
813           atom[cb->no].ph = 0;
814         }
815       }
816     }
817     //printf("%10d pairs, %f tol, min distance = %f\n",count,tol,val);
818
819     return(count);
820   }
821
822   void SearchDelete(struct node* lista,struct node* listb,double tol)
823   {
```

```
824     struct node* ca;
825     struct node* cb;
826     double dist;
827 
828     for (ca=lista;ca!=NULL;ca=ca->next) {
829       for (cb=listb;cb!=NULL;cb=cb->next) {
830         if ((atom[ca->no].ph >= 0)&&(atom[ca->no].ph < atom[cb->no].ph)
              ) {
831           dist = lenA2B(&atom[ca->no].r,&atom[cb->no].r);
832           if (( dist > 1e-5 )&&( dist < tol )) {
833             atom[cb->no].ph = -1;
834           }
835         }
836       }
837     }
838     return;
839 }
840 
841 /*int i2n(int i,int j,int k)
842 {
843     if (i==-1) i = nx-1;
844     if (j==-1) j = ny-1;
845     if (k==-1) k = nz-1;
846     if (i==nx) i = 0;
847     if (j==ny) j = 0;
848     if (k==nz) k = 0;
849     return i*ny*nz+j*nz+k;
850 }
851 
852 void n2i(int n,int *i,int *j,int *k)
853 {
854     *i = n / (ny*nz);
855     n -= *i * ny * nz;
856     *j = n / nz;
857     *k = n - *j * nz;
```

```
858     return;
859 }
860
861 void SearchPack(int i,int j,int k,double tol)
862 {
863     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i,j,k)],tol);
864     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i,j,k+1)],tol);
865     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i,j+1,k+1)],tol);
866     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i,j+1,k)],tol);
867     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i+1,j,k)],tol);
868     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i+1,j,k+1)],tol);
869     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i+1,j+1,k+1)],tol)
        ;
870     SearchDelete(searchbox[i2n(i,j,k)],searchbox[i2n(i+1,j+1,k)],tol);
871     return;
872 }*/
873
874 void printpercent(time_t start,double perc)
875 {
876     time_t end;
877     double runtime;
878     int hours,minutes,seconds;
879
880     time(&end);
881     runtime = difftime(end,start);
882     hours   = runtime / (60*60);
883     minutes = (runtime - hours*60*60)/60;
884     seconds = runtime - hours*60*60 - minutes*60;
885
886     printf("%6.2f%% done in %2d hours, %2d minutes, %2d seconds.\n",
        perc,hours,minutes,seconds);
887
888     return;
889 }
890
```

```
891 double  runline(Vector *r,double x,double y,double z,double dist) //
        FIXME
892 {
893   Vector p,d;
894   double t;
895
896   p.x = x - r->x; d.x = x;
897   p.y = y - r->y; d.y = y;
898   p.z = z - r->z; d.z = z;
899   t = lenA2B(r,&d);
900   p = unit(&p);
901
902   r->x = r->x + dist*p.x;
903   r->y = r->y + dist*p.y;
904   r->z = r->z + dist*p.z;
905
906   return t;
907 } //FIXME
908
909 int Vreorder(Vector *v,int i)
910 {
911   double a,b,c;
912
913   a = v->x; b = v->y; c = v->z;
914   if (i==0) {
915     v->x = b; v->y = c; v->z = a; i = 1;
916   } else if (i==1) {
917     v->x = b; v->y = c; v->z = a; i = 2;
918   } else if (i==2) {
919     v->x = b; v->y = a; v->z = c; i = 3;
920   } else if (i==3) {
921     v->x = b; v->y = c; v->z = a; i = 4;
922   } else if (i==4) {
923     v->x = b; v->y = c; v->z = a; i = 5;
924   } else {
```

```
925       printf("Something might be wrong!!\n");
926     }
927     return i;
928   }
929
930   Vector spherical(Vector *v)
931   {
932     double rho,phi,theta;
933     Vector r;
934
935     r.x = rho   = len(v);              // radial distance
936     r.y = phi   = acos(v->z / rho);    // zenith angle from pos-z (north
          pole)
937     r.z = theta = atan(v->y / v->x);   // azimuth angle from pos-x
938     return r;
939   }
940
941   int intriangle(Vector *v)
942   { // that 100, 110, 111 triangle
943     Vector r = spherical(v),n,o,a2x;
944     double pi=4.0*atan2(1.0,1.0);
945
946     o.x = o.y = o.z = 0.0;             // origin
947     n.x = 0.0; n.y = -1.0; n.z = 1.0; // normal = cross(100,111)
948     a2x = va2b(&o,v);
949     if ((r.z < pi/4)&&( dot(&a2x,&n)<=0 )) return 1;
950     else return 0;
951   }
952
953   Vector vpositive(Vector *a)
954   {
955     Vector s = *a;
956
957     if (s.x < 0.) s.x *= -1.;
958     if (s.y < 0.) s.y *= -1.;
```

```
959     if (s.z < 0.) s.z *= -1.;
960     return s;
961 }
```

Listing 8.4: global.h

```
1  //#define penalty 0.5 // allow 50% of 2*r0
2  #define lattice 3.615 // FCC (Cu) Copper Lattice
3  //#define lattice 3.52  // FCC (Ni) Nickel Lattice
4
5  Particle *atom;                    // pointer to atom
6  double px,py,pz,gmx,gmy,gmz;       // global domain definition
7  long ano,maxatom;                  // running atom # and max #
8  struct node** searchbox;           // pointer to search box
9  struct node** gb;                  // pointer to GB of each grain
10
11 extern int UpdateFilledList(Polyhedron *ph,int nopoly,int *filledlist
       );
12 extern int NotInThisList(int *list,int size,int check);
13 extern int PolyhedronHasThis(Vector *r,Polyhedron *ph,int nopoly);
14 extern Vector latsize(Vector *a,Vector *r,int mm);
15 extern int SaveAtom(Vector *rt,int j,int or,int noa,int tno);
16 extern void bound(Vector *min,Vector *max, Vector *p);
17 extern int FindNeighbor(Polyhedron *ph,int nopoly,int j,int *
       tobefilledlist,int tobefilled);
18 extern int FillAt(Polyhedron *ph,int nopoly,int j,Orientation *or,
       Vector *r);
19 extern Polyhedron MakeBox(double *rotM);
20 extern double dcos(double v);
21 extern Vector rotation(Vector p,double *r);
22 extern Vector rotationT(Vector p,double *r);
23 extern void printvector(int i,Vector *r);
24 extern void printatom(Vector *a, int value, int no);
25 extern void printlammps(Vector *a, int atomno, int tno, int fno);
26 extern int inpolyhedron(Vector *x, Polyhedron *p);
27 extern int inthebox(Vector *r);
28 extern Vector center2(Vector *r, Vector *s);
29 extern Vector shift(Vector *center, Vector *normal);
30 extern Vector center(Vector *r, Vector *s, Vector *t);
31 extern Vector center4(Vector *r, Vector *s, Vector *t, Vector *u);
```

```
32 extern Vector Vnormal(Vector *r, Vector *s, Vector *t, Polyhedron *ph
       , Face *fc);
33 extern Vector va2b(Vector *r, Vector *s);
34 extern double len(Vector *a);
35 extern double lenA2B(Vector *r, Vector *s);
36 extern Vector unit(Vector *a);
37 extern Vector unit3(double x,double y,double z);
38 extern Vector vscale(Vector *a,double value);
39 extern Vector vmodify(Vector *a,double value,int idx);
40 extern Vector cross(Vector *r, Vector *s );
41 extern double dot(Vector *a, Vector *b);
42 extern Vector periodic(Vector *r);
43 extern Orientation MakeOrientation(int k);
44 extern void EulerAngles(double *rotM,double phi,double theta,double
       psi);
45 extern void EulerParameters(double *rotM,double a,double b,double c,
       double d);
46 extern double RandomWithIn(double a,double b);
47 extern double degree(double radian);
48 extern double radian(double degree);
49 extern void Push(struct node** headref,int no);
50 extern int printbox(int boxno,int count);
51 extern int Search2Boxes(int boxA,int boxB,double tol);
52 extern void SearchDelete(struct node* lista,struct node* listb,double
        tol);
53 extern void printpercent(time_t start,double perc);
54 extern double  runline(Vector *r,double x,double y,double z,double
       dist);
55 extern int Vreorder(Vector *v,int i);
56 extern Vector spherical(Vector *v);
57 extern int intriangle(Vector *v);
58 extern Vector vpositive(Vector *a);
```

Listing 8.5: fillatom.h

```
1  typedef struct {
2    double  x;          // Vx
3    double  y;          // Vy
4    double  z;          // Vz
5  } Vector;
6
7  typedef struct {
8    int     noppf;      // # of points per face
9    int     p[50];      // points number
10   Vector  center;     // center (cg) of face
11   Vector  normal;     // normal of face
12 } Face;
13
14 typedef struct {
15   int     nof;        // # of faces
16   int     noa;        // # of atoms/particles in this polyhedron
17   int     nop;        // # of points
18   int     start;      // first particle # in
19   int     pr_count;   // # of periodic adjacent polyhedrons
20   int     blank;      // 1 = blank, 0 = filled
21   int     ngh[25];    // neighbor polyhedron number
22   int     p[100];     // point number
23   Vector  *p0;        // pointer point to the first point
24   Vector  c[30];      // centers of faces
25   Vector  n[30];      // normals of faces
26   Vector  periodic[8];  // reference point for filling adjacent ph
27   Vector  com;        // center of mass
28   Vector  min;        // absolute max location, change with orientation
29   Vector  max;        // absolute min location, change with orientation
30 } Polyhedron;
31
32 typedef struct {
33   int     ph;         // Polyhedron No
34   int     type;       // Atom Type
```

```
35    int      or;        // Orientation
36    Vector   r;         // Location vector
37  } Particle;
38
39  typedef struct {
40    double rotM[3][3];  // rotation matrix
41    int    or;          // orientation number
42  } Orientation;
43
44  struct node {         // search box link-list
45    int           no;
46    struct node*  next;
47  };
48
49  #define min(a,b) ((a) <= (b) ? (a) : (b))
50  #define max(a,b) ((a) >= (b) ? (a) : (b))
```

References

[1] A. L. Gurson. Continuum theory of ductile rupture by void nucleation and growth: Part 1 - yield criteria and flow rules for porous ductile media. *Journal of Engineering Materials and Technology*, 99:2–15, 1977.

[2] T. Belytschko, W. K. Liu, and B. Moran. *Nonlinear Finite Elements for Continua and Structures*. Wiley, 2001.

[3] M. A. Meyers, A. Mishra, and D. J. Benson. Mechanical properties of nanocrystalline materials. *Progress in Materials Science*, 51(4):429–450, May 2006.

[4] E. O. Hall. The deformation and ageing pf mild steel .3. discussion of results. *Proc. Phys. Soc. B*, 64:747–753, 1951.

[5] N. J. Petch. The cleavage strength of polycrystals. *J. Iron Steel Inst.*, 174:25, 1953.

[6] A. H. Chokshi, A. Rosen, J. Karch, and H. Gleiter. On the validity of the Hall-Petch relationship in nanocrystalline materials. *Scripta Metallurgica*, 23:1679–1684, 1989.

[7] M. A. Meyers and C. T. Aimone. Dynamic fracture (spalling) of metals. *Progress in Materials Science*, 28:1–96, 1983.

[8] S. Christy, H. R. Pak, and M. A. Meyers. *Mettallurgical Applications of Shock Wave and High-Strain-Rate Phenomena*. Dekker, New York, eds, 1986.

[9] D. C. Ahn, P. Sofronis, M. Kumar, J. Belak, and R. Minich. Void growth by dislocation-loop emission. *Journal of Applied Physics*, 101(063514):1–6, 2007.

[10] F. A. McClintock, A. S. Argon, S. Backer, and E. Orowan. *Mechanical Behavior of Materials*. Addison-Welsey, 1966.

[11] F. A. McClintock. A criterion for ductile fracture by the growth of holes. *Journal of Applied Mechanics*, 35:363–371, 1968.

[12] J. R. Rice and D. M. Tracey. On the ductile enlargement of voids in triaxial stress fields. *Journal of the Mechanics and Physics of Solids*, 17:201–217, 1969.

[13] A. Needleman. Void growth in an elastic-plastic medium. *J. Appl. Mech.*, 39:964–970, 1972.

[14] V. Tvergaard. Material failure by void growth to coalescence. *Adv. Appl. Mech.*, 27:83–151, 1990.

[15] Y. Huang, J. W. Hutchinson, and V. Tvergaard. Cavitation instabilities in elastic-plastic solids. *J. Mech. Phys. Solids*, 39:223–241, 1991.

[16] A. Needleman, V Tvergaard, and J. W. Hutchinson. *Fracture and Fatigue*. Springer Verlag, New York, 1992.

[17] R. Hill. *The Mathematical Theory of Plasticity*. Clarendon Press, Oxford, 1950.

[18] S. Timoshenko. *Theory of Elasticity*. McGraw-Hill Book Co., 1934.

[19] R. V. Southwell. *Introduction to the Theory of Elasticity*. Clarendon Press, 1936.

[20] V. A. Lubarda and M. A. Meyers. On plastic void growth in strong ductile materials. *Crnogorska Akademija Nauka I Umjetnosti*, 2003.

[21] A. L. Stevens, L. Davison, and W. E. Warren. Spall fracture in aluminum monocrystals: a dislocation-dynamics approach. *Journal of Applied Physics*, 43(12):4922–4927, 1972.

[22] V. Tvergaard. Influence of voids on shear band instabilities under plane strain conditions. *Int. J. Fract.*, 17:389–407, 1981.

[23] V. Tvergaard. On localization in ductile materials containing spherical voids. *Int. J. Fract.*, 18:237–252, 1982.

[24] V. Tvergaard and A. Needleman. Analysis of cup-cone fracture in a round tensile bars. *Acta Metallurgica*, 32:157–169, 1984.

[25] A. Needleman and V. Tvergaard. An analysis of ductile rupture in notched bars. *Journal of the Mechanics and Physics of Solids*, 32:461–490, 1984.

[26] LSTC. *LS-DYNA Keyword User's Manual*, version 970 edition.

[27] G. I. Taylor. Plastic strain in metals. *Int. J. Metals*, 62:307–324, 1938.

[28] J. Wen, Y. Huang, K. C. Hwang, C. Liu, and M. Li. The modified Gurson model accounting for the void size effect. *Int. J. Plasticity*, 21:381–395, 2005.

[29] W. Herrmann. Constitutive equation for dynamic compaction of ductile porous materials. *Journal of Applied Physics*, 40:2490–2499, 1969.

[30] M. Carroll and A. C. Holt. Suggested modification of the p-α model for porous materials. *Journal of Applied Physics*, 43:759–761, 1972.

[31] D. C. Barton, M. Waheed, M. S. Mirza, and P. Church. A numerical study of ductile void growth under dynamic loading conditions. *Int. J. Fracture*, 73:325–343, 1995.

[32] T. L. O'Regan, D. F. Quinn, M. A. Howe, and P. E. McHugh. Void growth simulations in single crystals. *Computational Mechanics*, 20:115–121, 1997.

[33] M. F. Horstemeyer, M. M. Matalanis, A. M. Sieber, and M. L. Botos. Micromechanical finite element calculations of temperature and void configuration effects on void growth and coalescence. *Int. J. Plasticity*, 16:979–1015, 2000.

[34] M. F. Horstemeyer and S. Ramaswamy. On factors affecting localization and void growth in ductile metals: A parametric study. *Int. J. Damage Mechanics*, 9:5–28, 2000.

[35] H. Agarwal, A. M. Gokhale, S. Graham, and M. F. Horstemeyer. Void growth in 6061-aluminum alloy under triaxial stress state. *Materials Science and Engineering*, A341:35–42, 2003.

[36] Z. Li, C. Wang, and C. Chen. The evolution of voids in the plasticity strain hardening gradient materials. *Int. J. Plasticity*, 19:213–234, 2003.

[37] T. Schacht, N. Untermann, and E. Steck. The influence of crystallographic orientation on the deformation behavior of single crystals containing microvoids. *Int. J. Plasticity*, 19:1605–1626, 2003.

[38] T. Tsuji. The void growth simulations in the hyper-elastic material with multiple seeds. *Trans Tech Publications, Switzerland*, pages 45–50, 2005.

[39] G. P. Potirniche, J. L. Hearndon, M. F. Horstemeyer, and X. W. Ling. Lattice orientation effects on void growth and coalescence in FCC single crystals. *Int. J. Plasticity*, 22:921–942, 2006.

[40] M. F. Ashby. The deformation of plastically non-homogeneous materials. *Philosophical Magazine*, 21(170):399–424, 1970.

[41] V. A. Lubarda, M. S. Schneider, D. H. Kalantar, B. A. Remington, and M. A. Meyers. Void growth by dislocation emission. *Acta Materialia*, 53:1397–1408, 2004.

[42] D. A. Jones and J. W. Mitchell. Observations on helical dislocation in crystals of silver chloride. *Philosophical Magazine*, 3(25):1–7, 1958.

[43] J. Silcox and P. B. Hirsch. Direct observation of defects in quenched gold. *Philosophical Magazine*, 4(37):72–89, 1959.

[44] F. J. Humphreys and P. B. Hirsch. The deformation of single crystals of copper and copper-zinc alloys containing aluminum particles ii. microstructure and dislocation-particle interactions. *Proc. Roy. Soc. Lond.*, 318:73–92, 1970.

[45] F. Seitz. Prismatic dislocations and prismatic punching in crystals. *Physical Review*, pages 723–724, 1950.

[46] L. M. Brown and G. R. Woolhouse. The loss of coherency of precipitates and the generation of dislocations. *Philosophical Magazine*, 21:329–345, 1970.

[47] M. S. Schneider, B. Kad, D. H. Kalantar, B. A. Remington, E. Kenik, H Jarmakani, and M. A. Meyers. Laser shock compression of copper and copper-aluminum alloys. *Int. J. Impact Engineering*, 32:473–507, 2005.

[48] A. M. Cuitiño and M. Ortiz. Three-dimensional crack-tip fields in four-point-bending copper single-crystal specimens. *Acta Materialia*, 44(6):863–904, 1996.

[49] T. Surholt and CHR. Herzig. Grain boundary self-diffusion in Cu polycrystals of different purity. *Acta Metallurgica*, 45:3817–3823, 1997.

[50] R. Raj and M. F. Ashby. Intergranular fracture at elevated temperature. *Acta Metallurgica*, 23:653–666, 1975.

[51] V. Tvergaard and C. Niordson. Nonlocal plasticity effects on interaction of different size voids. *Int. J. Plasticity*, 20:107–120, 2004.

[52] T. Ohashi. Crystal plasticity analysis of dislocation emission from micro voids. *Int. J. Plasticity*, 21:2071–2088, 2005.

[53] E. T. Seppälä, J. Belak, and R. E. Rudd. Effect of stress triaxiality on void growth in dynamic fracture on metals: A molecular dynamics study. *Physical Review B*, 69(134101):1–19, 2004.

[54] E. T. Seppälä, J. Belak, and R. E. Rudd. Onset of void coalescence during dynamic fracture of ductile metals. *Physical Review Letters*, 93(245503):1–4, 2004.

[55] E. T. Seppälä, J. Belak, and R. E. Rudd. Three-dimensional molecular dynamics simulations of void coalescence during dynamic fracture of ductile metals. *Physical Review B*, 71(064112):1–10, 2005.

[56] J. Marian, J. Knap, and M. Ortiz. Nanovoid cavitation by dislocation emission in aluminum. *Physical Review Letters*, 93(165503):1–4, 2004.

[57] J. Marian, J. Knap, and M. Ortiz. Nanovoid deformation in aluminum under simple shear. *Acta Materialia*, 53:2893–2900, 2005.

[58] S. G. Srinivasan, M. I. Baskes, and G. J. Wagner. Spallation of single crystal nickel by void nucleation at shock induced grain junctions. *Journal of Material Sciences*, 41:7838–7842, 2006.

[59] M. R. Gungor and D. Maroudas. Atomic-scale analysis of strain relaxation mechanisms in ultra-thin metallic films. *Mater. Res. Soc. Symp. Proc.*, 880E(BB2.2.1):03.2.1–03.2.6, 2005.

[60] L. P. Dávila, P. Erhart, E. M. Bringa, M. A. Meyers, V. A. Lubarda, M. S. Schneider, R. Becker, and M. Kumar. Atomistic modeling of shock-induced void collapse in copper. *Applied Physics Letters*, 86(161902):1–3, 2005.

[61] G. P. Potirniche, M. F. Horstemeyer, G. J. Wagner, and P. M. Gullett. A molecular dynamics study of void growth and coalescence in single crystal nickel. *Int. J. Plasticity*, 22:257–278, 2006.

[62] M. F. Horstemeyer, M. I. Baskes, and S. J. Plimpton. Length scale and time scale effects on the plastic flow of FCC metals. *Acta Materialia*, 49:4363–4374, 2001.

[63] N. A. Fleck and J. W. Hutchinson. A phenomenological theory for strain gradient effects in plasticity. *Journal of the Mechanics and Physics of Solids*, 41(12):1825–1857, 1993.

[64] N. A. Fleck, G. M. Muller, M. F. Ashby, and J. W. Hutchinson. Strain gradient plasticity: Theory and experiment. *Acta Metallurgica*, 42(2):475–487, 1994.

[65] J. Belak. On the nucleation and growth of voids at high strain-rates. *J. Comput.-Aided Mater. Design*, 5:193–206, 1998.

[66] J. Belak and R. Minich. *Fracture and ductile vs brittle behavior–theory, modeling and experiment*, volume 539. Warrendale, PA: Materials Research Society, 1999. MRS Symp. Proc.

[67] L. E. Murr, K. P. Staudhammer, and M. A. Meyers. *Metallurgical Applications of Shock-Wave and High-Strain-Rate Phenomena*. Dekker, 1986.

[68] G. P. Potirniche, M. F. Horstemeyer, B. Jelinek, and G. J. Wagner. Fatigue damage in nickel and copper single crystals at nanoscale. *Int. J. Fatigue*, 27:1179–1185, 2005.

[69] G. P. Potirniche and M. F. Horstemeyer. On the growth of nanoscale fatigue cracks. *Philosophical Magazine Letters*, 86:185–193, 2006.

[70] G. P. Potirniche, M. F. Horstemeyer, and P. M. Gullett. Atomistic modelling of fatigue crack growth and dislocation structuring in FCC crystals. *Proc. R. Soc. A*, 462:3707–3731, 2006.

[71] W. M. Ashmawi and M. A. Zikry. Single void morphological and grain-boundary effects on overall failure in F.C.C. polycrystalline systems. *Materials Science and Engineering*, A343:126–142, 2003.

[72] R. E. Rudd and J. F. Belak. Void nucleation and associated plasticity in dynamic fracture of polycrystalline copper: an atomistic simulation. *Computational Material Sciences*, 24:148–153, 2002.

[73] J. Belak. Molecular dynamics simulation of high strain-rate void nucleation and growth in copper. *Shock Compression of Condensed Matter*, 429:211–214, 1998. AIP Conf. Proc.

[74] J. A. Moriarty, J. F. Belak, R. E. Rudd, P. Söderlind, F. H. Streitz, and L. H. Yang. Quantum-based atomistic simulation of materials properties in transition metals. *Journal of Physics: Condensed Matter*, 14:2825–2857, 2002.

[75] P. Erhart, E. M. Bringa, M. Kumar, and K. Albe. Atomistic mechanism of shock-induced void collapse in nanoporous metals. *Physical Review B*, 72(052104):1–4, 2005.

[76] R. Sayle and E. J. Milner-White. Rasmol: Biomolecular graphics for all. *Trends in Biochemical Sciemces*, 20(9):374, 1995.

[77] K. Kadau. Private Communication.

[78] S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comp. Phys.*, 117:1–19, 1995. http://lammps.sandia.gov.

[79] M. S. Daw and M. I. Baskes. Semiempirical, quantum mechanical calculation of hydrogen embrittlement in metals. *Physical Review Letters*, 50(17):1285–1288, 1983.

[80] M. S. Daw and M. I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Physical Review B*, 29(12):6443–6453, 1984.

[81] S. J. Plimpton. *LAMMPS Documentation*. Sandia National Laboratory.

[82] K. Fuchs. *Proc. R. Soc. London, Ser. A*, 153:622, 1936.

[83] Y. Mishin, M. J. Mehl, D. A. Papaconstantopoulos, A. F. Voter, and J. D. Kress. Structural stability and lattice defects in copper: Ab initio, tight-binding and embedded-atom calculations. *Physical Review B*, 63(224106):1–16, 2001.

[84] C. S. Barrett and T. B. Massalski. *Structure of Metals, Crystallographic Methods, Principles and Data*. Pergamon, 3rd edition, 1980. page 1-16.

[85] K. Kadau, T. C. Germann, P. S. Lomdahl, B. L. Holian, D. Kadau, P. Entel, M. Kreth, F. Westerhoff, and D. E. Wolf. Molecular-dynamics study of mechanical deformation in nano-crystalline aluminum. *Metallurgical and Materials Transactions*, 35A:2719–2723, 2004.

[86] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 2nd edition, 2000. 147-163.

[87] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software*, 22(4):469–483, 1996.

[88] D. J. Benson. Private Communication.

[89] H. Goldstein, C. Poole, and J. Safko. *Classical Mechanics*. Addison Wesley, 3rd edition, 2002.

[90] C. L. Kelchner, S. J. Plimpton, and J. C. Hamilton. Dislocation nucleation and defect structure during surface indentation. *Physical Review B*, 58(17):11085–11088, 1998.

[91] D. Hull and D. J. Bacon. *Introduction to Dislocations*. Butterworth Heinemann, fourth edition, 2001.

[92] M. R. Gungor and D. Maroudas. Relaxation of biaxial tensile strain in ultrathin metallic films: Ductile void growth versus nanocrystalline domain formation. *Applied Physics Letters*, 87(171913):1–3, 2005.

[93] A. Y. Kuksin, I. V. Morozov, G. E. Norman, V. V. Stegailov, and I. A. Valuev. Standards for molecular dynamics modeling and simulation of relaxation. *Molecular Simulation*, 32(14-15):1005–1017, 2005.

[94] A. Y. Kuksin, G. E. Norman, V. V. Stegailov, and A. V. Yanilkin. Modeling of al crystal fracture under high-rate strain based on atomistic simulations. In M. Elert, M. D. Furnish, R. Chau, N. C. Holmes, and J. Nguyen, editors, *Shock Compression of Condensed Matter*, number 1, pages 317–320. APS, American Institute of Physics, 2007.

[95] G. E. Norman, A. Y. Kuksin, V. V. Stegailov, and A. V. Yanilkin. Atomistic simulation of plasticity and fracture of crystalline and polycrystalline metals under high strain rate. In M. Elert, M. D. Furnish, R. Chau, N. C. Holmes, and J. Nguyen, editors, *Shock Compression of Condensed Matter*, number 1, pages 329–334. APS, American Institute of Physics, 2007.

[96] A. V. Yanilkin, A. Y. Kuksin, G. E. Norman, and V. V. Stegailov. Atomistic simulations of fracture in nanocrystalline copper under high strain rates. In M. Elert, M. D. Furnish, R. Chau, N. C. Holmes, and J. Nguyen, editors, *Shock Compression of Condensed Matter*, number 1, pages 347–350. APS, American Institute of Physics, 2007.

[97] H. Van Swygenhoven, P. M. Derlet, and A. G. Frøset. Stacking fault energies and slip in nanocrystalline metals. *Nature*, 3:399–403, 2004.

[98] F. Cleri and V. Rosato. Tight-binding potentials for transition metals and alloys. *Physical Review B*, 48(1):22–33, 1993.

[99] J. Schiøtz and K. W. Jacobsen. A maximum in the strength of nanocrystalline copper. *Science*, 301:1357–1359, 2003.

[100] E. M. Bringa. Private Communication.

[101] H. N. Jarmakani, E. M. Bringa, P. Erhart, B. A. Remington, Y. M. Wang, N. Q. Vo, and M. A. Meyers. Molecular dynamics simulations of shock compression of nickel: From monocrystals to nanocrystals. *Acta Materialia*, 56:5584–5604, 2008.

[102] E. C. Aifantis. The physics of plastic deformation. *Int. J. Plasticity*, 3(3):211–247, 1987.

[103] H. Gao, Y. Huang, W. D. Nix, and J. W. Hutchinson. Mechanism-based strain gradient plasticity–I. theory. *Journal of the Mechanics and Physics of Solids*, 47:1239–1263, 1999.

[104] Y. Huang, H. Gao, W. D. Nix, and J. W. Hutchinson. Mechanism-based strain gradient plasticity–II. analysis. *Journal of the Mechanics and Physics of Solids*, 48:99–128, 2000.

[105] H. H. Fu, D. J. Benson, and M. A. Meyers. Analytical and computational description of effect of grain size on yield stress of metals. *Acta Materialia*, 49:2567–2582, 2001.

[106] H. H. Fu, D. J. Benson, and M. A. Meyers. Computational description of nanocrystalline deformation based on crystal plasticity. *Acta Materialia*, 52:4413–4425, 2004.

[107] M. A. Meyers and K. Chawla. *Mechanical Behavior of Materials*. Cambridge University Press, 2nd edition, 2009.

[108] A. C. F. Cocks and M. F. Ashby. Intergranular fracture during power-law creep under multiaxial stresses. *Metal Science*, 14:395–402, 1980.

[109] A. C. F. Cocks and M. F. Ashby. On creep fracture by void growth. *Progress in Materials Science*, 27:189–244, 1982.

[110] D. J. Bammann, M. L. Chiesa, and L. I. Horstemeyer, M. F. Weingarten. *Failure in ductile materials using finite element methods, Structural Crushworthiness and Failure*. Elsevier Applied Science, 1993.

[111] M. A. Meyers. Private Communication.

[112] M. F. Ashby, S. H. Gelles, and L. E. Tanner. The stress at which dislocations are generated at a particle-matrix interface. *Philosophical Magazine*, 19(160):757–771, 1969.

[113] M. F. Ashby and Lyman Johnson. On the generation of dislocations at misfitting particles in a ductile matrix. *Philosophical Magazine*, 20(167):1009–1022, 1969.

[114] S. Traiviratana, E. M. Bringa, D. J. Benson, and M. A. Meyers. Void growth in metals: Atomistic calculations. *Acta Materialia*, 56:3874–3886, 2008.

[115] D. J. Benson and O. Hallquist, J. A single surface contact algorithm for the post-buckling analysis of shell structures. *Computer Methods in Applied Mechanics and Engineering*, 78(2):141–163, 1990.

[116] K. S. Kumar, H. van Swygenhoven, and S. Suresh. Mechanical behavior of nanocrystalline metals and alloys. *Acta Materialia*, 51:5743–5774, 2003.

[117] N. Q. Vo, R. S. Averback, P. Bellon, S. Odunuga, and A. Caro. Quantitative description of plastic deformation in nanocrystalline cu: Dislocation glide versus grain boundary sliding. *Physical Review B*, 77:134108–1–9, 2008.

[118] M. F. Horstemeyer, M. I. Baskes, and S. J. Plimpton. Computational nanoscale plasticity simulations unsing embeded atom potentials. *Theoretical and Applied Fracture Mechanics*, 37:49–98, 2001.