Dislocation Based Modelling of Engineering Alloys



Bruce Bromage

University of Oxford

St Edmund Hall College

Department of Materials

Supervisors: Edmund Tarleton & Angus Wilkinson

A thesis submitted for the degree of

Doctor in Philosophy

Michaelmas Term 2022

Declaration

I certify that all work in this thesis is my own except where explicitly stated otherwise.

Bruce Bromage

Dedication

This thesis is dedicated to those who helped me through it: my friends, my family, my coworkers, any combination thereof and countless cups of tea.

Acknowledgements

I would like to thank my supervisors, Prof. Edmund Tarleton and Prof. Angus Wilkinson for their wisdom and assistance throughout my time as their student, particularly Ed for his seemingly endless patience with my unreliable communication and Angus for his expertise at navigating the idiosyncrasies of research life.

I would also like to thank Daniel Celis-Garza, who has always been a reliable sounding board and collaborator. Additionally, Haiyang Yu and Fengxian Liu for their helpful insights and willingness to test my ideas, as well as Prof. Felix Hofmann for inspiring conversations on three dimensional closure point positioning.

Special thanks goes to Dr. Johannes Ast, from the Friedrich Alexander University of Erlangen Nuernberg and formerly EMPA, for providing crucial experimental data and materials.

Publication relating to this thesis

- *Calculating dislocation displacements on the surface of a volume*, B. Bromage, E. Tarleton, Modelling and Simulation in Materials Science and Engineering, 2018 [1]

Chapter 5 is based almost exclusively on this work.

The code *EasyDD* around which this work is centred is currently open source and available at:https://github.com/TarletonGroup/EasyDD

Contents

1 Introduction **1**

1.1 Dislocations in Engineering Alloys 1
1.2 Types of Dislocation Models 2
1.3 Three Dimensional Discrete Dislocation Dynamics 5
1.4 Theory and Functionality of *DDLab* and *ParaDiS* 8
1.4.1 Stress and Force Calculations 8
1.4.2 Mobility Laws 11
1.5 Coupling Dislocation Dynamics with Finite Element Modelling . . 13
1.5.1 The Superposition Method 13
1.5.2 Displacement Calculations 15
1.5.3 Finite Volumes and Exiting Segments 17
1.5.4 Limitations and Avenues for Further Development 18

2 Basic Improvements **21**

2.1 Introduction 21
2.2 Topological Operations 21
2.2.1 Remeshing 22
2.2.2 Collision 24
2.2.3 Separation 27
2.2.4 Operation Synergy 30
2.3 The Time Integrator 36
2.3.1 The Quasi-Static Framework 36
2.3.2 Control Parameters 38
2.3.3 Tuning the Time Integrator 40
2.4 Summary 46

3 Mobility Laws **47**

3.1 Introduction 47
3.2 Mobility Law Basics 47
3.3 A Simple Example 52
3.4 Modifying for BCC Systems 59

3.5 Summary 67

4 Combining DDD with FEM **69**
4.1 Introduction 69
4.2 Displacement Rate 69
4.3 Virtual Segments 74
4.4 Variable Selection 78
4.5 Summary 79

5 Displacement Boundary Conditions **81**
5.1 Introduction 81
5.2 Adapting Barnett's Solution 81
5.2.1 A Non-Planar Method 81
5.2.2 Colliding and Non-planar loops 90
5.2.3 Correctly Using the Method 92
5.3 Calculating Displacements for Exited Segments 93
5.4 Implementing the Algorithm 97
5.5 Summary 103

6 Tungsten Cantilever **105**
6.1 Introduction 105
6.2 Simulation Setup 105
6.3 Results 110
6.4 Discussion 115
6.5 Summary 119

7 Conclusion **121**
7.1 Retrospective 121
7.2 Further Work 122
7.3 Final Conclusions 124

Chapter 1

Introduction to Modelling Plasticity

1.1 Dislocations in Engineering Alloys

Metals and alloys have been important materials for making tools for most of recorded human history. Since the industrial revolution, their importance has only increased as they have been used for major structural purposes. As with any engineering materials, a good understanding of the properties of these metals and alloys is key. One of the most important of these properties is how the material mechanically deforms under various conditions, since this knowledge allows us to find the best materials for a given purpose.

Mechanical deformation is not a simple process. Traditionally it has been investigated through mechanical testing, by running tests such as beam bending, indenting and tensile testing [2][3]. In 1907, Volterra calculated the elastic field for a type of crystal defect that was responsible for the plastic behaviour of crystalline materials [4]. Taylor subsequently named these defects "dislocations" in 1934 [5]. Both Taylor and Volterrra knew that the theoretical stresses need to match the energy requirements to move layers of atoms past each other to cause plastic yield were much higher than observed experimentally in metals and their alloys. These lower observed yield stresses were a result of dislocation action, which were revealed to be the primary mediation mechanism of plastic behaviour in these materials. Thus it becomes obvious that understanding the behaviour of dislocations provides crucial help to understanding how crystalline materials such as metals and alloys plastically deform.

Dislocations have been observed in crystalline materials by treatment of experimental samples. Slip steps on the surfaces of materials caused by exiting dislocations and dislocation structures such as persistent slip bands can be ob-

served in an optical microscope after etching the sample [6]. Further detail can be obtained by the use of an electron microscope [7], which can resolve individual dislocations. Mechanical testing of a sample in a transmission electron microscope can even provide a real-time view of how dislocations move through a material [8]. However this is restricted to very thin samples that can be used for electron microscopy and is not necessarily indicative of the general case. With current technology, it is not possible to directly observe dislocation behaviour in a bulk material thicker than a few tens of nanometers as it is mechanically deformed, which limits the understanding of dislocation behaviour obtainable through experimentation. Additionally, while it can be postulated why observed behaviour occurs, without direct observation only a model can provide evidence to support these theories. Examples include the nature of dislocation pile up in nanoindenting, the degree to which particular dislocation reactions result in work hardening, how dislocations interact with cracks during brittle and ductile fracture and precise dislocation movement processes that occur during necking.

Since practical experimentation has severe limitations when it comes to creating a complete picture of dislocation behaviour, theory must be used to fill in the gaps. By simulating dislocation behaviour through computational models, the theory of dislocation behaviour, and hence mechanical deformation of metals and alloys, may be further developed. The theory of the motion of individual dislocations and their resultant effect on deformation is called discrete dislocation plasticity (DDP), and the modelling of such behaviour is known as discrete dislocation dynamics (DDD). As the field of discrete dislocation plasticity is further developed, there is the scope to better understand how microstructural features of crystalline materials affect their properties and therefore learn how to design better alloys.

1.2 Types of Dislocation Models

Computational models of dislocations may be constructed over different scales of both length and time. One can use molecular dynamics to construct detailed models of how atoms are arranged in and around a dislocation and how they are rearranged as the dislocation moves. On a larger length scale, dislocation dynamics can be used to simulate dislocations as discrete objects, focusing on the movement and interactions of the dislocation system as a whole. Finally, at the largest length scales, continuum models can be constructed to demonstrate how a bulk material is likely to deform as a result of the general behaviour of the extant dislocations. As a general rule, if a method can model larger length scales then it can measure larger time scales as well.

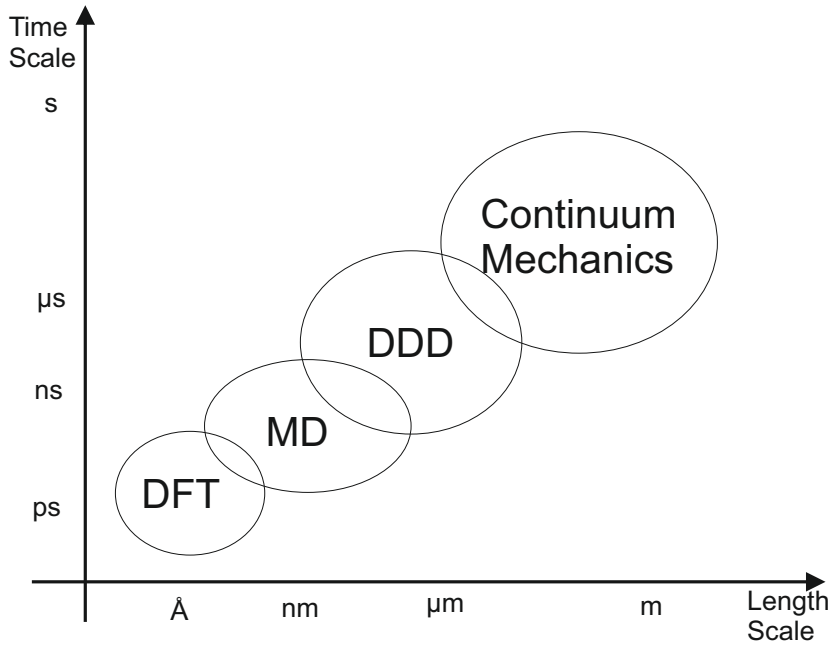


Figure 1.1: The time and length scales at which different modelling methods are used to model plastic behaviour.

Molecular dynamics (MD) has been used to model dislocation behaviour in two dimensions since the mid 1970s [9], using known inter-atomic forces as the basis for how atoms behave given the input conditions. These simulations can give a fairly accurate depiction of basic dislocation behaviour, such as glide [10] as well as interactions with point defects [11] at the atomic scale. However they are limited to pure edge dislocations which have a line direction perpendicular to the plane of atoms being simulated, since the cores of screw and mixed dislocations cause atomic displacements in three dimensions [12]. This means that the dislocation systems and interactions that can be modelled are limited. As computer technology improved, three dimensional molecular dynamics models of dislocations became possible, starting with the simplest case of a pure edge dislocation [13] and then progressing further to more complex cases. The core structure of screw and mixed dislocations are more intricate than those of edge dislocations. This can be further complicated by various crystal structures [14]. This means that MD simulations can be very useful for explaining and predicting dislocation core behaviour in a quantitative manner. Despite this, given that the total number of atoms in an MD simulation is limited to the order of a thousand atoms by the maximum available processing power, only a small total length of dislocations can be modelled at any one time, in a respectively small volume of material. Thus MD is good for modeling a single event, such as one collision between two dislocations and how their cores interact, but is not capable of giving context to that event, such as how it affects and is affected by a larger dislocation network. It should be

noted that dislocation core structure has also been modeled using density functional theory (DFT) [15]. Although this has only so far been descriptive of static dislocations, it is useful for finding activation energies for dislocation motion.

In contrast, DDD works by discretizing dislocations into lines which behave as individual objects in the simulation. This method has been used in two dimensions since 1959 [16]. DDD requires less programming power than MD since the effect of the input conditions on dislocations as a whole is calculated, rather than for the individual atoms that make up each dislocation. This requires formulae that describe basic dislocation behaviour, since these parameters are unknown, in contrast to MD simulations. The problem of developing these equations is not trivial [17] and the resultant laws of dislocation behaviour derived from them apply exclusively to an infinite body problem without further modification. However, as with MD, DDD simulations were first carried out in two dimensions, for which these formulae are simpler. A two dimensional DDD simulation can be conducted using one of two methods, depending on the geometric plane it is conducted in. The first method is in the plane that is normal to the dislocation line, where straight dislocations can be simulated as discrete points. This method can be used to simulate basic glide, climb and cross slip as well as certain dislocation interactions, such as annihilation and Lomer-Cotterel locks [18]. Using the second method, DDD simulations can alternatively be performed in the slip-plane of a specific slip system, which allows for curved dislocations to be modelled. This permits for the simulation of such events such as bowing and Frank-Read sources [19]. However 2D simulations rarely give a very complete demonstration of dislocation systems, except in very specific examples where the situation can be approximated to the 2D case, such as a thin film. Since dislocation lines move in three dimensions, 3D DDD models are necessary.

In 3D, DDD models become much more mathematically complex, since motion is not restricted to a single plane. The more complex formulae needed to describe the forces, stress fields, displacements and dislocation velocities makes them much more difficult to construct and more computationally expensive to run. However, these models give a more in-depth depiction of dislocation behaviour on the mesoscale, which is crucial for understanding mechanical properties [20]. As such there has been much work done on finding equations which better approximate physical dislocation behaviour when applied to discretised straight segments of dislocation [21][22]. Additionally there has been work done to apply parallel processing to DDD methods so that more complicated models can be simulated more quickly. Various different DDD codes have been created in order to address multiple physical problems, such as forest hardening [23], nanoindentation [24], scaling effects [25], dislocation clustering [26], fatigue [27], strain rate effects [28]

and temperature effects [29].

Continuum mechanics is a longer range method, operating from the mesoscale to the macroscale. It uses continuum equations to describe general behaviour after the application of given input simulation parameters. When describing mechanical deformation, frequently the Finite Element method (FEM) is used [30]. This method divides a notional simulated volume into smaller elements. Boundary and input conditions are used to define the simulation and then formulae are then used to calculate the average state of each element under these conditions. The length scale can be adjusted by altering the size of the elements. This makes it easier to model larger volumes than DDD or MD, as well as simulating many grained structures by combining multiple problems [31]. By constructing the relevant formulae in accordance with crystallographic symmetry, general rules for slip behaviour can be used to predict how dislocation behaviour affects the deformation of a single solid object [32]. Cleveringa et al. [33] compared such models to equivalent DDD models and experimental results. They found that while finite element models had a fairly good approximation to reality, at smaller length scales the importance of discrete events predicted by DDD models but not by FEM models affected the accuracy of the simulation. As such they concluded that, when viewed on their own, FEM models had severe limitations when attempting to predict plastic behaviour at the microscale.

These methods for modeling dislocation behaviour each have their own advantages and disadvantages. Some of these problems can be overcome by combining multiple types of model into a single multiscale model. MD has been used to model dislocation core behaviour in order to improve DDD simulations [34]. DDD can be combined with FEM modeling to improve functionality [35]. It has even been attempted to combine MD with continuum mechanics to better understand the long range effects of dislocation core structure and interactions with interfaces [36]. It can be seen that given the various length scales involved in dislocation behaviour, improving the capabilities of models at any of these length scales would aid in better understanding dislocation behaviour, although the primary focus here is dislocation dynamics.

1.3 Three Dimensional Discrete Dislocation Dynamics

There are two main types of 3D discrete dislocation dynamics model [37]: lattice based and nodal based. They are distinguished by how the dislocations are discretised, shown in Figure 1.2. Lattice based codes divide dislocations into screw and

edge segments that are confined to a predefined grid. Nodal based codes simulate the dislocations as a series of nodes along the dislocation line connected by straight segments.

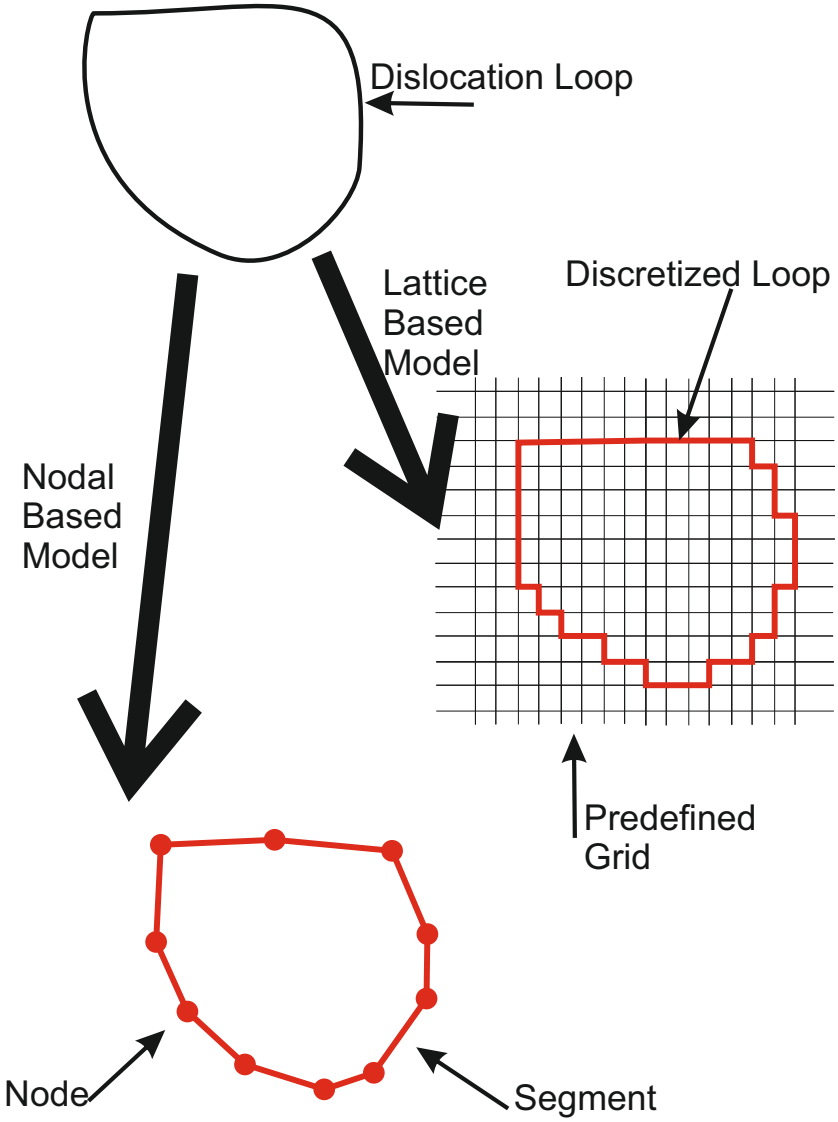


Figure 1.2: How the two main types of 3D discrete DDD models discretize a dislocation loop.

Modern lattice based codes are derived from work done by Kubin et al. in 1992 [38], which generated stress-strain curves from theoretical scenarios to be compared with experimental data. The original code has since been expanded upon to provide greater functionality. In 1998, Verdier et al. [39] used this work to create *Tridis*, a segment based code that was designed for looking at mechanical responses in FCC and BCC structures. It has been gradually refined to improve functionality, as well as being parallelised [40] and coupled with FEM modeling [41] in order to model more realistic scenarios such as finite volumes, although it is still limited to crystal systems with cubic symmetry. As well as basic dislocation

behaviour, *Tridis* has also been used to model nanoindentation, fatigue, torsion and compression tests and the effects of various types of grain [42].

The other major lattice based code is *microMegas*, with a derived code called *MobiDiC*. This is open source software designed by Lemarchand et al. in 2001 [43] to implement finite boundary conditions in order to more easily couple DDD and FEM code to create a Discrete-Continuous model which has features of both DDD and continuum mechanics. It functions with FCC, BCC and HCP crystal systems. *MobiDiC* is coded to work for BCC systems in the athermal temperature regime and, unlike *microMegas*, includes secondary slip systems. Areas of interest the *microMegas* code has been used to model are dislocation-particle interactions, forest hardening and grain size effects [44].

Lattice based DDD codes are generally quite fast to run and easily parallelised [40]. In addition to this, by altering the predefined grid the effects of crystallography can be easily incorporated into the model. However, due to the fact that arbitrary mixed dislocation types are not taken into account, there is a limit to their accuracy. This is most apparent in more complex dislocation reactions and when calculating the displacement caused by dislocation loops.

Multiple types of nodal DDD code have been created simultaneously in order to simulate different types of dislocation behaviour. The first 3D nodal DDD code was called *micro3D*, created by Zbib et al. in 1998 [45]. The initially basic DDD code was combined with FEM in 2001 by Yasin et al.[35] to improve the functionality so that it may model dislocation interactions with various surfaces. Further adaptations have incorporated thermal effects and dislocation inertia, the latter of which is unusual for DDD models due to their quasi-static nature. The current version of the code is called *Multiscale Dislocation Dynamics Plasticity* and has been variously used to model dislocation-crack interactions [46], dislocation-wave interactions [47] and shock loading [28].

The next 3D nodal DDD code was created by Schwarz a year after micro3D [48] and is called *PARANOID*. It was specifically designed for simulations in confined metallic and semi-conducting materials. It does not allow for the occurrence of cross-slip. Most significantly it has been used to explore the treatment of image forces on dislocations intersecting a free surface [49].

In 2004, Bulatov et al. [50] detailed the method by which a nodal based DDD code may be parallelised. This led to the development of the two codes, *ParaDiS* and *DDlab*, by Arsenlis et al [50][51], which are based on the same principles but written in different languages, C and MATLAB respectively. The key principals behind the operation of these codes were further outlined by Bulatov and Cai in 2006 [52]. The aim was to reduce the main disadvantage of nodal based codes when compared to lattice based codes, i.e. their speed. The basic versions of both

codes are open source, leading to it being adopted and developed by multiple groups, creating several different versions. The code possesses the potential to model multiple dislocation processes, with mechanisms included in the basic code to model glide, climb, cross-slip and periodic or free surface boundary conditions. The code is modular, enabling parts of it to be changed in order to alter its specific functionality without having to make major changes to the rest of the code to ensure it still works.

Nodal based codes have the advantage over lattice based codes that more diverse geometric configurations can be modeled, with the draw back that the calculations required to solve the problems associated with such configurations are more computationally expensive. Also, since there is no predefined grid, additional terms need to be introduced into the calculations in order to account for crystallography.

1.4 Theory and Functionality of *DDLab* and *ParaDiS*

1.4.1 Stress and Force Calculations

The *DDLab* and *ParaDiS* codes rely on established mathematical principals to run physical simulations. Modelling the forces required to move dislocations and their subsequent motion helps in understanding plastic behaviour. Peach and Koehler derived an equation for the force on a dislocation as a result of the local stress acting on the dislocation [53].

$$\boldsymbol{f}^{PK}(x) = (\boldsymbol{\sigma}(\boldsymbol{x}) \cdot \boldsymbol{b}) \times \boldsymbol{\xi}(\boldsymbol{x}) \tag{1.1}$$

where $\boldsymbol{f}^{PK}(\boldsymbol{x})$ is the force at point $\boldsymbol{x}$ as a result of the local stress, $\boldsymbol{\sigma}(\boldsymbol{x})$, $\boldsymbol{b}$ is the Burgers vector and $\boldsymbol{\xi}(\boldsymbol{x})$ is the tangent to the dislocation line at point $\boldsymbol{x}$. An additional term is required to account for self force, $\boldsymbol{f}^{SELF}(\boldsymbol{x})$, which is the force due to internal stresses generated by the shape of the dislocation network and the dislocation line tension. Singularities occur in the traditional expression used to describe these stresses, hence resulting in nonphysical representations of the forces. Cai et al. derived an expression to represent the dislocation core by an isotropic dispersion of the Burgers vector which is both non-singular and self consistent[51]:

$$\sigma^{NS}_{\alpha\beta}(\boldsymbol{x}) = \frac{\mu}{8\pi}\oint_C \partial_i\partial_p\partial_p R_a(b_m\epsilon_{im\alpha}dx'_\beta + b_m\epsilon_{im\beta}dx'_\alpha) + \frac{\mu}{4\pi(1-\nu)}\oint_C b_m\epsilon_{imk}(\partial_i\partial_\alpha\partial_\beta R_a - \delta_{\alpha\beta}\partial_i\partial_p\partial_p R_a)dx'_k \tag{1.2}$$

where:

$$R_a \equiv \sqrt{R^2 + r_c^2}$$
$$\partial_i\partial_j R_a \equiv \partial^2 R_a/\partial x_i\partial x_j \tag{1.3}$$

This expression describes the non-singular stress, $\sigma^{NS}_{\alpha\beta}$, at point $\boldsymbol{x}$ as a function of the shear modulus, μ, the Poisson's ratio, ν, the Burgers, $\boldsymbol{b}$, vector and the radial distance from the dislocation core, R_a, with R being the radial distance from the center of the dislocation and r_c being the dislocation core radius. The Levi-Cevita symbol, $\boldsymbol{\epsilon}$, combined with relevant component of the Burgers vector, b_m, allows the integral over the entire dislocation network, C, to evaluate the stress components with the correct sign and magnitude outside of the core of the dislocation in accordance with linear elastic theory. This non-singular stress can then be used to calculate $\boldsymbol{f}^{SELF}(\boldsymbol{x})$ in a similar manner to $\boldsymbol{f}^{PK}(\boldsymbol{x})$.

The energy associated with the dislocation core per unit length is known simply as the core energy, E_c. Physically speaking, this energy is a result of the misalignment of atoms in the dislocation core, causing the potential energy due to inter-atomic interactions to be greater than in the regularly positioned atoms in the rest of the bulk of the material. It can be seen that E_c is dependent on the number of misaligned atoms and hence the size of the dislocation core. This means that E_c is a function of the dislocation core radius, r_c. The actual energy contribution per unit volume from this distorted section of material can usually be expressed in terms of the material's shear modulus, μ. Alternatively, the results of work done using MD models may be used to describe the dislocation core energy [54].

Due to the nature of a given dislocation's core structure, normal elastic behaviour cannot be assumed and an extra term for all forces associated with the core must be added to the elastic forces mentioned previously. However this term is often ignored because there is insufficient atomistic data to calculate E_c and the contribution to the total force of the system is considered small[52]. Thus the total force on a dislocation at a given point $\boldsymbol{x}$, $\boldsymbol{f}^{TOT}(\boldsymbol{x})$, is assumed to be the sum of $\boldsymbol{f}^{PK}(\boldsymbol{x})$ and $\boldsymbol{f}^{SELF}(\boldsymbol{x})$.

As mentioned previously, the dislocation network has been discretised into nodes connected by segments, meaning that dislocation loops are approximated by polygons of a series of straight segments, with nodes situated at the vertices. Additional nodes may be included along straight sides of the polygon if an individual segment is too long. The Burgers vector must be conserved at every node except those that are immobile, such as the ones often used to model a Frank-Read source that is not part of a complete loop. The way a dislocation system moves is calculated by analysing the forces at the nodes. The force at a node that results from the segments it is attached to is modelled by integrating over a shape function, which is zero for all of the segments in the system not attached to the node. It is an integral of $\boldsymbol{f}^{TOT}(x)$ over the segments, weighted so that the contribution drops to zero at the connected nodes. Since the dislocation is approximated to a straight line, $\boldsymbol{\xi}(x)$ is simply the line direction of the relevant segment.

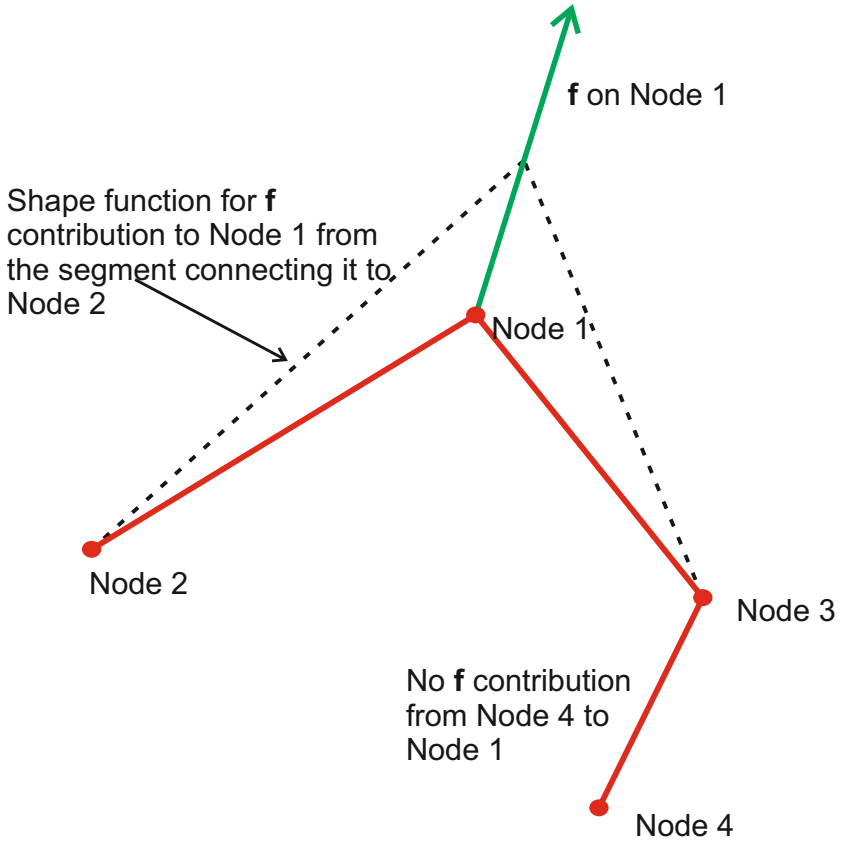


Figure 1.3: Schematic of the shape function used to calculate nodal forces. Forces acting on all segments are calculated. The forces on nodes are comprised of contributions from all segments they are connected to.

A key assumption of the model is that the forces acting to move a node are balanced by a drag force so that the resultant magnitude is zero. Since there is no net force, this means that there is no acceleration, so nodal velocities are constant over each time increment. This is known as the quasi-static condition. This allows

dislocations to change velocity only at the beginning of each time increment, when forces are recalculated, and at no other point. The exact way in which a node may move is defined by a mobility function relevant to the specific crystal system, however there are material properties that must be known in order to calculate forces. They are coded as input variables and are the shear modulus, the Poisson's ratio, the dislocation core radius, and the core energy of a pure screw dislocation. Time increments are limited to a maximum value, but need to be be decreased when the complexity of the events occurring per time increment increases. The time increment is decreased by using a time integrator which checks if the size of a time increments satisfies the necessary conditions.

1.4.2 Mobility Laws

Changes in the position of dislocations in the network can be simulated by changing the positions of the nodes while maintaining segment connectivity. To do this, one must calculate the nodal velocities to know how much to move each node after each time increment. This is covered in detail by Bulatov and Cai [52] and a brief summary will be given here. It is important to note the the notation used by Bulatov and Cai, and reproduced here, is **not** Einstein notation and that the indices i and j are used to refer to different nodes in the network. The velocity of a given node can be calculated using:

$$\boldsymbol{v}_i = M(\boldsymbol{f}_j) \tag{1.4}$$

where the velocity of node i, v_i, is some function, M, of the forces acting on all of the nodes it is connected to, f_j. M is known as the mobility function. The mobility function is derived by equating the driving force of motion, $\boldsymbol{f}_{drive}$, with the drag force resisting motion, $\boldsymbol{f}_{drag}$, so that there is no resultant force and hence a constant velocity at any point x along a segment:

$$\boldsymbol{f}_{drive} = -\boldsymbol{f}_{drag} = \boldsymbol{\mathcal{B}}(\boldsymbol{\xi}(x)) \cdot \boldsymbol{v}(\boldsymbol{x}) \tag{1.5}$$

The driving force is a combination of the Peach-Koehler force due to the applied stress and the interaction forces due to the stress fields generated by all dislocations in the system. The drag force at a given point $\boldsymbol{x}$ is dependent on the velocity at $\boldsymbol{x}$, $\boldsymbol{v}(\boldsymbol{x})$, and the drag coefficient tensor, $\boldsymbol{\mathcal{B}}$, which is material specific. Since the force on any node, i, is a weighted average of the force on the segments as described by the shape function, $N_i(\boldsymbol{x})$, the force on a node can be described as the sum of its interactions with all nodes it is connected to, j,:

$$\sum_j \boldsymbol{B}_{ij} \cdot \boldsymbol{v}_j = \boldsymbol{f_i} \tag{1.6}$$

where $\boldsymbol{B}_{ij}$ is the integral over the entire dislocation network, C, of the drag contribution from the segment connecting nodes i and j with respect to its length, $L(x)$:

$$\boldsymbol{B}_{ij} = \oint_C N_i(x)\boldsymbol{\mathcal{B}}(\boldsymbol{\xi}(x))N_j(x)dL(x) \tag{1.7}$$

If it is assumed that connected nodes have similar velocities (i.e $\boldsymbol{v}_i \approx \boldsymbol{v}_j$), Equation 1.6 can be rearranged to give:

$$\boldsymbol{v}_i \approx \left(\sum_j \boldsymbol{B}_{ij}\right)^{-1} \cdot \boldsymbol{f}_i \tag{1.8}$$

It has been shown that the velocity of dislocations not only depends on the external force applied but also the drag force on these dislocations. This means that the mobility function must involve the Burgers vector and slip plane of the dislocation, as well as the crystallographic system of the material and material constants dictating the ease of glide and climb. Due to the complexity of the relationships between these factors, different mobility functions must be created for body centered cubic, face centered cubic and hexagonal close packed systems respectively. The input file must specify the mobility function required for the simulation, as well as the drag coefficients relating to pure screw dislocations, $Bscrew$ and pure edge dislocations, $Bedge$. Two additional drag coefficients relating to screw dislocation movement perpendicular to line direction, $Beclimb$ (sic), and parallel to line direction, $Bline$, are also required. The code itself uses these inputs to calculate equivalents for edge glide and screw glide and then combines the results as necessary to account for mixed dislocations.

Once the velocity of all nodes has been obtained, calculating their position after a designated time increment requires the previously mentioned time integrator function, since velocity is position differentiated with respect to time. There are various functions which can be used to get an approximate solution, although there tends to be a trade off between accuracy and computational expense. The new location of all the nodes, and by extension segments, is plotted and the node and segment lists are updated with this information. However, due to the changes in the dislocation network, the forces previously calculated are no longer accurate. They are therefore recalculated for the new dislocation positions and used to generate new velocity vectors for all nodes which are then used to calculate new positions after the next time increment, completing another iteration. This

process continues until a previously designated number of iterations is completed, giving the final positions of all elements in the dislocation network at the end of the simulation.

There are open source mobility functions for BCC and FCC crystal systems [55], which can be used as they are or modified for a desired purpose. Cubic symmetry makes the drag matrix easier to construct. The converse of this means that developing a comprehensive mobility law for HCP structures is more challenging, along with the fact that conventionally they are indexed with four basis vectors. Additionally, more complex types of dislocation that can appear in HCP structures, such as combined dislocations with large Burgers vectors, may still be mobile. Various attempts have been made to model HCP structures with DDD, mainly by simplifying the problem by creating mobility laws with limited numbers of active slip systems. However these mobility laws only function for specific cases where the assumptions used for the simplification are valid. More recently, Aubry et al. [56] have developed a more comprehensive mobility law for HCP crystals, designed to apply to a more general case.

1.5 Coupling Dislocation Dynamics with Finite Element Modelling

1.5.1 The Superposition Method

When simulating bulk properties, periodic boundary conditions are generally sufficient to simulate a realistic response to an applied stress. However, simulating micromechanical tests requires simulating finite domains with complex geometry and boundary conditions. This can be achieved by coupling a DDD code with a finite element (FE) code. There are two commonly used methods for doing this. The superposition method [57] and the discrete-continuous model (DCM) [58]. In DCM the plastic strain generated by the dislocation motion is used in an elastic FEM code to solve for the total stress; rather than a corrective stress as in the superposition method. The DCM method therefore does not require the dislocation displacement field to be evaluated anywhere. However, the resolution of the predicted displacement field is then limited by the element size. Another challenge with DCM is avoiding "double counting" of the dislocation stress field when segments are in the same finite element.

The superposition method originally proposed in 2D [57], then extended to 3D [39, 59, 60] uses FEM to calculate the corrective stress field, $\hat{\boldsymbol{\sigma}}$, which arises due to any applied loading, and the loading generated by the elastic fields of any

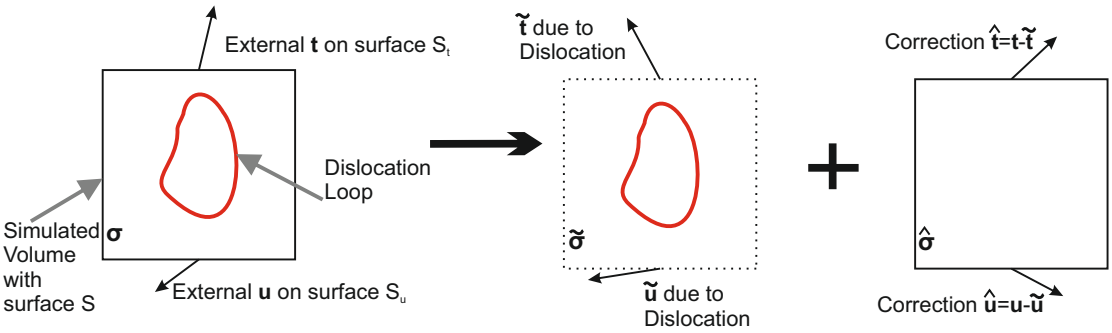


Figure 1.4: The superposition principle. The boundary values of tractions, $\boldsymbol{T}$, and displacements, $\boldsymbol{U}$, applied to the surfaces, S_T and S_U respectively. These are enforced by subtracting the tractions and displacements generated by the dislocations in an infinite body, $\tilde{\boldsymbol{T}}$ and $\tilde{\boldsymbol{U}}$, and solving the modified boundary value problem, $\hat{\boldsymbol{T}} = \boldsymbol{T} - \tilde{\boldsymbol{T}}$ on S_T and $\hat{\boldsymbol{U}} = \boldsymbol{U} - \tilde{\boldsymbol{U}}$ on S_U, using FEM to obtain the corrective stress field $\hat{\boldsymbol{\sigma}}$.

dislocations present inside the volume. First, a finite element mesh corresponding to the desired finite volume is constructed. As in any mechanical finite element problem, every FE surface node is either in S_T or S_U, depending on if tractions, $\boldsymbol{T}$, or displacements, $\boldsymbol{U}$, are specified. Then, at each time increment the segment tractions, $\tilde{\boldsymbol{T}}$, and hence the nodal forces, as well as the segment displacements, $\tilde{\boldsymbol{U}}$, arising from the dislocation network in an infinite medium are evaluated analytically at the FE nodes on S_T and S_U respectively. These boundary values are then subtracted from the desired traction, $\boldsymbol{T}$, and displacement, $\boldsymbol{U}$, values. These modified boundary conditions, $\hat{\boldsymbol{T}} = \boldsymbol{T} - \tilde{\boldsymbol{T}}$ on S_T and $\hat{\boldsymbol{U}} = \boldsymbol{U} - \tilde{\boldsymbol{U}}$ on S_U, are then used to solve for the corrective stress, $\hat{\boldsymbol{\sigma}}$, using an elastic FE code. The elastic stress in a finite body subject to the desired boundary conditions is finally found by simply adding the infinite body dislocation field, $\tilde{\boldsymbol{\sigma}}$, and the corrective field, $\hat{\boldsymbol{\sigma}}$, from FEM: $\boldsymbol{\sigma} = \tilde{\boldsymbol{\sigma}} + \hat{\boldsymbol{\sigma}}$ as illustrated in Figure 1.4. It should be noted that this is a purely elastic solution and that no shape change is accounted for, since a small strain condition is assumed. This means that, while the boundary conditions are updated at each increment, the boundary, S, and the FE mesh remain unchanged. An interesting discussion of this point is in [61]. This requires one additional constraint which is that dislocations are prevented from exiting through any part of the surface where displacement boundary conditions are specified, S_U. Formally,

the following is solved:

$$\left.\begin{aligned} &\nabla \cdot \tilde{\boldsymbol{\sigma}} = 0 \\ &\tilde{\boldsymbol{\sigma}} = \boldsymbol{C} : \tilde{\boldsymbol{\varepsilon}} \\ &\tilde{\boldsymbol{\varepsilon}} = \tfrac{1}{2}\left(\nabla\tilde{\boldsymbol{u}} + (\nabla\tilde{\boldsymbol{u}})^T\right) \end{aligned}\right\} \text{in V} \tag{1.9}$$

$$\tilde{\boldsymbol{\sigma}}.\boldsymbol{n} = \tilde{\boldsymbol{T}} \text{ on } S_T \tag{1.10}$$

$$\left.\begin{aligned} \tilde{\boldsymbol{u}} &= \tilde{\boldsymbol{U}} \quad t > 0 \\ \tilde{\boldsymbol{u}} &= \boldsymbol{0} \quad t = 0 \end{aligned}\right\} \text{ on } S_U \tag{1.11}$$

and the corrective problem

$$\left.\begin{aligned} &\nabla \cdot \hat{\boldsymbol{\sigma}} = 0 \\ &\hat{\boldsymbol{\sigma}} = \boldsymbol{C} : \hat{\boldsymbol{\varepsilon}} \\ &\hat{\boldsymbol{\varepsilon}} = \tfrac{1}{2}\left(\hat{\boldsymbol{u}} + (\nabla\hat{\boldsymbol{u}})^T\right) \end{aligned}\right\} \text{in V} \tag{1.12}$$

$$\hat{\boldsymbol{\sigma}}.\boldsymbol{n} = \boldsymbol{T} - \tilde{\boldsymbol{T}} \text{ on } S_T \tag{1.13}$$

$$\hat{\boldsymbol{u}} = \boldsymbol{U} - \tilde{\boldsymbol{U}} \text{ on } S_U \tag{1.14}$$

if the solution to both problems are known, then their superposition solves the desired mixed boundary value problem:

$$\left.\begin{aligned} &\nabla \cdot \boldsymbol{\sigma} = \nabla \cdot (\hat{\boldsymbol{\sigma}} + \tilde{\boldsymbol{\sigma}}) = 0 \\ &\boldsymbol{\sigma} = \hat{\boldsymbol{\sigma}} + \tilde{\boldsymbol{\sigma}} = \boldsymbol{C} : (\hat{\boldsymbol{\varepsilon}} + \hat{\boldsymbol{\varepsilon}}) = \boldsymbol{C} : \boldsymbol{\varepsilon} \\ &\boldsymbol{\varepsilon} = \hat{\boldsymbol{\varepsilon}} + \tilde{\boldsymbol{\varepsilon}} = \tfrac{1}{2}\left(\nabla(\tilde{\boldsymbol{u}} + \hat{\boldsymbol{u}}) + (\nabla(\tilde{\boldsymbol{u}} + \hat{\boldsymbol{u}}))^T\right) = \tfrac{1}{2}\left(\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) \end{aligned}\right\} \text{in V} \tag{1.15}$$

$$\boldsymbol{\sigma}.\boldsymbol{n} = \boldsymbol{T} \text{ on } S_T \tag{1.16}$$

$$\left.\begin{aligned} \boldsymbol{u} &= \boldsymbol{U} \quad t > 0 \\ \boldsymbol{u} &= \boldsymbol{0} \quad t = 0 \end{aligned}\right\} \text{ on } S_U \tag{1.17}$$

An advantage of using the superposition method is that the separation of the FEM and DDD problems means that alterations can be made to the implementation of one, such as if an improved DDD code were to be used instead of the original, and the way the superposition method is used would remain unchanged. The use of this method is only limited by the ability to calculate the tractions and displacements caused by dislocations on the relevant surfaces.

1.5.2 Displacement Calculations

Hirth and Lothe derived expressions for calculating the stress and displacement field due to a dislocation loop [62]. These expressions have been developed into expressions for an arbitrary finite segment as required in a DDD code. There are well validated analytic expressions, such as the non-singular expression derived by

Cai et al.[51] for evaluating the stress field, $\tilde{\boldsymbol{\sigma}}$ associated with finite dislocation segments in an infinite medium. Furthermore, the corresponding tractions and FE nodal forces have been obtained analytically by Queyreau et al [63] which are ideally suited for implementation in DDD codes. Unfortunately, the expression for the displacement field, which was pioneered by de Wit [64], requires an integral over complete dislocation loops. Although it can be used for approximating some situations [65], it is unsuitable for implementation in a DDD code, where dislocations are discretised into finite segments, whose elastic fields are evaluated individually.

In 1985, Barnett improved on this method to obtain an expression for the displacement caused by an arbitrary triangular dislocation loop [66]. This is a particularly useful solution for discrete DDD, because all dislocation loops are represented as polygons and any N-sided polygon can be divided into N triangles, one for each segment.

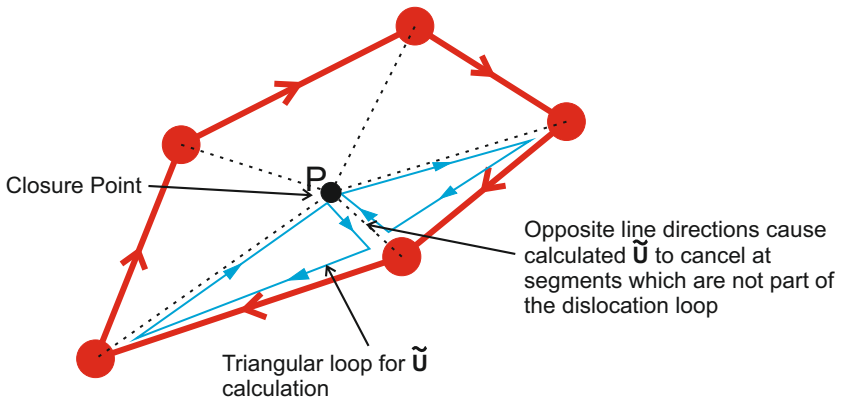


Figure 1.5: A simple dislocation loop divided into triangles for calculating displacement.

If continuity of line direction is maintained, then the displacement contribution from the sides of the triangles which are not part of the original dislocation loop will cancel, resulting in a correct calculation of the displacement when all segments are taken into account. Unfortunately, this method is far from perfect. A common "closure point" must be chosen for all segments so that the triangular loops formed cancel correctly. Mathematically speaking, it is appropriate to choose any point in the plane of the loop. Practically, this results in two problems. Firstly the loop must be planar and secondly the plane of the loop must be known. For simple shear loops this is the slip plane, the normal of which is the cross product of the line direction and Burgers vector. However, the slip plane is not well defined for screw segments so it can be difficult to define the closure point when constructing the displacement triangle for these segments. For a prismatic loop the habit plane of the loop is not the same as the slip plane anyway, since they

have “glide cylinders” instead, and for non-planar loops this method of finding the closure point fails completely. While it is possible to break up prismatic and non-planar loops into appropriate triangles, it once again becomes necessary to know the entire loop structure beforehand, which is problematic for DDD codes where calculations are performed piece-wise by segment.

To partially address these issues, Fivel and Depres [67] have developed an algorithm which breaks the dislocation network into triangular loops by generating a closure point for each segment such that screw dislocations do not contribute to the displacement. However the method requires projecting along the Burgers vector of each segment to find its closure point. When junctions have formed this is inconvenient as it requires storing the old Burgers vectors. Additionally, when applying this method to prismatic loops a correct answer is only obtained when two loops in the same slip prism are considered together such that the net Burgers vector of the combination is zero. To date, no method has been developed to consistently calculate displacement in 3D discrete DDD simulations for the completely general case. However, a new method to consistently calculate the displacement caused on the surface of a volume, and hence solve for the boundary conditions, will be outlined in this thesis.

1.5.3 Finite Volumes and Exiting Segments

In order to model a specific object with a finite volume, appropriate boundary conditions need to be applied as well as appropriate interactions between the simulated dislocation structure and any free surfaces present. The primary effects of concern are the image stresses caused by close proximity of a dislocation to a free surface, and the displacement caused when a dislocation exits through a free surface.

Image stresses occur when a dislocation approaches a free surface and can be described as a reflection of the stress field of the dislocation in the surface. Displacements can only be calculated for a closed dislocation loop. When taking these two things into consideration, Weygand et. al [68] proposed the use of virtual dislocations to handle the problem of dislocations exiting a free surface. The idea was to replace nodes exiting a free surface with a partial mirror of the dislocation structure on the outside of the surface which simultaneously simulates the image stress and closes the loop. Unfortunately, this approach has two major issues. The first is that the dislocation loop must extend to infinity to give the correct displacement at the surface when calculated using the Barnett method. This can be solved fairly easily though by changing how the virtual part of the loop is completed. The second is that the image stress is only simulated after part

of the dislocation network has already exited the surface, despite the fact that it becomes significant as the dislocation approaches the surface but before it exits. This method also requires the surface to be flat over a long range, which limits the applications it may apply to. The method developed by Liu and Shwarz [49] mentioned previously also suffers from this limitation.

Despite the problems with the Weygand method, virtual dislocations are still necessary to complete an exited loop so that displacements can be calculated. The solution is to create an arbitrary virtual dislocation loop which extends to "infinity" for which the stress terms are not calculated. This still leaves the matter of generating a correct stress field however. Recently, Celis Garza [find citation] has adapted the work done by Queyreau et al [63] to reliably emulate image forces. While this method requires the surface to be locally flat, this is the case when it is divided into elements, making it ideal for the superposition method.

1.5.4 Limitations and Avenues for Further Development

Despite being developed for over half a century, discrete dislocation plasticity models can still be considered to be in their infancy when comparing their current capabilities with what is ideally hoped they could accomplish. Continuum mechanics can be used to give a broad view of large length scales, whilst MD can provide information about the structure of individual dislocations. DDD can be used at the mesoscale to give a detailed picture of the behaviour of dislocation networks. The high complexity of the microstructures of real engineering alloys means that there is still much additionally desired functionality from these models. Advances are continually being made in this area. Recently, new work has be done at increased time and length scales using *ParaDiS* based code. Jogi and Bhattacharya [69] modeled the changes in dislocation networks at precipitate interfaces during high temperature creep in a $10\mu\text{m}^3$ volume. Furthermore, Lehtinen et al. [54] have created a multiscale model which combines MD with DDD to simulate dislocation-precipitate interactions at the micrometer length scale. Most microstructural features of alloys such as grain boundaries, grain shapes, precipitates, cracks and complex crystal geometries have all had basic models constructed to try to predict their effect on dislocation behaviour. However these models do not cover all aspects of these effects. The challenge now is to improve these models so that they can be applied more generally across a wider variety of possible scenarios. Combining aspects of different models is also necessary in order to observe the interplay between various effects as they would occur in real engineering alloys. The final goal should be to increase the size of volumes that can be simulated, as well as the detail and complexity of the models, so that a

wider variety of entire real objects can be modeled. Multiscale models are likely to be the best way towards achieving this. This thesis details work done towards this end, improving a code adapted from *DDLab* called *EasyDD* for the purpose of improving the functionality and performance of mesoscale simulations so that they may be applied to a wider range of purposes. *EasyDD* is coupled with a custom written FEM code and utilises GPU acceleration [70] to enable accurate simulations on a desktop PC. The primary aim of this thesis is to further expand the capabilities of the *EasyDD* code by implementing new solutions for various issues and to demonstrate the code's effectiveness by simulating a real mechanical test.

Chapter 2

Basic Improvements to *EasyDD*

2.1 Introduction

In order to improve the performance and functionality of the *EasyDD* code, the mechanisms behind the operation of the original *DDLab* code must be understood and refined. We have improved various aspects of the code so that it can better handle larger and more complicated simulations and to make it easier to add further functionality with the addition of new code.

2.2 Topological Operations

For any dislocation dynamics simulation, it is necessary to produce an acceptable approximation of the dislocation structure. In *DDLab*, dislocations are represented as a series of nodes connected by straight segments. It is thus essential that topological changes are performed correctly so that this approximates the more complicated curved structure to reflect real materials to a suitable degree of accuracy. There are three processes in the original *DDLab* code that do this:

- Collision detects when segments should collide and modifies the structure to connect them.
- Separation detects when a node has too many segments attached to it and splits it up into more nodes with fewer connections accordingly.
- Remeshing ensures that the structure approximates the nature of dislocation lines within set parameters, so that there are enough segments to get a result with an acceptable level of accuracy whilst not having so many that run-time requirements become detrimental.

A cursory examination may make it seem like these three processes are independent of each other but the truth is that they form a complex synergy, which

can cause problems that may make running even basic simulations impractical if the ways that these processes affect each other are ignored.

2.2.1 Remeshing

The remeshing process is required for even the most basic simulation of a single expanding loop. There are four input variables used to control the remeshing. They are the minimum segment length, maximum segment length, minimum area between two segments and maximum area between two segments (l_{min},l_{max},A_{min} and A_{max} respectively).

The variable l_{min} is used to define the resolution of the model. The smaller it is, the finer the detail that the model can simulate. For purposes of force calculations, there is a minimum possible segment length for which the equations are valid. This means the following inequality **must** be satisfied:

$$l_{min} > 2\frac{r_c|\boldsymbol{b}|}{a} \tag{2.1}$$

where r_c is the dislocation core radius, $\boldsymbol{b}$ is the Burgers vector and a is the lattice parameter. The core radius need not be a physical distance but instead can be tuned to optimise performance, as it is only needed to keep the segment-segment force calculation non-singular, as outlined by Cai et al. [51]. The value used for r_c typically lies between 1 and 10$|\boldsymbol{b}|$. If the condition in (2.1) is not satisfied, then a segment of length l_{min} will exist almost entirely within the volume designated as the cores of any connected segments. The calculation of the force on this segment will thus be highly inaccurate and may lead to nodal velocities that disrupt the progress of the simulation or create non-physical dislocation structures. It may be desirable to have l_{min} significantly larger than this minimum value though, since it will reduce the total number of nodes and segments in the simulation, hence reducing the total number of calculations needed. Naturally this comes at the expense of the accuracy of the model, since fewer straight segments of longer length approximate a continuous curve less well.

The variable l_{max} is necessary to ensure there is a sufficient number of nodes in the network. If l_{max} is too large, then there are too few nodes in the system and it becomes too inaccurate an approximation, while creating a phenomenon of "rigidity" whereby segments require a greater stress applied to them in order to move. Additionally, when coupling with FEM, it is undesirable to have a segment pass through too many elements and reducing l_{max} can prevent this. If l_{max} is too small on the other hand, then this increases the total number of segments unnecessarily. Since refining the network involves halving the length of segments deemed to be too long, the minimum value of l_{max} must be **at least**

twice that of l_{min}. However, using this lower bound as the actual value means that refining can only be performed when a segment is of length l_{max}, making it more difficult to control the change in curvature of a dislocation loop. It also means that any coarsening action will be immediately refined, reducing the capability of the code to optimise the number of nodes. Instead, it is recommended to make l_{max} significantly larger than l_{min}, by a minimum of a factor of ten.

The variables A_{min} and A_{max} are used to control how well the series of straight segments approximate a curve. They are equivalent to an *angle per unit length*. This means that as an angle between two segments gets sharper, the segments are shortened to more accurately capture the curvature. Conversely, as the angle between two segments increases, the segments are allowed to become longer as it and its neighbouring segments can be more closely approximated as a straight line. The two variables A_{min} and A_{max} together control how frequently and to what degree dislocations are remeshed as sections of the network change curvature. Naturally A_{max} must be larger than A_{min}. At least a factor of 4 difference is recommended so that the coarsening is not immediately triggered if two adjoining segments of length l_{max} are refined to reduce their area. Additionally, A_{min} must be greater than $2l_{min}^2$, otherwise the refining of smaller segments as a result of a change in curvature will not be smooth, as will be elaborated on later.

The original code used these four variables to form conditions under which the network should be remeshed as outlined by Bulatov and Cai [52]:

- If a segment is longer than l_{max}, divide it into two by creating a new node at the midpoint of the segment
- If a segment is shorter than l_{min} then merge it with an adjoining segment
- If two segments form a triangle with area larger than A_{max}, refine the longer segment into two shorter segments unless this would create a segment with length less than l_{min}
- If two segments form a triangle with area less than A_{min} and the nodes are moving so that the area is decreasing, coarsen the structure by merging the node they share with one of the other nodes

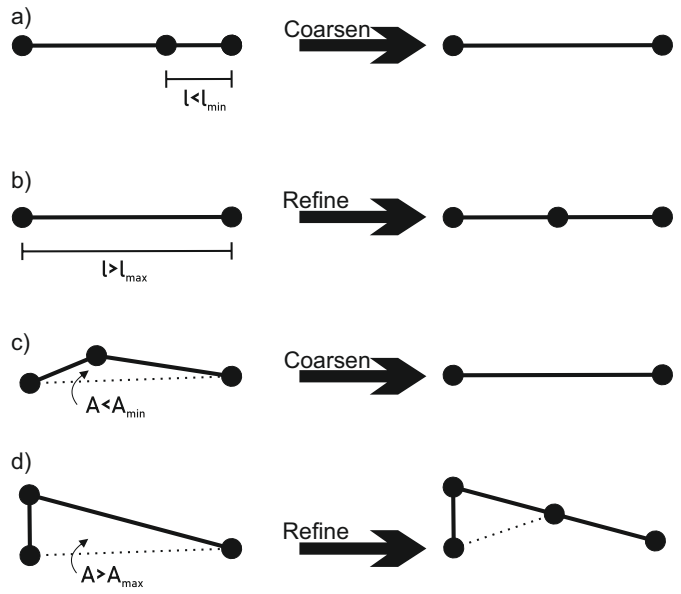


Figure 2.1: Some basic examples of the standard remeshing system. In (a) a segment shorter than l_{min} is removed by coarsening; (b) a segment longer than l_{max} is split into two by refining; (c) two segments forming an area less than A_{min} are combined by coarsening; (d) one of two segments that form an area greater than A_{max} is split into two by refining to form a smaller area.

For simple structures, these remeshing rules work well. However there are issues that arise as the network becomes more complicated. Before this can be discussed, the other processes affecting topological changes must first be outlined.

2.2.2 Collision

The main variable that controls when segments collide is the annihilation radius, r_{ann}. This is an input variable representing the distance that two segments must be from each other for their collision to be deemed inevitable if the two segments are moving towards each other. There are two modes of collision, designated in the code as "two-line" condition and "hinge" condition.

Collision handling for the two-line condition is performed by checking the list of segments for segments that are not connected to each other but are within r_{ann} of each other and then comparing the velocities of their respective nodes. If the distance between two segments is decreasing, then a collision is performed. First the points on each segment that are closest to the other segment must be identified and a new node is created at each of these points. These nodes are then merged to form the new collided structure, as shown in Figure 2.2. If the formation of one of these nodes would form a segment of length less than l_{min} then the node on the respective segment closest to the collision point is designated as the node to be merged instead.

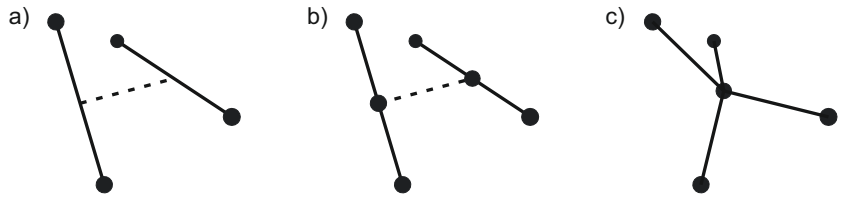


Figure 2.2: The two-line collision (in 3 dimensions): (a) Two unconnected segments move to within r_{ann} of each other; (b) New nodes are generated at the points on each segment closest to the other segment; (c) The two new nodes are merged, completing the collision.

The second mode of collision is the hinge condition. In actuality, only the two-line condition is a true collision of dislocations. The hinge condition primarily serves to allow for smoother junction zipping by allowing for the merging of two segments connected to a node with more than two connections. It is included in the collision handling rather than the remeshing because detecting the conditions under which hinge collisions occur is most conveniently done using the collision checking function. However, for hinge collisions, rather than considering the minimum distance between segments, the distance between nodes and segments is considered instead, as shown in Figure 2.3. Imagine Node 2 in Segment $\overrightarrow{23}$ that is less than r_{ann} away from Segment $\overrightarrow{34}$, which is connected to Segment $\overrightarrow{23}$ and another segment by Node 3. If $\overrightarrow{23}$ and $\overrightarrow{34}$ are moving closer together, then a new node for merging is created on $\overrightarrow{34}$ as for the two-line collision and the node on $\overrightarrow{23}$ is merged into this new node.

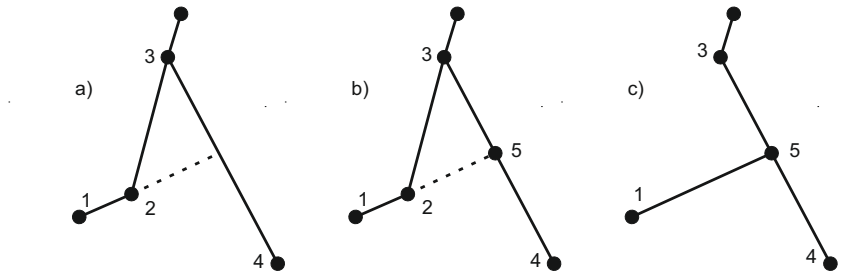


Figure 2.3: The hinge collision: (a) Two connected segments, $\overrightarrow{23}$ and $\overrightarrow{34}$, that are close together so that the end of $\overrightarrow{23}$ is within r_{ann} of $\overrightarrow{34}$; (b) A new node, 5, is generated on Segment $\overrightarrow{34}$ at the point closest to Node 2; (c) Node 2 is merged into Node 5.

The collision handling framework works well at dealing with individual collisions and is the best tool available for processing events that fit the hinge condition. Unfortunately, the basic method for checking collisions is problematic when applied to larger models. The order in which collisions are performed was originally determined by the order they were listed in memory, assuming that the order collisions are performed in is irrelevant. This ignores some details of the actual

structure of the network and can cause collisions to be performed in non-physical ways, especially when there is a dense dislocation structure. The way to avoid this is to first detect all potential collisions before performing any, and then perform collisions in order starting with the colliding segments that are the closest together. This change was implemented with the aid of the coding expertise of H. Yu and D. Celis Garza and can be found on the Tarleton group GitHub. This requires more computation but is advantageous in the long run since it prevents the formation of undesired dislocation structures that can cause the simulation to grind to a halt. Examples of such structures are miniature dislocation loops that continually collide with themselves, junctions with non-physical $\boldsymbol{b}$ or segment line directions and nodes that are highly connected and act as obstructions to motion. A shortcut can be used for hinge collisions where it is ensured that the segment designated as a single node is the shorter of the two , as shown in Figure 2.4. This change was implemented in the *EasyDD* with the aid of D. Celis Garza and can be found on the Tarleton group GitHub.

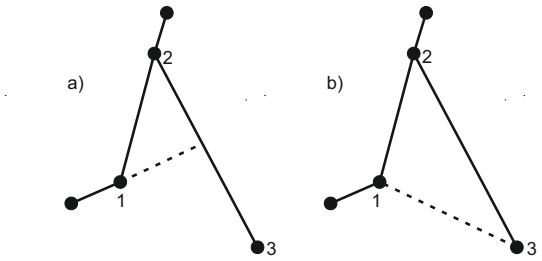


Figure 2.4: The ends of both segments may be within the collision radius of the other segment. (a) shows the shortest distance from node 1 to segment $\overrightarrow{23}$ while (b) shows the shortest distance from node 3 to segment $\overrightarrow{12}$. Merging node 1 into segment $\overrightarrow{23}$ would be the correct operation to perform, since the distance from this node to the segment is shorter.

Another consideration that may need to be addressed is the phenomenon of so-called super dislocations. This is where two segments with the same Burgers vector, $\boldsymbol{b}$, collide to create a new dislocation segment with $\boldsymbol{b}$ twice that of the original segments. This is not something that occurs physically, since the repulsive force between such dislocations increases inversely with the distance between them. Non-physical collisions may occur in simulations as a result of physical phenomena, such as in a pile up, because the code interprets the situation as one that requires a collision. The leading dislocation in an array emitted by the same source may be moving more slowly than the one behind it, for example, and the code would detect this as a viable collision condition once the two dislocations are within r_{ann} of each other. This essentially becomes a resolution issue, with the code being unable to differentiate dislocations that remain too close to each

other for multiple time increments. Super dislocations can be advantageous since they reduce the memory and processing requirement for larger scale simulations by merging multiple dislocations into one. However there is currently no method for separating them again so it may be desirable to prevent them from forming altogether. To do this, the collision checker must identify colliding segments with the same Burgers vector or negative Burgers vectors. In situations without climb, it can be assumed that dislocation segments with the same or opposite Burgers vectors are close to co-planar at the point where a collision will occur, since even if they have different slip planes it is geometrically necessary for them to collide along the line where these slip planes intercept. This means that relative orientation of segments can be used to decide whether a collision is viable due to limited possible geometries. Segments with like Burgers vectors can be permitted to collide when such segments are anti-parallel since this will result in an annihilation event. For Burgers vectors of opposite sense this is true when the segments are parallel. Super dislocation formation can thus be reduced by disallowing collisions where the Burgers vectors are the same and the angle between the the line direction is between 0° and 90° or 270° and 360°. For opposite Burgers vectors this regime is between 90° and 270°. This is an appropriate method for all situations where a preexisting node is not involved in the collision. Since a preexisting node is linked to two segments, the nature of the junction that would form when the collision occurs must be taken into account with more complicated geometric factors involved, since the problem can no longer be reliably approximated to two dimensions and more than two segments will be involved. Given the rarity of such events, the complete fix has yet to be implemented and would involve comparing the directions of all segments involved.

By itself, collision will rarely form a complete junction. In most instances, it will form a node with 4 or more arms. Separation is therefore required to complete the process.

2.2.3 Separation

Any node with 4 or more segments, also known as “arms”, connected to it can be split into nodes with fewer arms until all nodes have 3 arms or fewer. For the original *DDLab* and *ParaDiS*, a convenient method of maximising power dissipation was developed, which only requires information about the dislocations directly connected to the node of interest. The way this is done, as outlined by Arsenlis et al. [71], is to find all of the ways a node can be split, or “splitting modes”, and then to calculate the power dissipation of each splitting mode. The network is then updated to the state with the greatest power dissipation:

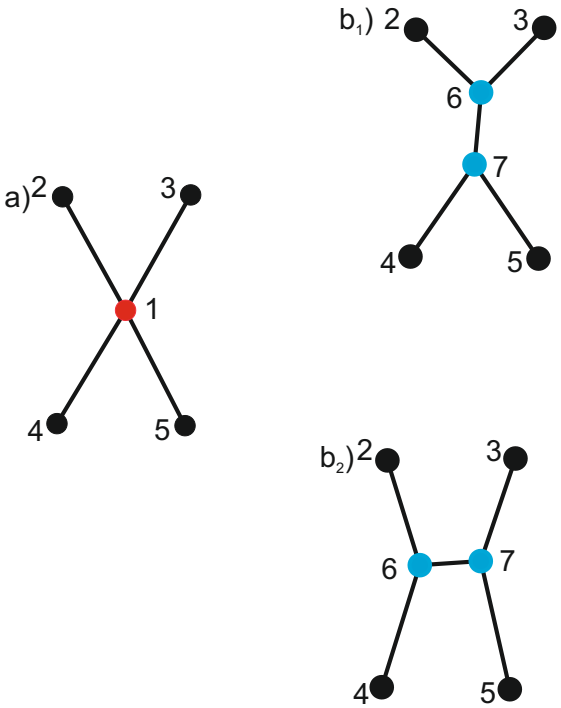


Figure 2.5: How a node may be split. In a) node 1 has 4 arms and must be split. Two possible splitting modes are shown in b_1) and b_2), where node 1 has split into two new nodes, 6 and 7, but with different connectivity in each example.

$$P = \sum_{i}^{j} \boldsymbol{v}_i \cdot \boldsymbol{f}_i$$
$$\Delta P = P_1 - P_0 \tag{2.2}$$

where P is the power calculated in a system, $\boldsymbol{v}_i$ and $\boldsymbol{f}_i$ are the velocity and force on node i respectively, j is the total number of nodes in a system, P_0 is the power in the system before splitting, P_1 is the power in the system after a proposed split and ΔP is the power dissipation caused by the split. A splitting mode is created by assigning each of the arms attached to the node that is being split to one of the two new nodes created by the splitting. The velocities of these new nodes can then be calculated to find the line direction of the segment required to connect them. This is shown in Figure 2.5.

For a node with N arms, the number of splitting modes, C, can be calculated by:

$$C = \sum_{n=2}^{\lfloor N/2 \rfloor} \frac{N!}{(N-n)!n!} \tag{2.3}$$

When N is even, some of these splitting modes are equivalent. The number of unique splitting modes, C', can be found by:

$$C' = C - \frac{N!}{2\left(\frac{N}{2}!\right)^2} \tag{2.4}$$

Equation 2.3 shows that the number of splitting modes increases incredibly quickly as N increases. This is also only for the first split, since any new nodes created that have more than 3 arms will subsequently need to be split. As N increases, the number of splits necessary to completely break up the node into nodes with only 3 arms also increases. The power dissipation of each final state, where all nodes have 3 or fewer arms, very quickly becomes impractical to calculate. For example, a node with 6 arms has 25 unique splitting modes for the first split. However, when the splitting of the subsequent nodes is accounted for, there are 540 possible end configurations where all nodes only have 3 arms. For a node with 7 arms, these numbers change to 56 for the initial split and 14490 possible end configurations. To reduce the number of calculations, the separation function only calculates the power dissipation for the initial split and then performs that split before continuing the process with the newly formed nodes. For a 6 armed node, this reduces the total number of calculations from 540 to either 31 or 38, depending on the initial splitting mode. In terms of computational efficiency, this method is convenient, although it has two possible major drawbacks. The first is that it assumes the power dissipation pathway to the configuration with the greatest possible power dissipation is always through the route with the most power dissipation of any given split, which may not necessarily be true. Secondly, nodes with many arms tend to have low velocities, to the point where their direction after they are normalised can be affected by numerical imprecision. This means that the line direction of the formed connecting segment may be incorrect. The degree to which this affects the separation of nodes with many arms is difficult to establish as it involves running trillions of calculations to compare the outcomes for the splitting of a node with just 9 arms. Fixing these issues by altering the current separation algorithm is also impractical since it involves greatly increasing the complexity of the code and the number of calculations that need to be made. Fortunately, the synergy with other topological operations can be used as a means to avoid the formation of many armed nodes in the first place.

2.2.4 Operation Synergy

The main cause of the formation of many armed nodes is repeated collisions during a single time increment. This occurs due to the way the code runs the topological operations. The basic code performs all collisions, then all separations and then all remeshing operations, in that order, before starting the next time increment. Combining the collision and separation functions can avoid the formation of many armed nodes if the separation function is called after every individual collision. This is acceptable from a physical standpoint, since in reality dislocations tend not to collide to form connections at discrete points anyway. Instead, sections of dislocation lines are often involved when collisions occur, forming junctions. Immediately forming the junction structure after a collision operation mimics this behaviour. However, to perform this action properly, several precautionary measures must be taken. Firstly, the previously mentioned alteration to the collision code to enforce the correct order of collisions must be made. Any junction formed by the separation code will be able to take part in further collisions, so the correct junction must be formed and then collided in the correct way to ensure the resultant structure is a suitable representation. Secondly, it must be ensured that any collision that occurs will not be immediately undone by the separation function as this will create an infinite loop. In theory, a properly constructed collision function should avoid all potential collisions that would cause this. In practice, the only way to guarantee such conflict does not occur is to add a power dissipation check to the collision handling. If a collision would form a node that would immediately be split back into its previous state, do not perform that collision.

Collision handling can also cause conflicts with remeshing. This is for two main reasons. The first is that the selection of merging nodes for the hinge condition is the same as for two-line condition. This can cause the conflict shown in Figure 2.6.

In order to resolve this conflict, the hinge node that both colliding segments share, node 3 in Figure 2.6, must be forbidden from being directly merged with the colliding node. To compensate for this, another collision point must be found instead. A new collision point can be designated as a point l_{min} away from the hinge node along the line direction of the colliding segment, connecting nodes 3 and 4 in the figure. If generating a new node here would create a segment shorter than l_{min}, then the node in the segment that is not the hinge node should be designated as the node to be merged, which is node 4 in this example.

The second source of conflict between collision handling and remeshing is because hinge collisions have essentially been used as an additional remeshing step, but the conditions by which they are triggered are not the same as for remeshing.

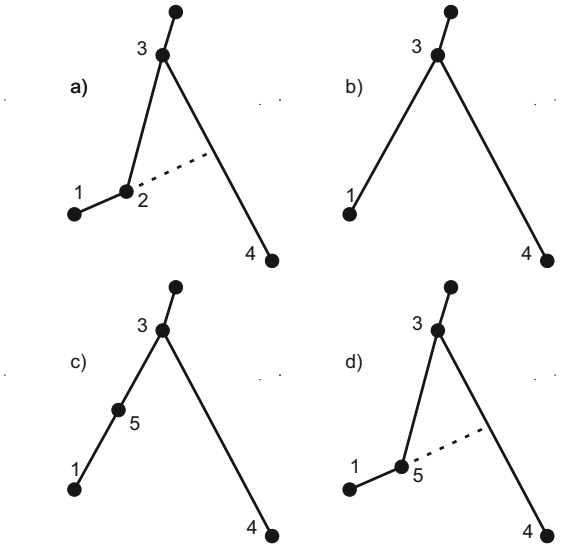


Figure 2.6: Conflicts between remeshing and hinge collisions can result in repeating actions: (a) the designated collision point is too close to node 3 so (b) node 2 is merged into node 3. (c) Remeshing refines the network by adding a new node, node 5, to segment $\overrightarrow{13}$. (d) Node 5 is moved in the following time increment, recovering the previous structure by replacing node 2, generating an infinite loop.

This means that the same collision can be repeated over multiple time increments as it is subsequently undone by the remeshing, and vice versa. Hinge collisions and remeshing can be relatively simply separated so that they do not directly overlap. Mesh coarsening only involves nodes that have 2 arms, so collision handling can be updated to only use nodes with 3 or more arms as hinge nodes. However, making this alteration necessitates changing remeshing to compensate. To understand this, we must look at the area, A, formed by two segments, $\boldsymbol{p}$ and $\boldsymbol{q}$, with an angle between them, θ.

$$A = \frac{1}{2}|\boldsymbol{p}||\boldsymbol{q}| \sin\theta \tag{2.5}$$

From this equation, we can see that A is largest when $\theta = 90°$, and subsequently decreases as the angle changes from 90°, being at a minimum of 0 when $\boldsymbol{p}$ and $\boldsymbol{q}$ are either parallel or anti-parallel. For the sake of simplicity, consider an example where the lengths of $\boldsymbol{p}$ and $\boldsymbol{q}$ are the same. We can consider the area as a function of θ and see how the remeshing algorithm handles the segments as the angle between them changes. In the original code, as θ changes the remeshing regimes for $\boldsymbol{p}$ and $\boldsymbol{q}$ change as follows in Figure 2.7:

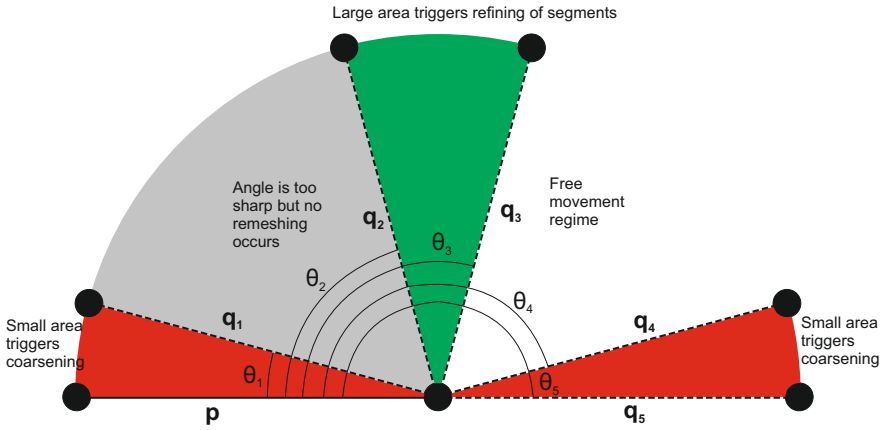


Figure 2.7: The original remeshing regimes for two connected segments, $\boldsymbol{p}$ and multiple potential orientations of $\boldsymbol{q}$, with lengths less than l_{max} and greater than $\sqrt{2A_{max}}$. The largest angle, θ_5, is 180° since angles after this up to 360° just produce a reflection of the regimes shown.

Figure 2.7 shows at which angles remeshing occurs for the specific segments $\boldsymbol{p}$ and $\boldsymbol{q}$. The area between them is less than A_{min} between angles of 0 and θ_1. The same is true between θ_4 and θ_5. This causes coarsening to occur, where the shared node depicted at the center is merged into one of the other two. The area between $\boldsymbol{p}$ and $\boldsymbol{q}$ is larger than A_{max} between θ_2 and θ_3, which will trigger refining, causing a new node to be created on either $\boldsymbol{p}$ or $\boldsymbol{q}$. These are the only regimes in which the remeshing is triggered, leaving two sets of angles that do not trigger remeshing. One, when the angle between $\boldsymbol{p}$ and $\boldsymbol{q}$ is between θ_3 and θ_4, is the expected regime for no remeshing to occur, where the segment lengths are considered a suitable approximation to a curve. In the last regime, when the angle is between θ_1 and θ_2, it is inappropriate for no remeshing to occur since the angle between the segments is sharp enough that we would want remeshing to occur in order to better approximate a curve. However, no remeshing is performed since the area between $\boldsymbol{p}$ and $\boldsymbol{q}$ is still less than A_{max} and larger than A_{min}.

It becomes immediately apparent that the original remeshing works exactly as desired for obtuse angles, but is not right for acute angles. This is not an issue for simple dislocation structures where everything is made up of basic dislocation loops, since the rate of change in curvature occurs such that all angles between segments are greater than a right angle. However, as networks become more complex or obstacles are introduce, dislocations become more prone to sudden changes in direction to avoid obstructions to their motion. The ideal remeshing for the above setup would be as shown in Figure 2.8:

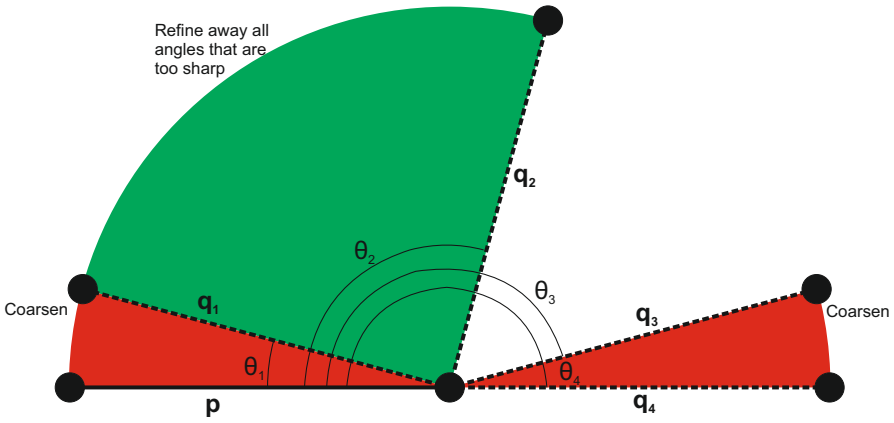


Figure 2.8: The ideal remeshing regimes for two segments, $\boldsymbol{p}$ and all potential orientations of $\boldsymbol{q}$, with lengths less than l_{max} and greater than $\sqrt{2A_{max}}$. At angles less than θ_2, segments are always remeshed.

Figure 2.8 shows a new remeshing method. For obtuse angles, it operates exactly the same as the old method. However for acute angles it allows remeshing to occur as necessary to properly approximate a curve.

If the dot product of the two vectors is positive, we can design the function so that it always refines segments if it does not have to coarsen. This coarsening is necessary to remove segments that bend back on each other, essentially occupying the same space so that they should be annihilated, as well as preventing many small segments forming a saw-tooth type shape where a single straight segment would make an acceptable approximation. This change to the remeshing works unless the following condition is met:

$$\frac{|\boldsymbol{p}||\boldsymbol{q}|}{2} < A_{max} \tag{2.6}$$

In this situation the maximum area two segments can make is less than A_{max}, which is the point where the old method would stop using refining to approximate a curve. Using the new method at this point means that if the angle between the two segments is acute, the structure will be continually refined until either the segments are too short to remesh or the area becomes less than A_{min}, since in the current proposed form refining always occurs for acute angles unless coarsening is occurring. This results in more remeshing occurring than necessary and many more nodes and segments being created than required, both of which will impact performance. Thus an additional modification of the remeshing is required.

When the condition shown by inequality 2.6 is met, the angle at which refining is triggered should decrease as segment length decreases. A typical remeshing setup for this regime would be as shown in Figure 2.9:

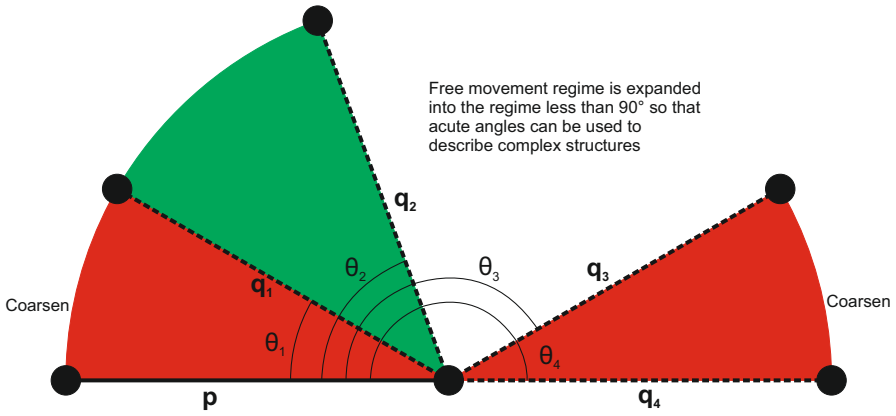


Figure 2.9: The ideal remeshing regimes for two segments, $\boldsymbol{p}$ and several potential orientations of $\boldsymbol{q}$, with lengths less than $\sqrt{2A_{max}}$ and greater that $2l_{min}$.

We can define a new set of conditions for refining to occur:

$$\begin{aligned} \frac{|\boldsymbol{p}||\boldsymbol{q}|}{2}(1 + \Gamma \cdot \cos\theta) &> A_{max} \\ |\boldsymbol{p}||\boldsymbol{q}| \sin\theta &< A_{min} \end{aligned} \tag{2.7}$$

Here, Γ is a necessary factor used to control the smoothness of the change between remeshing regimes. We can derive what it must be fairly simply. Since refining must stop when segments are too short to be halved anymore (i.e. their length is less than $2l_{min}$), we can use this as a calibration point, giving us the following two equations:

$$\begin{aligned} 2l_{min}^2 \sin\theta_c &= A_{min} \\ 2l_{min}^2(1 + \Gamma \cdot \cos\theta_c) &= A_{max} \end{aligned} \tag{2.8}$$

We can then rearrange to find $\sin\theta_c$, use $\cos^2\theta + \sin^2\theta = 1$ to find a term for $\cos\theta_c$ and substitute to solve for Γ. This gives us:

$$\Gamma = \frac{A_{max} - 2l_{min}^2}{\sqrt{4l_{min}^4 - A_{min}^2}} \tag{2.9}$$

Another small addition that needs to be made to the remeshing is that if two dislocations form a hairpin such that their cores overlap for the entirety of both of their lengths then they should be coarsened away regardless of other factors. This is because such dislocations would have merged already in a physical scenario and this fact must be enforced. The modified remeshing is demonstrated in Figure 2.10

Now that we have a remeshing system that works exactly as desired, we need

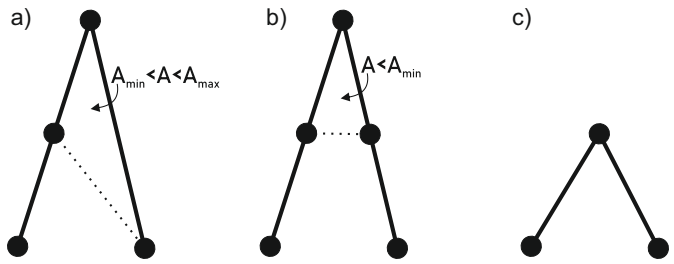


Figure 2.10: An example of how the modified remeshing functions. First, (a) a sharp angle is formed by two dislocations but the area between them is less than A_{max} and more than A_{min} so the original remeshing would ignore this scenario. The modified remeshing detects this situation resulting in (b) where a new node has been added through refining. This then allows the angle to be removed if necessary by coarsening, leaving (c) as the final situation with and acceptable approximation of curvature.

to reconsider sensible values for A_{min} and A_{max}. The smallest change in curvature is defined by A_{min}. It can be thought of in two ways. The first is the smallest loop we want to be able to resolve in our simulation. This should be some regular polygon with side lengths of l_{min}. The triangle formed by two sides gives the area we are interested in. Alternatively, we can consider two joined segments of length l_{max} forming a straight line. If we define a minimum resolvable distance for the central node to move and use that to form the height of a triangle with two sides of l_{max}, the area of which is a candidate for A_{min}. It is best to use both methods and then pick the smaller of the two answers so that all of the features we are concerned with can be resolved.

The definition of A_{max} allows for control over when nodes are added to the structure. This is is not only necessary to approximate a curve well, but also to display complexity of a structure in an area of high localised force, since more nodes allow for more complicated shapes. Optimising A_{max} is more complicated. We must consider how the structure will evolve and then assign a suitable A_{max}, even though it may be at the detriment of simulation time. For example, if we are trying to model an extensive network produced by relatively small Frank-Read source, A_{max} must be small enough that new nodes are created relatively soon after bowing of the source has occurred to allow for smooth growth of the loop, even though this means that the loops produced may contain more nodes than necessary once they have grown to a substantial size.

2.3 The Time Integrator

2.3.1 The Quasi-Static Framework

The primary mathematical basis for DDD is the quasi-static framework. Theoretically, this means that the system should be allowed to reach a natural equilibrium that occurs due to the applied conditions before these conditions are changed. In practice, so long as conditions are changed suitably incrementally then the quasi-static framework is still considered valid. There are several aspects of a DDD code where this must been kept in mind to ensure the validity of the simulation.

The greatest effect of the quasi-static is the necessity for all nodes in a simulation to be maintaining constant velocity (i.e. having zero acceleration). This means that the resultant force on any given node is zero. Thus the forces acting on any node to cause it to move must be exactly balanced by those inhibiting motion:

$$\boldsymbol{f}_{app} = -\boldsymbol{f}_{drag} \tag{2.10}$$

Here, $\boldsymbol{f}_{drag}$ is the drag force on a given node as a result of the segments it connects and $\boldsymbol{f}_{app}$ is the applied force. The applied force may be a combination of various forces, such as the Peach-Koehler force as a result of the applied stress, mirror forces from a free surface or interaction forces between segments. Any of these may play a significant role in dislocation behaviour. This balance of forces is a key component without which it would be impossible to calculate the velocities of the nodes in a any mobility law, which will be discussed in further detail in Chapter 3. In order for the mathematical validity (and therefore overall accuracy) of the model to be maintained, the change in any of the forces on a given dislocation segment between time increments must be small. To envisage why, it is best to consider the process of the evolution of a dislocation structure. In reality, a given part of a network can be observed and its velocity recorded and plotted against time. When looking at one direction, this gives a curve, the gradient of which at any given point is the acceleration in the chosen direction at that particular time. However, as stated previously, DDD assumes that there is no acceleration at any given time. This means at the beginning of a time increment a single velocity is calculated and maintained for the duration of the time increment. At the next time increment, a new velocity is calculated and then maintained and so on for the entirety of the simulation. This means if velocity is plotted against time the figure will show a series of steps rather than a curve, as shown in Figure 2.11.

If a physical situation for which velocity/time data had been gathered were to be simulated using DDD, ideally these steps should follow the curve of the real data

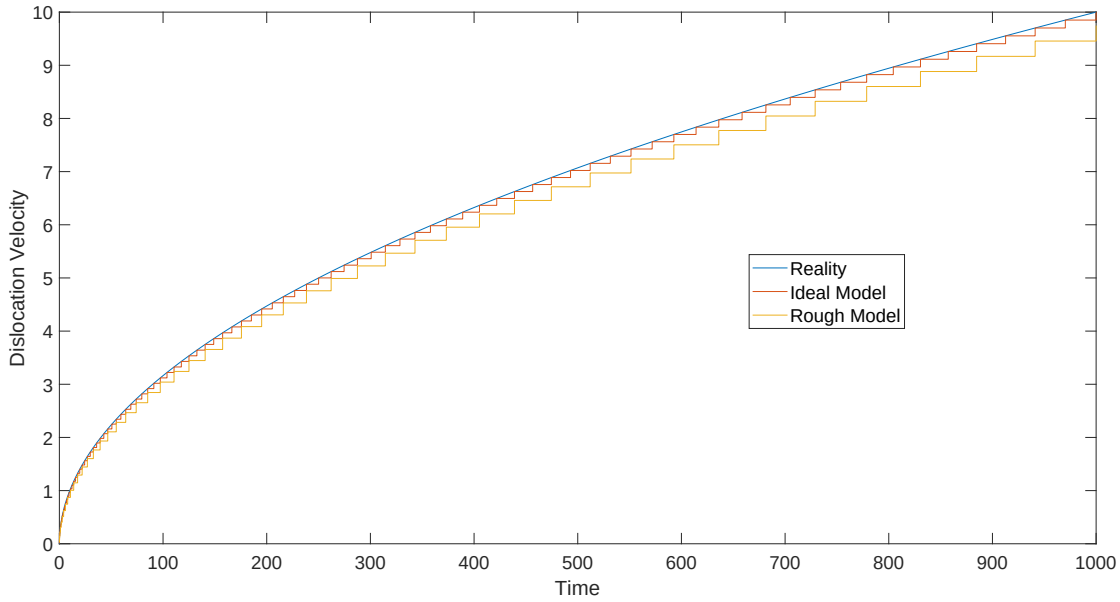


Figure 2.11: A hypothetical plot of how dislocation velocity changes with time, comparing reality with an idealised DDD model and a more poorly calibrated model (units are arbitrary). In reality, velocity changes continuously but in the model velocity is only calculated at the beginning of each time increment and then remains constant for the duration of an increment.

to give a good approximation. This means the duration of the time increment, Δt, must be controlled. Firstly this prevents the steps on the plot from appearing too different from the real curve so a good estimate of the acceleration can be obtained. Secondly, and more crucially, this prevents the structure from deviating too much from what it would be doing in reality so that the continued evolution is a realistic approximation. Consider a single growing shear loop in a constant stress field. As the loop expands, the line tension and forces between the opposite sides of the loop decrease, allowing it to move faster. This means there is an increase in velocity as distance from the original position increases. In DDD, because a constant velocity is maintained over a time increment, the distance a dislocation moves each time increment does not change as much as it does in reality. This means that the velocity calculated in the subsequent time increment will be an underestimate. This error will only grow after subsequent time increments. If time increments are longer, then the calculation of the final position of the loop at the end of the simulation is less accurate than if time increments are shorter. However if time increments are too short then the computational cost is too high. Thus a compromise must be reached, giving a result with an acceptable margin of error. If the acceleration is relatively small, then larger time time increments can be used but if the acceleration is large then smaller time increments **must** be used. In most simulations, the way the acceleration changes is extremely complicated. A time integrator is used to control the size of the time increment to ensure it is

appropriate.

2.3.2 Control Parameters

The time integrator is vital for proper functionality of any DDD code. With the possible exception of remeshing, no other component provides such a great effect on the balance between performance and accuracy. At its core, the time integrator is a prediction mechanism. It compares the positions and velocities of the nodes after a proposed time increment with their current positions and velocities. It then uses the difference between the two to decide whether the size of the time increment should be increased or reduced and then repeats the process. There are two parameters that can be used to judge whether the time increment is appropriate. The first is the furthest distance moved by any node in the simulation. This is simple to calibrate for, simply by directly comparing the current positions of the nodes with their predicted positions. If at least one node would move farther than the maximum allowed distance, r_{max}, then the time increment must decrease, i.e.:

$$\Delta t|\boldsymbol{v}_i| < r_{max} \tag{2.11}$$

The second parameter is the error in the position. To find this we can use a predictor-corrector method, where a second separate prediction of node velocities is made and compared to the original prediction. If the largest difference observed is larger than the maximum allowed tolerance, r_{tol}, then the size of the time increment must be decreased. Generally speaking, using the error in position is considered more stable than using the greatest distance moved. However there is a convincing argument for the use of both parameters simultaneously, which will be detailed here.

To understand how best to tune a time integrator, the specific predictor-corrector method used must be understood. The predictor-corrector method used in *EasyDD* is the Euler-trapezoid method, which is outlined by Bulatov and Cai [52], where an Euler forward predictor is used to and then controlled by a trapezoid corrector. This involves first calculating the forces on the dislocation segments and using this to find the resultant forces and velocities of the nodes at their current position and then extrapolating the nodal positions after a proposed time increment. This is the first prediction. The second prediction is made by extrapolating the positions to after **half** of the proposed time increment and then recalculating the forces and velocities. Another result for the position after the full time increment can then be extrapolated by proceeding through the second half of the time increment with these new velocities. The error between the two predictions can

hence be calculated as follows:

$$\begin{aligned}\epsilon &= \left| \boldsymbol{v}_0 \Delta t - \left(\frac{1}{2}\boldsymbol{v}_0 \Delta t + \frac{1}{2}\boldsymbol{v}_1 \Delta t \right) \right| \\ &= \frac{1}{2}|\boldsymbol{v}_0 - \boldsymbol{v}_1| \Delta t \end{aligned} \tag{2.12}$$

where ϵ is the error, $\boldsymbol{v}_0$ is the initial velocity, $\boldsymbol{v}_1$ is the velocity after half of one proposed time increment and Δt is the proposed time increment duration being tested. This error must be found for all nodes and if the largest error is larger than r_{tol} then the proposed time increment is halved. Effectively this means that the next time increment tested is that just used as a comparison to find the error. At a glance, it seems like this is looking at the acceleration or deceleration of the nodes in the simulation, but it would be more insightful to consider it as an estimate of how smoothly the dislocations are moving. If there are fluctuations in the overall stress state of the material then the forces on segments and hence the velocities of nodes will in turn fluctuate more over a given time period as they move through the material, so the size of a time increment should be smaller to capture the nuances of this more complicated behaviour. Figure 2.11 shows an idealised example of time integrator operation, with the size of time increments increasing as the rate of change in velocity decreases. The optimum time increment size is the largest possible time increment for which the error is less than r_{tol}.

The error provided by the predictor is an important piece of information for deciding the size of a time increment, however other factors should be taken into consideration as well. One of these is the total distance a given node travels. Segments interact and need to be remeshed as the dislocation structure evolves. A limit must be put on the furthest distance any node can travel during one time increment to ensure that collision, separation and remeshing can occur with sufficient frequency. In the original *DDLab* Euler-trapezoid time integrator, r_{max} was not used as a control parameter. This is likely due to the assumption that any interactions dislocations may be involved with will result in a change in the stress state that will increase the error measured by the predictor-corrector. With the addition of an enforced maximum time increment size, Δt_0, this can provide a soft limit on how far nodes can move during a time increment. However there are two problems with this. The first is that Δt_0 must be made relatively small to stop nodes from moving too far when they are moving fast, which means small time increments are taken even when the dislocation structure is not evolving, greatly increasing computational cost. The second problem is that the error does not **always** detect interactions. Multiple factors effect the force on dislocation

segments and happenstance may result in two changes in the dislocation structure happening simultaneously in a way that partially cancel out the forces on a segment, so that the error measured does not indicate an imminent collision, for example. This means there is potential for some behaviour to not be captured and for non-physical structures to form. There are multiple ways to fix these issues, such as using a more sophisticated predictor-corrector. Using r_{max} as a control parameter is arguably the simplest, and it is intuitive to calibrate since it can be directly compared with the collision radius r_{ann}, as will be discussed later.

The other key factor that controls Δt to consider is the actual process of finding a suitable Δt. We have described how it is deemed whether or not Δt is appropriate, but this requires testing multiple possible values, which involves an iterative process. Once a predictor technique has been selected, this process is fairly straight forward: Test a potential time increment and if it is unsuitable then decrease it, otherwise increase it, then test the new possible time increment. There are four key variables that control the effectiveness of this loop. The first is the time increment size tested in the first iteration. It is most common to start with the Δt chosen for the previous time increment. If it is the first time increment of a simulation then Δt_0 can be used instead. The second variable is how much to decrease a time increment by if the tested one is unsuitable. As mentioned previously, the Euler-trapezoid method works well when the time increment is halved. This leaves the final two variables: how much to increase a suitable time increment by and how many iterations to perform before choosing a time increment. Suitable time increments must be increased by less than the factor unsuitable time increments are decreased by, so that multiple iterations move closer to the optimum value for Δt. The number of iterations performed can be calibrated for the function of a given time integrator, so that multiple iterations with little improvement are taken, since this will increase computation time. Both of these variables are independent of other aspects of the simulation and can be hard coded into the integrator rather than be input parameters. With this is mind, we can now discuss how to select suitable values for all of the parameters mentioned.

2.3.3 Tuning the Time Integrator

There are many ways that the time integrator can be optimised and various interactions to consider. Bulatov and Cai [52] outline a very basic algorithm for a time integrator, which they readily admit has much room for improved efficiency.

```
Result: Advancement of the simulation by one time increment
t_0 = t// Record current time
Δt = Δt_1// Obtain Δt from last time increment
SegForce = CalculateForce(rn, links)// Calculate forces on all
 segments
vn = CalculateVelocity(rn, SegForce)// Calculate all node
 velocities
while t = t_0 do while time has not advanced
    (rn_0, links_0) = CalculatePosition(rn, links, vn, Δt)// Find node
     and segment positions after Δt
    (rn_1, links_1) = CalculatePosition(rn, links, vn, Δt/2)// Find node
     and segment positions after Δt/2
    SegForce_1 = CalculateForce(rn_1, links_1)// Find forces on
     segments after Δt/2
    vn_1 = CalculateVelocity(rn_1, SegForce_1)// Find node
     velocities after Δt/2
    (rn_1, ) = CalculatePosition(rn_1, links_1, vn_1, Δt/2)// Find node
     positions after two increments of Δt/2
    ε = CalculateError(rn_0, rn_1)// Find the maximum error between
     the two predictions
    if ε > r_tol then
        Δt = Δt/2// Decrease Δt if the error is too large
    else
        rn = rn_0// Update structure
        links = links_0
        if Δt = Δt_1 then // If Δt has not been decreased
            Δt_1 = min(1.2Δt, Δt_0)// Increase Δt next time
             increment up to Δt_0
        else
            Δt_1 = Δt// Use the same Δt next time increment if Δt
             needed to be decreased
        end
        t = t + Δt// Advance time by Δt
    end
end
```

Algorithm 1: Pseudocode of a possible implementation of the simple algorithm proposed by Bulatov and Cai.

They suggest increasing the time increment size by 20 percent once per time increment, essentially having the maximum number of iterations as 1. This was adapted into the Euler-trapezoid time integrator used in *DDLab*. For more efficient performance, a few alterations to the algorithm can be considered. First the factor by which the time increment is increased, Λ, can be varied depending on the the size of the error:

$$\Lambda = \Lambda_{max}\left(1 + \frac{\epsilon}{r_{tol}}(\Lambda_{max}^{n} - 1)\right)^{-1/n} \tag{2.13}$$

where Λ_{max} is the maximum factor we want to increase Δt by and n is an exponent used to control how Λ changes with ϵ. We can see that Λ varies between 1, when $\epsilon = r_{tol}$, and Λ_{max}, when $\epsilon = 0$. This system works well to reduce the number of iterations required to get to the optimum value, with values of $\Lambda_{max} = 1.2$ and $n = 20$ typically used, and even with the single iteration algorithm it improves performance by reducing the frequency with which the size of the time increment is halved. However, an algorithm that performs multiple iterations is more efficient for finding time increment sizes closer to the optimum. This was implemented in collaboration with D. Celis Garza and the full code is available on the Tarleton group GitHub:

```
Result: Advancement of the simulation by one time increment
ϵ_1 = r_tol// Set error for Δt_1
Δt = Δt_1// Obtain Δt from last time increment
SegForce = CalculateForce(rn, links)// Calculate forces on all segments
vn = CalculateVelocity(rn, SegForce)// Calculate all node velocities
i = 1// Set iteration counter
while i < maxiter do // While the maximum iteration has not been reached
    (rn_0, links_0) = CalculatePosition(rn, links, vn, Δt)// Find node and segment
     positions after Δt
    r = max|rn_0 − rn|// Find the maximum distance a node moves after Δt
    (rn_1, links_1) = CalculatePosition(rn, links, vn, Δt/2)// Find node and
     segment positions after Δt/2
    SegForce_1 = CalculateForce(rn_1, links_1)// Find forces on segments after
     Δt/2
    vn_1 = CalculateVelocity(rn_1, SegForce_1)// Find node velocities after
     Δt/2
    (rn_1, ) = CalculatePosition(rn_1, links_1, vn_1, Δt/2)// Find node positions
     after two increments of Δt/2
    ϵ = CalculateError(rn_0, rn_1)// Find the maximum error between the two
     predictions
    if ϵ < r_tol ∧ r < r_max then // If the error and maximum distance moved are
     acceptable
        Δt_1 = Δ_t// Record most recent usable Δt
        if Δt = Δt_0 then // If Δt is the maximum allowed value
            i = maxiter// Exit the loop
        else
            ϵ_1 = ϵ// Record the error of Δt_1
            i = i + 1// Increase the iteration counter
            Λ = CalculateFactor(ϵ, r_tol, Λ_max, n)// Find the factor to
             increase Δt by
            Δt = min(ΛΔ_t, Δt_0)// Increase Δt if the maximum iteration has
             not been reached
        end
    else
        if ϵ_1 < r_tol then // If Δt_1 is a usable time increment
            Δt = Δt_1// Change Δt to the usable value
            (rn_0, links_0) = CalculatePosition(rn, links, vn, Δt)// Find node and
             segment positions after Δt
            i = maxiter// Exit the loop
        else
            Δt = Δt/2// Decrease Δt if the error or maximum distance
             moved is too large
        end
    end
end
rn = rn_0// Update structure
links = links_0
t = t + Δt// Advance time by Δt
```

Algorithm 2: Pseudocode of an implementation of an alternative time integration algorithm.

This algorithm has multiple advantages over the previous, more basic algorithm. It does not unnecessarily decrease Δt, it can enlarge the time increment more efficiently and it is more flexible in its operation. A typical value for the maximum number of iterations would be 10, however the algorithm rarely progresses through more than 5 before finding a suitable value for Δt, with the error or maximum distance within 5 percent of the set limit. While there is still room for further improvement, this overcomes the major shortcomings of the original algorithm.

Comparing the original algorithm with the new one reveals some interesting points. Using the same input variables a test simulation was run to show performance during elastic deformation. The original algorithm completed the test in 8705 seconds after 2393 time increments. The new algorithm completed the simulation in 9695 seconds in 1531 time increments. At first, this seems like the original algorithm has better performance, however there are two points to note. The first is that the new algorithm allows for the use of a higher r_{tol} without the simulation breaking. Using a higher r_{tol} allows the new algorithm to complete the test in 2021 seconds after 261 time increments, which is approximately 4 times faster using nearly an order of magnitude fewer number of time increments. The second point is that reducing the number of time increments is more important when simulating plastic deformation. During elastic deformation, the process that requires the most amount of real time to perform is the calculation of the forces between dislocation segments. This must be done with every iteration of the time integrator, which is why the original algorithm is faster despite using more time increments, since it requires fewer iterations per time increment. During plastic deformation, the dislocation network is much more complicated in terms of its geometry, movement and interactions between segments. As a result, topographical changes become the rate limiting step. Since these happen once per time increment, using fewer time increments greatly reduces the run time for the simulation. For example, in the same time it took the new algorithm to simulate approximately 50nm of plastic displacement, around 67 hours, the original algorithm was only capable of simulating slightly more the 1.5nm of plastic deformation. This means that to simulate a similar amount of plastic deformation would take the original algorithm in excess of 2000 hours, which is impractical. To simulate the amount of plastic deformation typically recorded by micromechanical tests would take multiple years.

Directly improving the algorithm has an immediately noticeable impact on performance, however in the end it is merely a better tool for the job at hand, and it must be used correctly. This requires properly calibrating the input control parameters: r_{max}, r_{tol} and Δt_0.

Let us first consider r_{max}, the maximum distance any node can move. This parameter serves to ensure no dislocation event we would want to simulate is skipped. A key example is the collision of dislocation segments. A good way to decide on a suitable value for r_{max}, is to consider the situation of the "near miss".

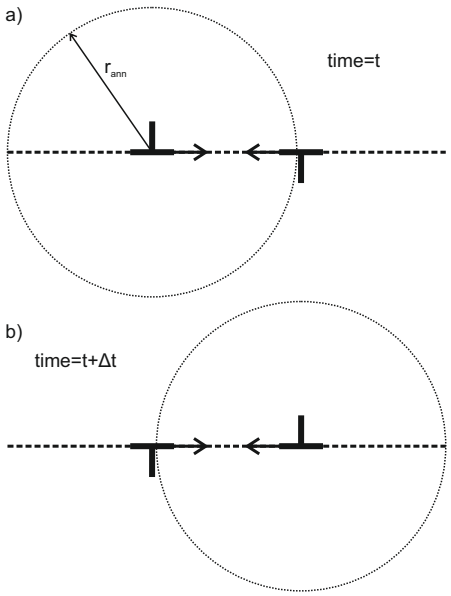


Figure 2.12: Two dislocations are on a collision course in (a). The following time increment is too large, resulting in them passing through each other without the collision being detected as shown in (b).

Figure 2.12 shows how two dislocations that should collide can "pass through" each other if Δt is too large. If we assume that, given their proximity to each other, the forces they exert on each other are dominant compared to other forces they may feel, then their velocities can be presumed to remain such that they are still moving towards each other after the time increment. Thus, they may move back and forth indefinitely without actually colliding. This can be prevented by ensuring that $r_{max} < r_{ann}$. Making r_{max} too small will significantly increase computation time by unnecessarily increasing the number of time increments, so it is best to make r_{max} as large as possible. A good value might be $0.99r_{ann}$, rather than directly setting it to r_{ann}, in order to account for numerical instability. In more complicated simulations, r_{max} need reducing further.

Once we have set r_{max}, r_{tol} can be calibrated. To ensure good agreement with r_{max}, we can consider a shear loop expanding in infinite space. We can take the error value of a time increment required for the fastest moving node to progress by r_{max} as r_{tol}. This way, any increase in the error for a more complicated simulation can be assumed to be due to the presence of other dislocations. When combining DDD with FEM, other factors need to be considered, and this will be discussed in Chapter 4.

Finally, the parameter Δt_0 is fairly arbitrary for DDD simulations with an otherwise well calibrated time integrator. It should not be too small, otherwise it will cause multiple unnecessary time increments to be used where r_{max} and r_{tol} would allow only one. If it is made too large, then it may take a long time to find a suitable time increment during the first iteration of the simulation, although it would have no other impact. However, once you combine DDD simulations with FEM, Δt_0 becomes much more important, as will be expanded upon further in Chapter 4.

2.4 Summary

Improving the performance and reliability of the topological operations and the time integrator as they were implemented in *DDLab* has had extremely beneficial effects on the capabilities of *EasyDD*. The run times of test simulations were decreased by an order of magnitude from the streamlining of each of these functions alone, and the code is able to simulate greater numbers of segments and higher dislocation densities than before. Additionally, combining these improvements with correctly fine tuned variables has prevented the occurrence of several phenomena which would prevent simulations from running past a certain point. Now that we have reached this point, we can focus on further additions that will improve the functionality of the code.

Chapter 3

Understanding and Improving Mobility Laws

3.1 Introduction

The mobility law is a key component of any DDD model. When combined with the force calculations, it determines how well a simulation reflects the way dislocations would actually behave by incorporating the effects of crystallography. Thus it is important to properly understand how they work and fine tune them to be as effective as possible.

3.2 Mobility Law Basics

Once all of the forces acting on the dislocation segments have been calculated, a shape function is used to assign appropriate portions of these forces to the nodes. These nodal forces must be used to calculate the velocities of the nodes. To do this we must use a mobility law. The basic methodology for creating a mobility law is discussed by Bulatov and Cai [52]. Again, it should be noted that throughout this section that the index notation used is not Einstein notation but the notation used to indicate different nodes associated with vectors and tensors. The principle is as follows:

$$\boldsymbol{v}_i = \boldsymbol{M}\left(\{\boldsymbol{f}_j\}\right) \tag{3.1}$$

where the velocity on node i, $\boldsymbol{v}_i$, is equal to some mobility function, $\boldsymbol{M}$, applied to the sum of the forces acting on all nodes connected to node i, $\boldsymbol{f}_j$. The mobility function is derived by looking at the drag acting on all the dislocation segments attached to node i. Due to the quasi-static framework used in DDD, the total force on any given node is zero, that is to say the drag force is equivalent to the

negative of the applied force. Bulatov and Cai thus derive the following equation:

$$\sum_j \boldsymbol{B}_{ij} \cdot \boldsymbol{v}_j = \boldsymbol{f}_i \tag{3.2}$$

where $\boldsymbol{B}_{ij}$ is the drag tensor of the segment connecting nodes i and j, and $\boldsymbol{f_i}$ is the total force applied to node i. For a given segement, $\boldsymbol{B}_{ij}$ is a function of the segment length and orientation as well as material properties and crystal structure. The total refers to the sum of all drag contributions from each node attached to node i. This then gives us a series of simultaneous equations to solve for each dislocation loop. Since velocity calculations in DDD are done piece-wise by node, this is not ideal. In order to generate equations that could be solved for each node, Bulatov and Cai proceeded to decouple the equations by making the assumption that the velocities of nearby nodes are roughly equivalent:

$$\boldsymbol{v}_i \approx \boldsymbol{v}_j \tag{3.3}$$

This is not a good assumption to make and several basic examples can demonstrate how it is physically invalid, since velocities may have very similar magnitudes while maintaining orthogonal directions, even if the nodes are neighbouring. Alternatively, a mobile node may be neighbouring a node that is fixed because it is part of some structure such as a junction. Since segment length is deliberately limited to in turn limit the difference in the stress state at either end and the force on the segment is distributed to the nodes by the shape function, one can make the more robust assumption that the drag force on two adjacent nodes as a result of the segment connecting them is roughly equivalent:

$$\boldsymbol{B}_{ij} \cdot \boldsymbol{v}_i \approx \boldsymbol{B}_{ij} \cdot \boldsymbol{v}_j \tag{3.4}$$

However, mathematically speaking, this assumption has the same effect as the previous assumption due to the symmetry of the drag tensor of an individual segment and the fact that it is the same no matter which node in the segment it is applied to. Hence, regardless of which assumption is made, the derived equation is the same:

$$\boldsymbol{v}_i = \left(\sum_j \boldsymbol{B}_{ij} \right)^{-1} \cdot \boldsymbol{f}_i \tag{3.5}$$

Thus, in order to find a good approximation for the velocity of a node, we must simply construct a drag tensor for each of the segments it is connected to.

A drag tensor requires three coefficients for each type of drag that can affect the segment in question. They are for movement in the slip plane, B_{glide}, movement

out of the slip plane, B_{climb}, and movement in the line direction, B_{line}. For each of these coefficients, a normalised vector describing the direction it is acting in must be defined, so that:

$$\boldsymbol{B} = |\boldsymbol{l}|(B_{line}\widehat{\boldsymbol{l}} \otimes \widehat{\boldsymbol{l}} + B_{glide}\widehat{\boldsymbol{m}} \otimes \widehat{\boldsymbol{m}} + B_{climb}\widehat{\boldsymbol{n}} \otimes \widehat{\boldsymbol{n}}) \tag{3.6}$$

where $\boldsymbol{l}$, $\boldsymbol{m}$ and $\boldsymbol{n}$ are the line, glide and climb directions respectively. This is simple for the line direction since it is known. For non-screw dislocations, the glide direction can be found using the cross product of the line direction with the slip plane normal. The climb direction should be the slip plane normal itself. Defining a slip plane normal can be non-trivial however. Previous mobility laws have used the normalised cross product of the line direction with the Burgers vector. This gives a **habit** plane for the segment, which in the case of a shear loop is the slip plane. Unfortunately, this is not universally true, such as when a dislocation is climbing or a junction forms with a Burgers vector that is a combination of Burgers vectors that exist on different slip planes. Climb is a process related to the crystallography of the material being simulated, so a defined climb direction **must** be along a crystallographic slip plane normal. To find an appropriate slip plane normal, the dot products between the habit plane normal and all of the viable slip plane normals can be used. The dot product with a magnitude closest to 1 indicates which slip plane the segment is closest to occupying. This dot product also acts as a measure of how close the segment is to being in the slip plane. For a segment that does not inhabit a viable slip plane, the line direction is not perpendicular to the slip plane normal and so cannot be used as a vector for the construction of the drag tensor. To account for this, the line direction can be projected into the slip plane, shown in Figure 3.1, although the drag coefficient must be interpolated between B_{line} and B_{climb} as necessary to give a suitable drag coefficient in this direction.

The construction of a drag tensor requires three perpendicular vectors. These can be difficult to define for screw segments. The original mobility laws as outlined by Bulatov and Cai ignored this problem by allowing screw dislocation nodes to move freely in any direction perpendicular to the line direction and much faster movement in the line direction, making the drag tensor for a screw segment with line direction $\boldsymbol{l}$:

$$\boldsymbol{B} = |\boldsymbol{l}|(B_{screw}\boldsymbol{I} - (B_{screw} - B_{line})\,\widehat{\boldsymbol{l}} \otimes \widehat{\boldsymbol{l}}) \tag{3.7}$$

where $\boldsymbol{I}$ is the identity matrix and B_{screw} is the glide drag coefficient for screw segments. This is a simple first approximation. In simple simulations, screw segments are usually attached to mixed and edge segments that prevent the pure

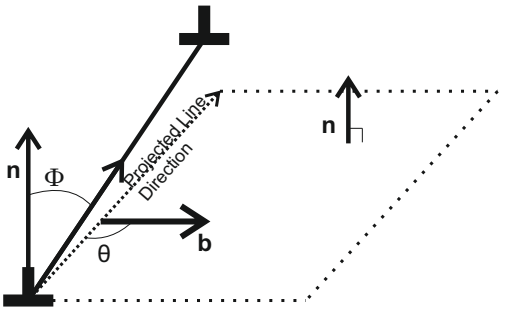


Figure 3.1: An edge dislocation its line direction not in any crystallographic slip plane, so the closest slip plane must be identified. Φ is the angle between the dislocation line direction and the slip plane normal $\boldsymbol{n}$, while θ is the angle between the dislocation line direction and the Burgers vector. For pure edge dislocations, $\theta = 90°$. The line direction projected into the slip plane can be used as a vector to construct the drag tensor.

screw part of the dislocation from moving in a direction that would not be in a crystallographic slip plane by contributing elements to the nodal drag tensor that restrict motion appropriately. However, as simulations become more complex, this non-physical description of screw mobility can lead to screw segments moving on planes that they would not be able to in reality and thus forming structures that would not be observed experimentally and act as obstacles to the progression of the simulation. In order to compensate for this, some means of defining a slip plane that the screw is moving on must be devised. This can be accomplished by looking at the force on the dislocation. Since the line and Burgers vector are the same, a given screw segment has a number of crystallographic slip planes it exists in simultaneously because the screw segment lies along the line where these planes intersect. For example, in BCC structures, a $\frac{a}{2}[111]$ Burgers vector screw segment exists in the $(10\bar{1})$,$(1\bar{1}0)$ and $(01\bar{1})$ planes. Which of these planes it will move in can be estimated by using the resolved force vector that is acting on the segment. The slip plane in which the largest component of the applied force exists can be designated as the slip plane for the segment. Alternatively, a similar treatment can be done using the resolved shear stress. This can then be used to find a glide direction and the screw segment can be treated as any other segment.

The drag coefficients used to construct the drag tensor are not necessarily input parameters. B_{climb} can be since it should always act parallel to a crystallographic slip plane normal and describes the physical difficulty for a segment to move out of the slip plane, which for most current simulations should be an approximation of infinity, since climb is much slower than glide and the mechanisms that govern it more complicated. Since mobility laws able to incorporate climb behaviour have not been developed to date, making B_{climb} as large as possible is necessary in order to inhibit any motion out of the slip plane due to climb. B_{glide} describes the ease

of glide for a dislocation segment and varies depending on the segment orientation. The input parameters B_{edge} and B_{screw} describe the drag on pure edge and pure screw segments in the glide direction respectively, so for mixed dislocations B_{glide} should be an appropriate interpolation between these two constants. Such interpolations are dependent on crystallography to determine the shape of dislocation loops. Additionally, as a segment becomes orientated more out of a viable slip plane, it becomes more difficult to move. This means there needs to be an interpolation for both B_{line} and B_{glide} as the segment is oriented more out of plane so that they become closer to B_{climb}. This makes it more difficult for out of plane segments to move or increase their length, both of which are actions determined to be unfavourable by the crystallography. The actual values of the drag coefficients are somewhat arbitrary, and it is more the ratios between them that matters. The key areas of note are that, in the model, one of the drag coefficients, usually B_{edge}, is divided by the shear modulus, μ, to give the time units and that the simulated loading rate must be appropriately calibrated to maintain quasi-static conditions. Lower drag coefficients require a lower loading rate. This means that the purpose of the actual value of the drag coefficient is to convert simulation time units into seconds, so appropriate values can be chosen for numeric stability and then the elapsed time and dislocation velocities can be converted after the simulation has completed.

Another crucial point to notice when dealing with mobility laws is that care should be taken when inverting the total drag tensor. While a single drag tensor for each segment is constructed to be invertible, the total drag tensor for a node is not necessarily so, since the sum of multiple matrices with all non zero eigenvalues can create a matrix with one or more zero eigenvalue, eg:

$$\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 4 & 2 \\ 4 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix} \tag{3.8}$$

In truth, because of machine precision, computers are incapable of inverting any matrix where there is a large order of magnitude difference between two eigenvalues. If this is the case, a treatment must be applied to gain an approximation of the correct result. The original *DDLab* mobility laws diagonalised the uninvertible matrices and used principle component analysis to remove elements too small to be within machine precision. Unfortunately, due to a typo in the code, this was done on the wrong side of the inversion, meaning that components that should have been very large became zero. This resulted in the velocities calculated being of the wrong magnitude, although they largely acted in the correct direction. The upshot of this was that the variation in velocities between time increments

increased massively when certain criteria were met, causing the time integrator to generate smaller time increments and thus make the code take longer to run. This issue was somewhat mitigated by improving the time integrator, reducing the chances of the simulation coming to a complete stop, but it still remained a problem. Fixing the principle component analysis can fix this problem, although an alternative method is to perturb the drag tensor slightly by adding an identity matrix multiplied by an appropriately small factor, c, and inverting the resulting matrix to gain an approximation of the velocity:

$$\boldsymbol{v}_i \approx \left(\left(\sum_j \boldsymbol{B}_{ij}\right) + c\boldsymbol{I}\right)^{-1} \cdot \boldsymbol{f}_i \tag{3.9}$$

Mathematically, the perturbation may be one in the opposite direction by subtracting the same identity matrix and additional stability can be gained by doing both and then using the mean of the results:

$$\boldsymbol{v}_i \approx \frac{1}{2}\left(\left(\left(\sum_j \boldsymbol{B}_{ij}\right) + c\boldsymbol{I}\right)^{-1} \cdot \boldsymbol{f}_i + \left(\left(\sum_j \boldsymbol{B}_{ij}\right) - c\boldsymbol{I}\right)^{-1} \cdot \boldsymbol{f}_i\right) \tag{3.10}$$

The perturbation is simpler, faster and requires fewer lines of code than the correctly implemented principal component analysis, however principal component analysis has the potential to be more accurate. Care should also be taken when perturbing by subtraction since this will likely make drag in one direction negative. Negative drag is not an observed physical phenomenon and would cause a node to move in the opposite direction to which force is being applied if all drag coefficients were negative. The negative perturbation should only be done if used to create an average with the positive perturbation.

3.3 A Simple Example

In order to visualise how a mobility law works, let us consider a fictional example of a slip system with Burgers vector direction, edge line direction and climb direction in each of the coordinate axes respectively. The drag tensor for an edge dislocation in such a slip system would be as follows:

$$|\boldsymbol{l}| \begin{pmatrix} B_{edge} & 0 & 0 \\ 0 & B_{line} & 0 \\ 0 & 0 & B_{climb} \end{pmatrix} \tag{3.11}$$

In this tensor, B_{edge} is the drag in the direction perpendicular to the edge dislocation line direction but in the slip plane, B_{line} is the drag in the dislocation line direction and B_{climb} is the drag in the direction out of the slip plane. Climb requires multiple factors to model accurately, such as temperature and vacancy diffusion. As a result of this, it can be convenient to model dislocation motion through climb in a separate function and have it "turned off" inside the mobility law. To have zero climb however requires B_{climb} to be infinite. This is not a practical input parameter so it must instead be a very large number compared to the next highest drag coefficient. In theory, there is no drag in the line direction since a node is not a physical feature and placing it anywhere in the structure should be equivalent. It cannot be made to be zero since this will make it mathematically impossible to invert the drag tensor to solve the equation for velocity. It is most simple to assign a very small value for B_{line} so as to approximate zero, however this has two major problems. The first is that this tends to make the drag tensor ill conditioned, making it unsuitable for computational functions to invert. An approximation to the required inversion can be found (as discussed previously), but if this approximation needs to be used more often than not then it will influence the accuracy of the simulation. The second issue involves how the time integrator depends on the mobility law. The size of a time increment as decided by the time integrator depends on the fastest moving node and the node with the greatest change in velocity between time increments. This is because the greatest distance any node moves and the greatest change in velocity any node has must both be restricted to be below set critical values to maintain the quasi-static condition. Typically, only one of these will be the limiting factor for a given time increment. Time increment sizes are controlled so that the maximum distance moved and the maximum change in velocity are limited as necessary. We can use a few simple examples to determine what the drag in the line direction should be. For these is examples, the applied stress, $\boldsymbol{\sigma}$, is as follows:

$$\boldsymbol{\sigma} = \begin{pmatrix} 0 & 0 & \sigma \\ 0 & 0 & 0 \\ \sigma & 0 & 0 \end{pmatrix} \tag{3.12}$$

where σ is some factor such that the Peach-Koehler force on [100] Burgers vector dislocations is several orders of magnitude larger than the self forces and remote forces. Consider the following example of two Frank-Read sources:

In Figure 3.2, the fastest moving node will be the one moving along its line direction, where there is significantly less drag, even if the magnitude of the force is less. However the dislocations in the Frank-Read source it is a part of are not actually moving. The node will shorten one segment and elongate the other until

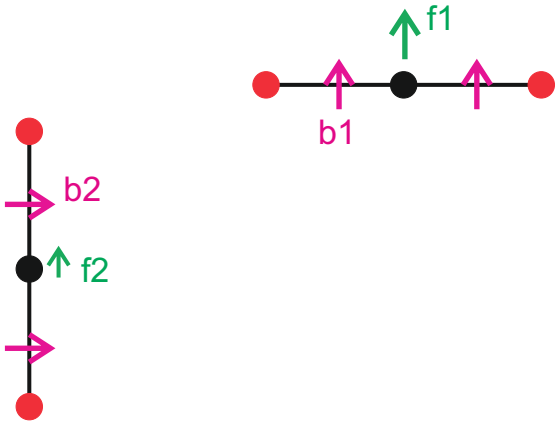


Figure 3.2: Two perpendicular Frank-Read sources in the same simulation. Due to having perpendicular Burgers vectors ($\boldsymbol{b}_1$ and $\boldsymbol{b}_2$) and line directions, the same applied stress results in different Peach-Koehler forces and the self forces are different. The total force on node 2 may be smaller than on node 1 but the velocity can be much faster since the node is moving in the line direction

remeshing creates a new node. This new node will move as fast as the old node which will eventually be remeshed out. The time integrator will limit the time increment based on the velocities of these nodes despite the Frank-Read source not actually being active. This means that the number of time increments required to simulate the motion are unnecessarily increased. The best way to reduce this effect is to assign B_{line} a value with a similar order of magnitude to B_{edge}. However if the value of B_{line} is not much less than B_{edge} then it may affect the shape of the dislocation structure, particularly the geometry of any arc formed by bowing and similar behaviour, which would not be physical. So what value should be assigned to B_{line} to maintain both the efficiency of the model and the shape of the dislocation structure?

In order produce a suitable line drag coefficient, it is necessary to consider the total drag tensor applied to a node rather than the drag tensors of individual segments. Let us further simplify our fictional example by saying that we can always reliably identify the slip plane of screw dislocations. Consider a square loop, as shown in Figure 3.3:

Nodes at the midpoints of the sides of the square are either connected to two segments of edge character or two segments of screw character. This means that the combined drag tensor for these midpoint nodes (i.e. the total of all $\boldsymbol{B}_{ij}$) is either twice the previously shown tensor for edge sides or as follows for screw sides:

$$2|\boldsymbol{l}| \begin{pmatrix} B_{line} & 0 & 0 \\ 0 & B_{screw} & 0 \\ 0 & 0 & B_{climb} \end{pmatrix} \tag{3.13}$$

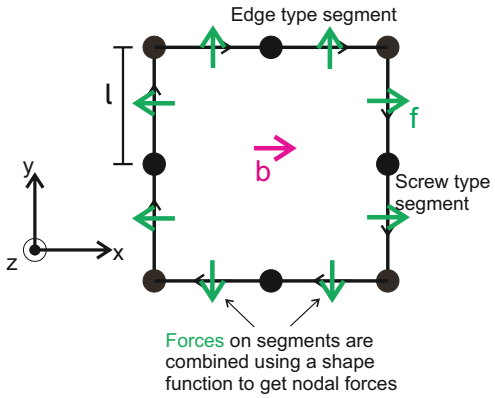


Figure 3.3: A square loop of Burgers vector $\boldsymbol{b}$ with sides of pure edge and pure screw character.

This does not tell us much since glide and line motion are independent, so it is fine if line drag is reduced by a factor of how much faster we want nodes to be able to move in the line direction. The real issue occurs with the corner nodes, which all have the following total drag tensor:

$$|\boldsymbol{l}| \begin{pmatrix} B_{edge} + B_{line} & 0 & 0 \\ 0 & B_{screw} + B_{line} & 0 \\ 0 & 0 & 2B_{climb} \end{pmatrix} \tag{3.14}$$

The ratio between B_{edge} and B_{screw} informs us of the ideal shape that dislocation loops prefer to take. If these two drag coefficients are the same then loops will expand to be circular when a uniform stress is applied. As the ratio changes, the side of the loop with greater drag, usually the screw side, becomes longer as the faster moving side with less drag moves away from the centre of the loop with a greater velocity. From this we can see that the degree to which a direction of motion is preferred by the system needs to be controlled by the input drag coefficients so that the loops produced match those observed in experiments. In the given example, when B_{line} is very small its exact value has little impact on the resultant velocities of the nodes, since the other drag coefficients dominate, making it reasonable to simply assign a suitably small number as the value of B_{line}. However, as previously stated, assigning very small line drag coefficients can be problematic when trying get the code to run smoothly. The magnitude of B_{line} cannot simply be increased though, because this means that as angles between adjoining segments become sharper, the resultant velocity away from the centre of the loop becomes more influenced by the assigned line drag. Practically speaking, this is caused by B_{line} altering the rotational properties of the drag tensor, so that when the inverted drag matrix is applied to the force vector, the direction of the resultant velocity vector is not correct. If we wish the direction of motion

inside the slip plane to remain consistent, then the ratio between the resultant drag coefficients applied to the nodes must be maintained. Looking at the given example, we can see that this can be achieved by making the line drag dependent on the orientation of the dislocation. If we use some scaling factor, λ, to control how much faster dislocations move in the line direction than glide direction, we can assign the line drag for edge dislocations as λB_{screw} and for screw dislocations as λB_{edge}, giving the following drag tensor:

$$|\boldsymbol{l}| \begin{pmatrix} (1+\lambda)B_{edge} & 0 & 0 \\ 0 & (1+\lambda)B_{screw} & 0 \\ 0 & 0 & 2B_{climb} \end{pmatrix} \tag{3.15}$$

Given that B_{climb} is always much larger than either of the glide drag coefficients, this is change maintains the preferred ratios, thus causing the degree to which the tensor rotates the direction of a force when calculating a velocity to be preserved. Any fine tuning can be done by altering the value of λ. If λ is very small, it is effectively the same as simply having a very small B_{line}. If it is too large, it becomes too difficult for nodes to move in the segment line direction. Suitable values for λ typically lie between 0.1 and 1. Now that we know the necessary line drag for edge and screw segments, we need to be able to extend this to all segments. Consider another square loop, in the same slip system as the previous one but rotated through 45° in the (001) plane:

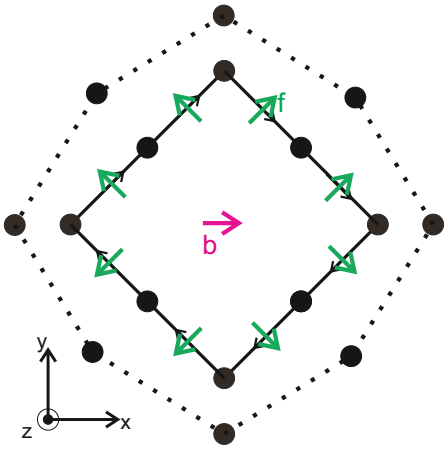


Figure 3.4: A square loop of Burgers vector $\boldsymbol{b}$ with sides of mixed character, oriented 45° away from pure edge type. The segments have either $\langle 110 \rangle$ or $\langle 1\bar{1}0 \rangle$ type line directions. The node loop's position after one time increment is shown in dashed lines.

Both types of mixed dislocation presented here are oriented exactly halfway between screw and edge type, meaning they would have the same drag coefficient in their line directions, B_{glide}. The drag tensors can be found by constructing drag tensors rotated out of the main coordinate axes, $\boldsymbol{B}_{rot}$, using their glide, line

and climb directions as coordinate axes and then rotating the tensors into the coordinate axes used for the previous example. For those midpoint nodes with two segments running in the $\langle 110 \rangle$ direction, this rotation is 45° and the resulting drag tensor is:

$$\begin{aligned}
\boldsymbol{R} &= \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
\boldsymbol{B}_{rot} &= |\boldsymbol{l}| \begin{pmatrix} B_{glide} & 0 & 0 \\ 0 & B_{line} & 0 \\ 0 & 0 & B_{climb} \end{pmatrix} \\
2\boldsymbol{R}\boldsymbol{B}_{rot}\boldsymbol{R}' &= |\boldsymbol{l}| \begin{pmatrix} B_{glide} + B_{line} & B_{line} - B_{glide} & 0 \\ B_{line} - B_{glide} & B_{glide} + B_{line} & 0 \\ 0 & 0 & 2B_{climb} \end{pmatrix}
\end{aligned} \tag{3.16}$$

Note that the rotated tensor is doubled since each node has two segments. Similarly, by rotating in the opposite direction,the midpoint nodes with two segments in the $\langle 1\bar{1}0 \rangle$ direction have the following drag tensors:

$$|\boldsymbol{l}| \begin{pmatrix} B_{glide} + B_{line} & B_{glide} - B_{line} & 0 \\ B_{glide} - B_{line} & B_{glide} + B_{line} & 0 \\ 0 & 0 & 2B_{climb} \end{pmatrix} \tag{3.17}$$

Finally, nodes at corners all have the same drag tensor, obtained by combining one $\langle 110 \rangle$ segment drag tensor and one $\langle 1\bar{1}0 \rangle$ segment drag tensor:

$$|\boldsymbol{l}| \begin{pmatrix} B_{glide} + B_{line} & 0 & 0 \\ 0 & B_{glide} + B_{line} & 0 \\ 0 & 0 & 2B_{climb} \end{pmatrix} \tag{3.18}$$

These drag tensors imply that the loop would expand symmetrically when a uniform stress is applied, since applying the drag tensors to forces with directions as shown in Figure 3.4 will yield velocities with identical magnitudes all pointing directly away from the center of the loop. However this will only last for one time increment. To understand why, we must compare the velocity of a midpoint node with that of a corner node. If the forces on the segments are all of the same magnitude and in $\langle 110 \rangle$ directions, then we can consider two nodes: a midpoint and a corner node. Given the symmetry of the example, these two node will inform us of the behaviour of all of the other nodes. For any midpoint node, the forces on both segments it is attached to are identical, Giving it a resultant force in the

same direction. This is not the case for a corner node:

$$\begin{aligned} \boldsymbol{f}_1 &= (f, f, 0) \\ \boldsymbol{f}_2 &= (f, -f, 0) \\ \boldsymbol{f}_{node} &= \frac{1}{2}(\boldsymbol{f}_1 + \boldsymbol{f}_2) = (f, 0, 0) \end{aligned} \tag{3.19}$$

where $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are the forces on segments 1 and 2 attached to the node respectively, $\boldsymbol{f}_{node}$ is the total force on the node and f is some scaling factor. The half factor in the equation is due to the shape function, since in this basic example half of the force on each segment would be felt by each node it is attached to. One of the neighbouring midpoint nodes of this corner node would have a total force of $(f, f, 0)$. Immediately we can see that the force the corner node feels is less than the midpoint nodes, but we can't know what effect this will have until we apply the inverted drag tensor to find the velocity. The drag tensor for the midpoint node is shown in Equation 3.17 and the drag tensor for the corner node is in Equation 3.18. We can now use Equation 3.5 to calculate the velocities. The process of doing this will not be shown here since it is trivial and left to the reader, though the results are as follows:

$$\begin{aligned} \boldsymbol{v}_{midpoint} &= \frac{|\boldsymbol{l}| \cdot f}{2B_{glide}}(1, 1, 0) \\ \boldsymbol{v}_{corner} &= \frac{|\boldsymbol{l}| \cdot f}{B_{glide} + B_{line}}(1, 0, 0) \end{aligned} \tag{3.20}$$

To maintain the square shape of the loop, the magnitude of the velocity of the corner node, $\boldsymbol{v}_{corner}$, would need to be larger than that of the midpoint nodes, $\boldsymbol{v}_{midpoint}$, by a factor of $\sqrt{2}$, which can only be the case when $B_{line} = 0$. We have already seen why this cannot be the case. Thus it can be seen that after one time increment the square would immediately become an octagon, with some sides being of greater edge character and others closer to screw type. Previously, four of the segments in the loop were oriented clockwise from pure edge by 45° and the other four were oriented similarly anticlockwise, both sets being exactly halfway between edge and screw type. After the expansion, of each set of four, two will now be oriented at slightly more than 45° from edge type and two will be slightly less. This means that when the drag tensors are combined at the nodes that were previously the corners of the squares, two will be a combination of the orientations closer to edge type and two will be combinations closer to screw type. Thus the symmetry in the drag coefficients will be lost and continued expansion

after a sufficiently large amount of time should result in the same shape of loop for both square loop examples.

From the treatment of the second square loop example, it can be seen that for all initial orientations of dislocation segments we should have the same B_{line} of λB_{glide}. If we wish to maintain consistency between the two examples and all orientations in between, the simplest solution is to interpolate the line drag coefficients in the same way the glide drag coefficients are interpolated, which is dependent on the crystallography. This means that the drag in the line direction for a given segment is identical to the glide drag for a perpendicular segment in the same slip system, scaled by λ. This allows for control of how fast nodes can re-position themselves within the network whilst maintaining the correct geometry of any dislocation arcs formed, even at fine resolutions. As a further note, all of the examples given deal with segments with equivalent lengths for the sake of simplicity. Drag tensors for segments are scaled by the length of the segments, since larger segments have a greater contribution to the total drag tensor of a node. This is balanced by the fact that the force applied to longer segments is also larger.

3.4 Modifying for BCC Systems

Tailoring a mobility law to a body centred cubic crystal, as with any crystallographic arrangement, requires coding in the viable slip systems. The first step is to check the Burgers vector; if $\boldsymbol{b} \neq \frac{a}{2}\langle 111 \rangle$ then it is not part of a viable BCC slip system. Such dislocations can form when other dislocations collide, usually forming a sessile junction. These dislocations must have their mobility altered accordingly. The simplest method is to assign a large drag coefficient, such as B_{climb}, to the direction perpendicular to the line direction and a suitable line drag coefficient to the line direction based on its orientation. More complicated treatments may be applied if junctions are not sessile or if junctions with different types of Burgers vector have different zipping and unzipping behaviour, requiring different line drag coefficients so that they can increase and decrease their lengths appropriately.

Once a dislocation has been identified as having a suitable Burgers vector, further steps must be taken to construct a suitable drag tensor. First, the angle between the Burgers vector and the line direction must be found. If this angle is close to 0°, then the dislocation is screw type and requires special treatment. For now, let us consider mixed and edge dislocations. The normal of the habit plane of a given segment can be found using $\widehat{\boldsymbol{l}} \times \widehat{\boldsymbol{b}}$. This can be compared to a list of slip plane normals by again using dot products. The slip plane that has a dot product

with a magnitude closest to 1 can assumed to be the slip plane of the dislocation. If the dot product is exactly 1 or -1 one then the dislocation can be said to be lying in the slip plane, where it will have the highest mobility. As the line direction of the dislocation moves out of the slip plane, the drag must be increased. The drag in the direction of the slip plane normal must always be B_{climb}, since this is the climb direction as dictated by the crystallographic arrangement. Interpolation is used to find the relevant drag coefficients from the basic coefficients as they are inputted. First the drag and line coefficients are found for a dislocation in plane by using an interpolation based on how close to being screw type the dislocation is. Arsenlis et al. [55] developed the following interpolation, which causes dislocation loops to take the geometry observed in BCC crystals:

$$\begin{aligned} \cos\theta &= \widehat{\boldsymbol{l}} \cdot \widehat{\boldsymbol{b}} \\ B_{glide} &= \left(\left(\frac{\cos^2\theta}{B_{screw}} \right)^2 + \left(\frac{\sin^2\theta}{B_{edge}} \right)^2 \right)^{-\frac{1}{2}} \end{aligned} \tag{3.21}$$

where θ is the angle between the line direction and the Burgers vector. This interpolation can be used to derive glide coefficients, as well as line coefficients if one is using the previously mentioned approach:

$$B_{line} = \lambda \left(\left(\frac{\cos^2\theta}{B_{edge}} \right)^2 + \left(\frac{\sin^2\theta}{B_{screw}} \right)^2 \right)^{-\frac{1}{2}} \tag{3.22}$$

Calculating the line drag coefficient in this manner is redundant if a simple significantly small line drag coefficient is used instead. However, both coefficients must further be interpolated based on how well the dislocation segment lies in the slip plane. As a segment moves out of plane,it should become more difficult for it to move and so the derived glide and line coefficients need to be further modified to make them closer to the climb coefficient. The following simple interpolation can be used:

$$\begin{aligned} \cos\Phi &= \widehat{\boldsymbol{l}} \cdot \widehat{\boldsymbol{n}} \\ B_g &= \sqrt{B_{glide}^2 \cos^2\Phi + B_{climb}^2 \sin^2\Phi} \\ B_l &= \sqrt{B_{line}^2 \cos^2\Phi + B_{climb}^2 \sin^2\Phi} \end{aligned} \tag{3.23}$$

where $\boldsymbol{n}$ is the calculated slip plane normal and B_g and B_l are the final glide and line drag coefficients to be used. These interpolations are designed so that a pure

edge segment with a habit plane equivalent to a crystallographic slip plane has $B_g = B_{edge}$ and $B_l = \lambda B_{screw}$. Once the final coefficients have been calculated, the drag tensor can be constructed using $\boldsymbol{n}$ as the climb direction, $\boldsymbol{n} \times \boldsymbol{l}$ as the glide direction, $\boldsymbol{m}$, and $\boldsymbol{m} \times \boldsymbol{n}$ as the "effective" line direction, which is the projection of the line direction into the selected slip plane. This projection must be performed since the three vectors should be orthogonal for the drag tensor to be well behaved.

If the segment has been found to be screw type, then the construction of the drag tensor is a bit more complicated. The simplest part is the line drag, which always uses the same drag coefficient and is always in the same direction. Constructing the drag tensor perpendicular to the line direction is where the problems arise. Defining the slip plane is no longer trivial. The force on the dislocation can be used to find the closest slip plane, similarly to how we used the Burgers vector for edge and mixed dislocations. This tells us which plane the dislocation would prefer to slip on. Using the force on the dislocation allows us to model cross-slip by detecting when the segment would slip on a different plane. Simply constructing a drag tensor using the derived glide and climb directions for this slip plane would be problematic though. This can be demonstrated by considering how different drag components contribute to the total velocity of a node attached to two pure screw segments. The velocity perpendicular to the line direction can be split into component vectors in the directions of glide and climb. A single slip system can be represented by plotting these components in a polar plot as a function of the orientation of a unit force orthogonal to $\boldsymbol{l}$, as shown in Figure 3.5.

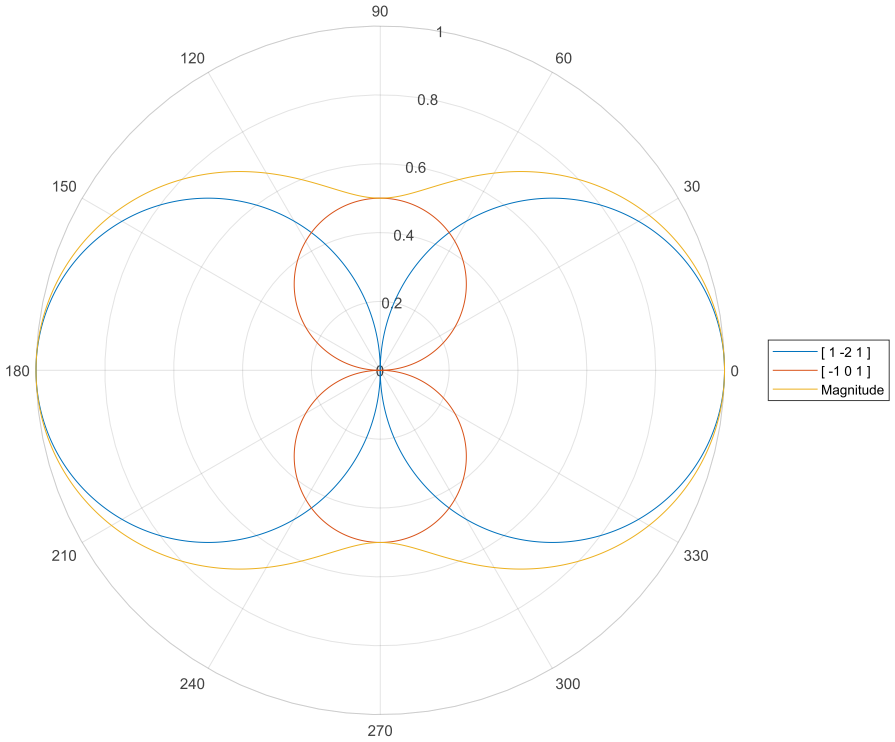


Figure 3.5: A 2D graphical representation of how glide and climb drag coefficients effect the velocity of a $\frac{a}{2}[111]$ Burgers vector dislocation. The curves show the component of the velocity in the glide, $[1\bar{2}1]$, and climb, $[\bar{1}01]$, directions as the orientation of the force relative to the dislocation changes, as a proportion of the velocity in the glide direction, as well as the total magnitude of the velocity vector. The [111] direction is out of the page, the $[1\bar{2}1]$ direction is at 0° and the $[\bar{1}01]$ direction is at 90°. The climb drag coefficient is only twice that of the screw glide coefficient for ease of depiction.

Figure 3.5 gives an indication of the rotational properties of the drag tensor as well as how the magnitude of the velocity is changed. It does this by depicting the total magnitude of the calculated velocity as components in multiple directions. Thus it can be seen that for the majority of applied force directions, the glide direction contributes overwhelmingly to the direction of the total velocity, keeping the dislocation moving in plane. This effect becomes more pronounced as B_{climb} is increased in proportion to B_{screw}. However, if we are going to interchange between slip systems as the force orientations dictate, the behaviour in other slip directions must be acknowledged as well. For the sake of simplicity, let us only consider the {110} type slip planes. This gives three possible glide directions. Plotting the components in all three directions for one slip system yields the plot shown in Figure 3.6.

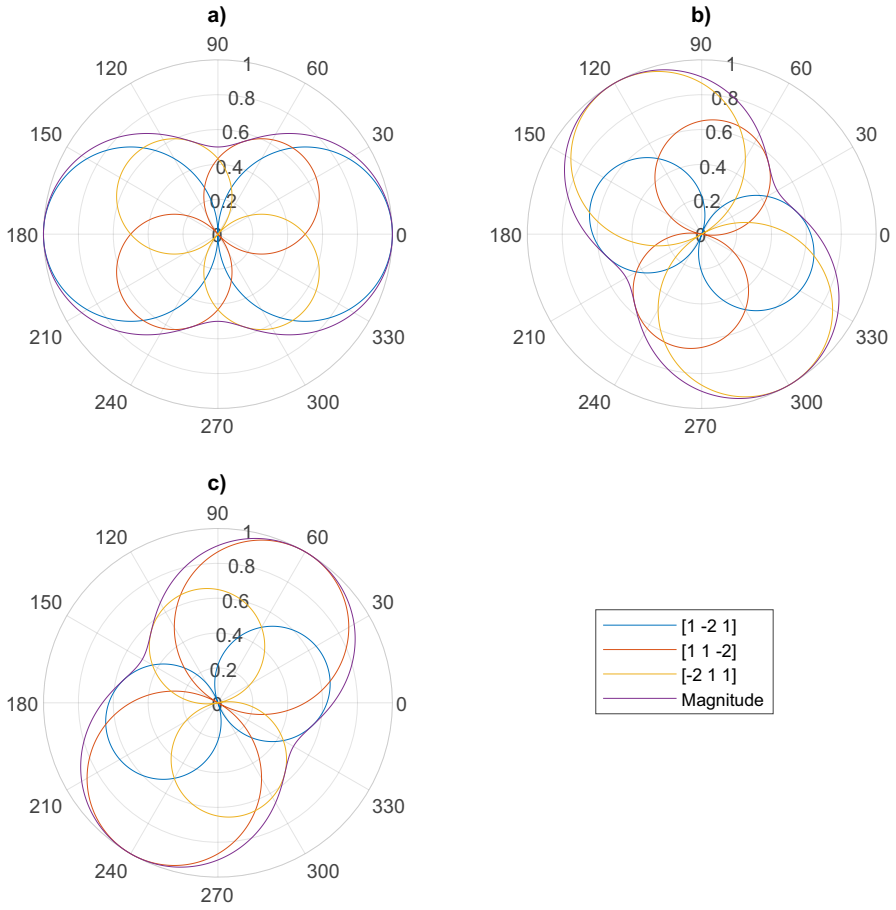


Figure 3.6: A 2D graphical representation of all $\frac{a}{2}[111]$ Burgers vectors in (101) type slip planes slip systems, with the velocity component in all possible screw glide directions shown. The slip planes are: a) $(\bar{1}01)$, b) $(\bar{1}10)$, c) $(0\bar{1}1)$.

Figure 3.6a shows the same behaviour as Figure 3.5, but shows how velocity components in the other possible slip directions change to better depict the effect the drag tensor has. If we allow for dislocations to switch between slip systems, such as those shown in 3.6b and 3.6c, as a result of cross-slip, then 3.6a does not represent the full range of behaviour, only the behaviour when the force is closer to lying in the $(\bar{1}01)$ plane than either of the others. There is a 60° angle between each plane and the next closest plane. This means that when the force moves 30° out of plane, the slip system regime changes and drag is applied differently. For the full circle around the line direction, there are six such regimes, two for each plane. We can construct a more comprehensive plot of glide behaviour by combining only the relevant directions of the three plots in Figure 3.6, as shown in Figure 3.7.

Figure 3.7 shows the same behaviour as Figure 3.6 between the angles of 150° and 210° and the angles of 330° and 30°, which are the regimes where the force is closest to lying in the $(\bar{1}01)$ plane. There is a change between regimes at 60° intervals as the force becomes closer to existing in other slip planes. If the

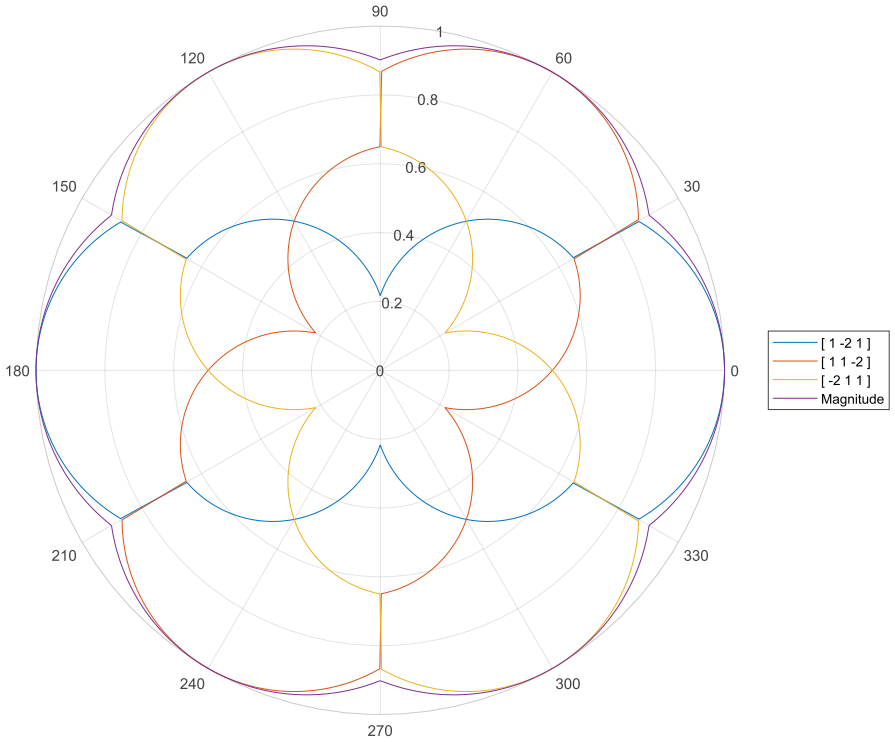


Figure 3.7: A possible model for screw glide behaviour. The component of the velocity in each of the glide directions as well as the magnitude of the velocity are shown. The slip system selected to create the drag tensor is whichever the force vector is closest to lying in. Step changes in the curves can be seen at 60° intervals. This represents a sudden change in direction of the dislocation's velocity, which is not physical.

construction of the drag tensor is performed as if it is simply an edge or mixed dislocation, then there is a sudden change in behaviour as the force moves from one slip system regime to another, as is shown by the sudden step changes in Figure 3.7, indicating a sudden shift in the direction of the velocity. This cannot be physical, since a minor change in the direction of the applied force should not result in a major change in dislocation motion. To account for this, the drag coefficients must be interpolated in a way that satisfies several conditions:

- Drag behaviour must be the same for neighbouring slip system regimes at the orientation where the regime changes (30° out of plane in this case)
- The gradient of the change of drag behaviour must also be close to the same at the orientation at which slip system regimes change for a smooth transition of behaviour
- The slip system regime change orientation must be the orientation at which glide drag is the greatest, since there is no real preferred direction of motion at this orientation and motion in general should be prohibited

These conditions require us to create an interpolation that is both dependent on the angle of the force between planes and the angle between the planes themselves. A basic interpolation would be:

$$\alpha_1 = \frac{1}{2}\left(\frac{\sin\theta_c}{\sin\frac{\phi}{2}}\right)^2$$

$$\alpha_2 = 1 - \alpha_1$$

$$B_{glide} = \left(\frac{\alpha_1}{B_{climb}^2} + \frac{\alpha_2}{B_{screw}^2}\right)^{-\frac{1}{2}}$$

$$B_{sclimb} = \left(\frac{\alpha_1}{B_{screw}^2} + \frac{\alpha_2}{B_{climb}^2}\right)^{-\frac{1}{2}} \tag{3.24}$$

(3.25)

where θ_c is the angle between the closest slip plane and the direction of the force vector, ϕ is the angle between viable slip planes, shown in Figure 3.8, B_{screw} is the glide coefficient of a screw segment moving perfectly in plane, B_{climb} is the standard climb coefficient and B_{sclimb} is a "screw climb" drag coefficient found by interpolation that is needed to keep screw segments moving in a crystallographic slip plane.

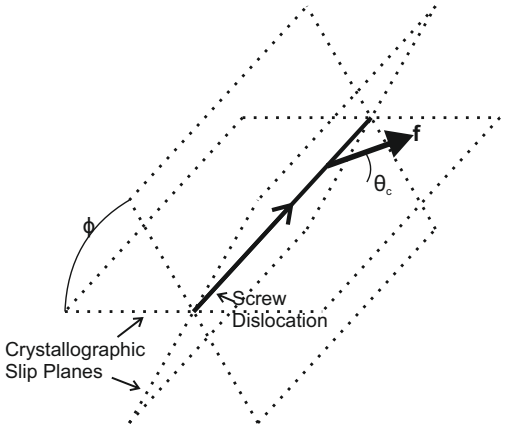


Figure 3.8: A screw dislocation that is feeling a force, $\boldsymbol{f}$, not in an crystallographic slip plane. The angles describing how close the dislocation is to cross slipping are shown.

The derived coefficients B_{glide} and B_{sclimb} can be used to construct the drag tensor as usual, shown in equation 3.6, using the slip plane that the force vector acting on the segment is closest to lying in. This means that θ_c always lies between $0°$ and $\frac{\phi}{2}$. When only considering the $\{110\}$ slip planes, ϕ is $60°$ and the plot of the combined glide behaviour is shown in Figure 3.9.

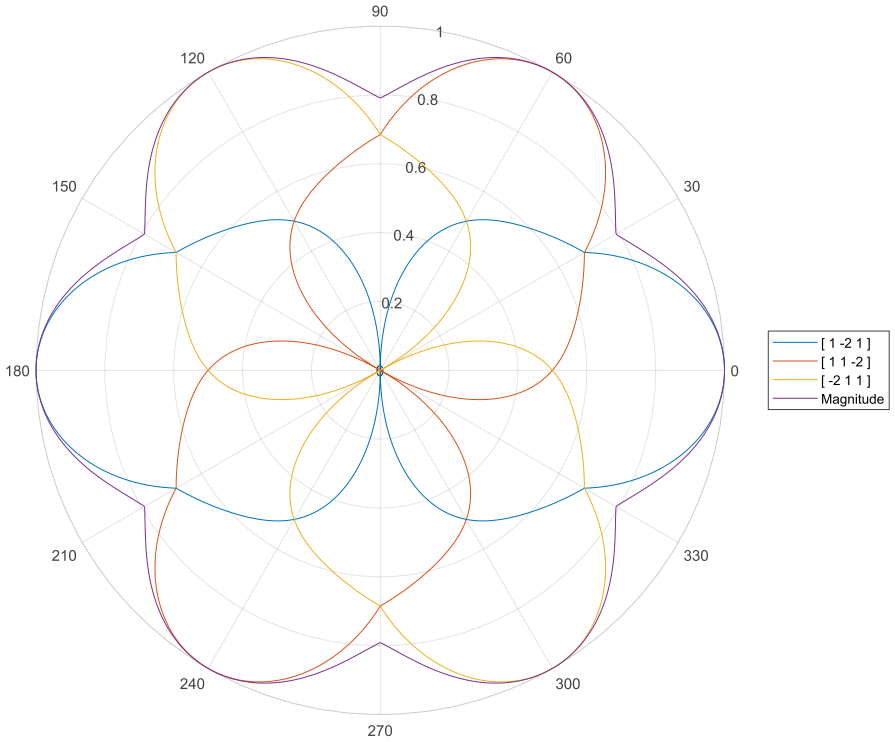


Figure 3.9: A possible model for screw glide behaviour. The drag coefficients are interpolated to allow for a smooth change between slip system regimes. The component of the velocity in each of the glide directions as well as the magnitude of the velocity are shown.

Figure 3.9 demonstrates appropriate behaviour for a BCC screw dislocation. The magnitude of the velocity can be seen to change appropriately, being greatest when the force is applied in viable slip planes and lowest when the force is applied along a plane exactly halfway between two viable slip planes. Similarly, the glide directions expected contribute the most to the resultant velocity direction when the force is close to being in plane. When the force is applied at the halfway between slip systems there is no certainty as to which glide direction is most desired, so the to from the closest slip systems contribute equally. This is not a viable slip direction though, so the magnitude of this velocity is less. The dominance of given slip directions change relatively smoothly so that there is no sudden change in dislocation behaviour if the force on the segment changes gradually.

There are further improvements that can be made to this model of screw behaviour. Adding additional planes is possible, but ϕ must be adjusted accordingly. Adding just the $\{112\}$ type planes means that ϕ is 30°, and adding both $\{112\}$ and $\{123\}$ type planes makes it 15°. Preferential slip planes can be modelled by using a different value of B_{screw} for different plane types. Additionally, more sophisticated interpolations may be used to capture more subtle changes in behaviour as the direction of the force on the dislocation changes. Further complexity can

be modelled by looking at the force per unit length of a segment and only considering slip planes after it reaches some critical value. It should be noted that the combined drag tensor of a node will prevent cross-slip when one segment is not screw type, since the component drag tensors will be constructed using different slip systems. This means that two consecutive screw segments are required for cross-slip to initiate, implicitly setting a minimum required screw dislocation length of twice the minimum segment length before cross-slip can occur.

3.5 Summary

The mobility law acts as a major workhorse of any DDD simulation. Introducing the interpolation for B_{line} has made the drag tensor easier to invert and reduced run times in test simulations by two to three orders of magnitude, without compromising how well the simulation reflects reality. The new BCC mobility law applies these advantages to a specific crystallography, as well as solving the issue of pencil glide of screw dislocations. This new code allows for the accurate simulation of $\frac{a}{2}[111]$ BCC slip systems and prevents non-physical "free roaming" dislocations that have previously been observed. We can now try to use this set up to simulate finite volumes in *EasyDD*.

Chapter 4

Combining DDD with FEM

4.1 Introduction

With the improvements made to improve the performance and functionality of the DDD part of the code, it can now be used in combination with FEM to simulate simple physical objects. However, there are various considerations that must be made to do this properly so that the simulation performs efficiently and the results obtained are meaningful.

4.2 Displacement Rate

One of the major differences between a DDD model in infinite space and one confined to a finite volume with an FE framework is the complexity of the applied stress field. When using the base *DDLab* code, an infinite DDD problem can only have a homogeneous applied stress tensor, which can change with time if required. However when using FEM, the stress state can be different within each individual element, allowing the applied stress field to vary with spacial position in addition to time. The way the stress field varies is defined by setting boundary conditions. In *EasyDD*, we use displacement and force boundary conditions. A stress field is generated by having one or more surfaces fixed in position and then causing another surface to move in a controlled manner. Linear elasticity can then be used to determine the stress that results in the volume, with the differences in simulated materials caused by the shear modulus and Poisson's ratio that are used. The movement of the surface can either be force-controlled, where a known force is applied and displacement is left independent, or displacement-controlled, where the opposite is the case. This is useful for simulating micromechanical tests performed with a nanoindenter, which can also be either displacement or force-controlled. For the purposes of this discussion, we will be examining the issues

surrounding displacement-controlled simulations.

Any change in the stress state between time increments must be kept small enough to ensure the validity of the predictions made by the time integrator as well as maintaining quasi-static conditions. Since the time integrator does not take factors such as a displacement rate or free surfaces into account, there must be a maximum possible time increment, Δt_0, defined so that the largest possible change in the stress state at any given point in the simulated space does not exceed some previously defined tolerance.

This brings us to the matter of selecting an appropriate displacement rate. An applied displacement rate will cause a change in the applied stress field every time increment, proportional to the size of the previous time increment. This can be used to model several mechanical tests, such as pillar compression, cantilever bending and tensile testing. Under a constant stress, a given dislocation network will continually enable deformation of the material it is present in, unless it reaches an equilibrium state, which it may not necessarily do. The applied stress itself will also result in a fixed amount of elastic deformation. In order to correctly capture plastic behaviour, the rate of elastic displacement must be negligible compared to the rate of plastic displacement that occurs when dislocations are freely moving. Essentially, after yield has occurred, dislocations action should be the main cause of strain in the material. However, if an applied displacement rate is chosen that is too small, it will take many more time increments to simulate the same behaviour. Thus it is important to make the displacement rate as large as possible without making it so large that the entirety of plastic behaviour is not captured. It should be noted that, given these constraints, there is a limit to the kinds of mechanical tests that can be simulated. Rapid loading that involves rate dependent effects cannot be modelled for two reasons. Firstly, operations to reproduce inertial and elastodynamic effects are not included in the current version of the code. Secondly, loading rates high enough to cause these effects would not be compatible with the quasi-static conditions and the mathematical framework would need to be reevaluated to verify its validity, with appropriate changes being made. This is a highly complex problem and beyond the scope of this thesis.

At first glance, it appears that the displacement rate is dependant on Δt_0 and vice versa. One might think that so long as the maximum stress change between two time increments remains suitably incremental, the displacement rate and maximum time increment can be adjusted to any values that satisfy this condition in order to maintain quasi-static conditions. This would be a false assumption however. To understand why this is, we need to look at the interplay between dislocation behaviour and elastic behaviour in the model. Dislocations produce forces and displacements on all surfaces of the FEM problem. For a displacement-controlled

boundary condition, displacements caused by dislocations alter the amount of "external" displacement that needs to be applied and forces caused by dislocations alter the amount of force required to cause this displacement. Measuring the applied force can hence yield a force/displacement plot for comparison to real world data. Unfortunately, the forces and displacements caused by dislocations cannot increase freely as the loading rate increases.

Consider how a dislocation network contributes to plastic displacement for a bending cantilever. The change in displacement on the loaded surface in the direction of applied displacement caused by dislocations of a given slip system can be estimated by:

$$\Delta u_{dis} = \rho A(\boldsymbol{b} \cdot \bar{\boldsymbol{u}}) \tag{4.1}$$

In this equation Δu_{dis} is the displacement contribution on the loaded surface in the load direction made by dislocations of the considered slip system, ρ is the dislocation density, A is the mean area swept by a unit length of dislocation during a time increment, $\boldsymbol{b}$ is the Burgers vector and $\bar{\boldsymbol{u}}$ is the unit vector in which the displacement is applied. The two variables which are not constant are A and ρ, provided there are dislocation sources or other means of generating dislocations in the model. Let us now compare plastic displacement that occurs with the displacement boundary condition. During a time increment velocity is constant, so A is the mean of the magnitude of the dislocation velocity per unit length, $|\bar{\boldsymbol{v}}|$, multiplied by the size of the time increment, Δt. Since neither $\boldsymbol{b} \cdot \bar{\boldsymbol{u}}$ nor a unit length of a dislocation change, we can make them a constant, C, with units of m^2:

$$\begin{aligned} \Delta u_{tot} &= \dot{u}\Delta t \\ \Delta u_{dis} &= C\rho|\bar{\boldsymbol{v}}|\Delta t \\ \Delta u_{tot} &= \Delta u_{dis} + \Delta u_{elas} \end{aligned} \tag{4.2}$$

Here, Δu_{tot} is the total displacement change between time increments on the surface with the enforced boundary condition, $\dot{u}$ is the applied displacement rate and Δu_{elas} is the necessary change in the applied elastic displacement to maintain the boundary conditions. It can be easily seen that if $\dot{u} >> C\rho|\bar{\boldsymbol{v}}|$ then Δu_{elas} must be very big. This means that when the loading rate is too large, the simulated volume appears to behave completely elastically, regardless of Δt. Conversely, if $\dot{u} < C\rho|\bar{\boldsymbol{v}}|$ then Δu_{elas} must be negative to satisfy the boundary conditions. Essentially this means that displacement is applied in the opposite direction to the loading direction in order to reduce the stress induced. This slows down dislocation

motion, unnecessarily increasing number of time increments required, and hence the time taken to run the simulation. Given that $|\bar{\boldsymbol{v}}|$ is roughly constant for a given applied stress, the important variable is ρ. The larger ρ is, the more dislocations there are in the material which can cause plastic displacement. This means the optimum $\dot{u}$ is larger for larger values of ρ. A similar effect can be observed when using force boundary conditions instead of displacement boundary conditions but explicitly making the mathematical argument is more complicated because surface tractions are dependent on additional variables, such as proximity of segments to the surface and orientation of segments relative to the surface. We should also remember that this argument for displacement boundary conditions is also only a rough estimate of $\dot{u}$ for a single slip system. Similar issues arise when dealing with force-controlled boundary conditions and trying to calculate an appropriate loading rate. The way the dislocation density changes during a simulation also has additional nuance depending on the starting dislocation network. A given initial dislocation structure will have a certain potential for plastic strain. This is proportional to the dislocation density that the evolution of this system can produce. A single dislocation loop allows for limited plastic strain, a Frank-Read source for much more given that it can theoretically produce an indefinite number of dislocations. Several Frank-Read sources or other more complicated structures can exponentially increase the amount of plastic strain possible in the simulated volume. The general conclusion can be drawn that the applied strain rate needs to be chosen so that it is appropriate for the dislocation structure being simulated but directly calculating what this strain rate should be is impractical.

To summarise, if the stress conditions of a simulation are changed too rapidly, the dislocation network will be unable to evolve fast enough to cause a sufficient degree of plastic strain and the simulation will make up the deficit in necessary strain to allow for a given applied displacement by adding more elastic strain. This causes a non-physical result which we do not wish to occur. Similarly, if a displacement rate is chosen that is too small, dislocation action may cause more plastic strain than the simulation is attempting to cause, resulting in a reduction in the applied force so that dislocation action is reduced. This unnecessarily results in a dislocation structure taking more time to evolve to the same situation it would have reached anyway.

Since trying to estimate a potential displacement rate analytically is non-trivial, it has become common practice to find an appropriate loading rate using convergence studies. These studies are fairly simple. A simulation is run with a high initial displacement rate and the stress-strain data is recorded. The displacement rate is then reduced and the same simulation is run again. This process is repeated until the stress-strain data remains the same (within error). At this

point, the largest displacement rate with the observed constant stress-strain behaviour can be used for this simulation and any simulations with similar starting parameters. In principle this is very easy, however when dealing with multiple slip systems, some complications were observed.

The following example comes from a convergence study performed on a simulation of a BCC cantilever containing 24 sources of 3 different {101} slip systems.

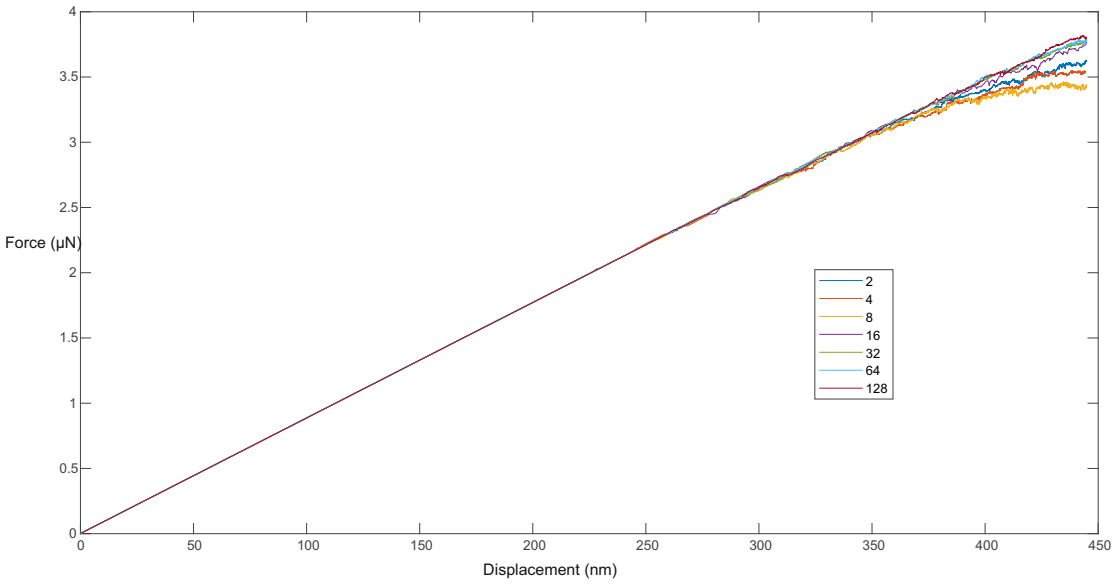


Figure 4.1: Force displacement data of a convergence study. Units for the loading rates are arbitrary.

Figure 4.1 shows the force-displacement data for multiple simulations run with the identical parameters except for the displacement rate. From this data, the proportionality limit can be obtained by plotting the gradient against the displacement and observing the region where the gradient ceases to be constant. This can then be plotted against displacement rate, as shown in Figure 4.2. This feature is common in all of the data sets and so make a convenient point of comparison.

Normally, we would expect to see a continual decrease in proportionality limit as displacement rate is decreased, especially since the displacement rate is on a logarithmic scale. The fact that this is not the case demonstrates that there are additional factors not initially taken into account. At high loading rates, the large forces on dislocations cause the time integrator to slow the simulation down, resulting in the proportionality limit hitting a roughly constant value rather than allowing the simulation to approach purely elastic behaviour. However, at lower loading rates we still do not seen a continual decrease in the proportionality limit. The exact cause is not entirely clear and additional work may be required beyond the scope of this thesis. However, a brief analysis of the various dislocation structures involved gives a possible answer. Sources of different slip systems have

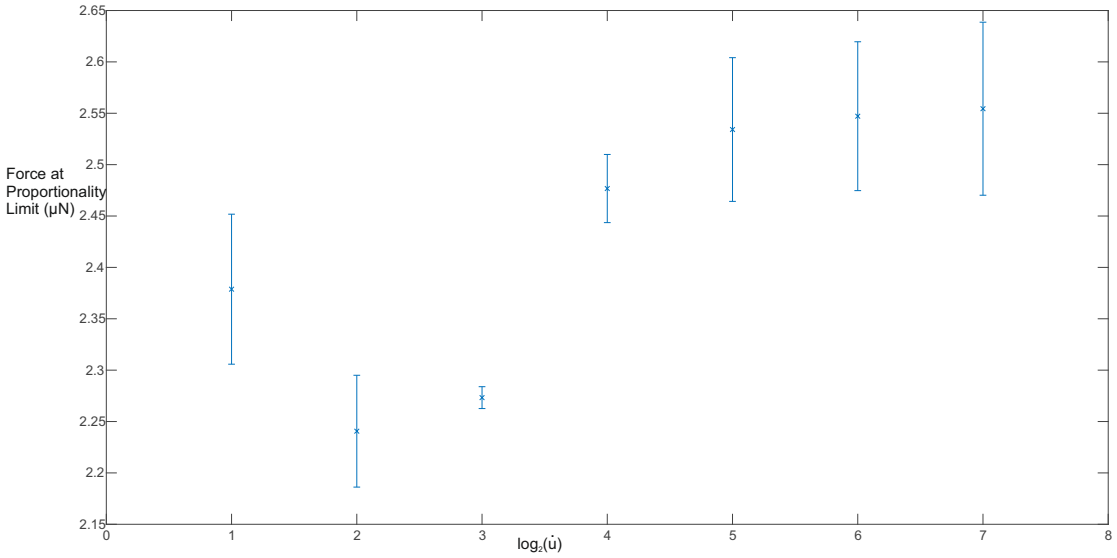


Figure 4.2: The forces at which the proportionality limit is reached for various loading rates, $\dot{u}$. It should be noted that the error bars are the indication of the region in which the proportionality limit lies.

different activation stresses. The degree to which the first slip system activated can evolve before the second slip system activates is different for different displacement rates. This is because the amount of time between each of them reaching their activation stress is different. It seems that at certain displacement rates, the activation of the second slip system impedes the action of sources belonging to the first slip system, reducing the overall plastic strain that the dislocation network can produce. To avoid this as a potential complication, further convergence studies may be performed only using sources of the slip system with the smallest non-zero Schmid factor, and therefore the last to activate. This may produce a displacement rate that is smaller than optimum but still appropriate. It has the additional benefit of being a simpler simulation to run, allowing the convergence study to be performed in less time. An alternative method is to run the convergence study as before but to monitor the corrective elastic displacements. If Δu_{elas} is frequently negative in order to reduce the total displacement, then this can be used as an indication that movement of the dislocation network is responsible the majority of deformation and that the loading rate can be increased. While this is much better at finding an optimum displacement rate for a given simulation setup, it requires much more time to find a suitable value.

4.3 Virtual Segments

Another phenomenon that needs to be accounted for when coupling DDD with FEM is when dislocation segments meet the surfaces of the finite volume being

simulated. Correctly handling these situations is essential for smooth and accurate performance.

The first consideration to make is the behaviour of dislocations immediately before exiting a surface. The stress state of the material through which the dislocations are moving is calculated using the FE framework. Each element of the FE mesh has a stress associated with it. The stress acting on the dislocation is taken from whichever element the midpoint of a segment lies in. There are no elements outside of the volume being simulated and therefore no stress associated with any location outside of the volume. Initially this may seem fine, since we do not wish to simulate any dislocation behaviour outside of this volume. Unfortunately, there is an issue that arises concerning the time integrator.

The time integrator uses a predictor to decide if the change in velocity of a given node is too great during a given time increment. Since there are no elements outside of the volume, if the midpoint of any segment exits the volume then the force acting on that segment as a result of stress in the material is calculated as zero. This in turn causes a reduction of the force contribution of this segment to both of the nodes it is connected to, greatly altering their velocities. This massive change in velocity causes the time integrator to reduce the size of the time step until the segment has a force acting on it. Under these circumstances, a dislocation cannot exit through a surface until a node moves a short enough distance outside of a volume that the midpoint of all of the segments it is attached to remain inside the volume. This means that any attempt of a dislocation to exit the volume unnecessarily causes the entire simulation to progress more slowly. The remedy for this is to apply an appropriate stress to exited segments. The easiest solution is to assign the stress state of the closest element to any exited dislocations when preforming force calculations. When implemented, this seems to improve the situation while not solving the problem entirely. This suggests that there are other complexities involved with the force calculations of exiting segments that have yet to be identified and further investigation may be required.

Once a node has exited a surface, a decision must be made as to what to do with it. Exited nodes can no longer be treated in the same manner as internal nodes since they no longer represent a physical phenomenon in the context of the simulation. However the motion that caused them to exit through the surface is completely legitimate, so the correct treatment of such nodes is important to maintain the veracity of the model. A convenient treatment is to move nodes far away from simulated volume, while still maintaining their connection to the dislocation network, and then ceasing all force calculations on them and their connected segments, making them “virtual” segments. This has the advantage of recording the location of the exiting event and supplying the means to calculate

any slip steps that arise without the presence of these nodes and segments affecting the rest of the simulation.

The position to which a node is moved once it has exited the surface must satisfy several criteria:

- Long range displacements must be correctly calculated.
- The node must be an appropriate distance from the simulated volume.
- The method of moving nodes must be consistent for all possible scenarios.
- It must be possible to find the exit location of a node from its position outside of the volume in order to find the location of a slip step.

Details in calculating displacements will be discussed in greater detail in Chapter 5. When looking at a single extended loop, the long range displacement calculated is relative. This means that as the angle of an extended loop relative to the surface is changed, the displacement on one side of the loop decreases in magnitude whilst the other side increases by the same amount, so that the degree to which the material on one side of the loop is displaced from the other is the same, namely $\boldsymbol{b}$. To maintain consistency, any segments in the same loop exiting out of the same surface must be projected in the same direction. The maths for calculating slip step displacement with an extended loop exactly requires the loop to extend to infinity. This is obviously impractical and using numbers that are too large causes problems with numerical precision. Fortunately, the error in slip step calculation becomes negligible at a distance that is approximately 3 orders of magnitude of the largest dimension of the simulated volume. This is also far enough away that any generated virtual segments from multiple exited nodes will not intersect the volume. For the purposes of this discussion, this distance will be designated as "pseudo-infinity". Finally, virtual segments at pseudo-infinity must be able to be mapped back onto the surface they were projected from in order to recover where they exited. The simplest way to satisfy all of these conditions is to project along the surface normal, as shown in Figure 4.3.

To project exited nodes to the correct position, surface nodes must first be positioned correctly. This is done by finding the point, $\boldsymbol{Q}$, where a segment AB intersects the surface. If node $\boldsymbol{A}$ exits and $\boldsymbol{B}$ remains inside then:

$$\boldsymbol{Q} = \left(\frac{(\boldsymbol{Q_0} - \boldsymbol{B}) \cdot \boldsymbol{n}}{(\boldsymbol{A} - \boldsymbol{B}) \cdot \boldsymbol{n}} \right) (\boldsymbol{A} - \boldsymbol{B}) + \boldsymbol{B} \tag{4.3}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are the position vectors of the external and internal nodes respectively, $\boldsymbol{Q_0}$ is an arbitrary point on the surface and $\boldsymbol{n}$ is normal to the surface through which $\boldsymbol{A}$ exited, see figure 4.3. It should be noted that it is important

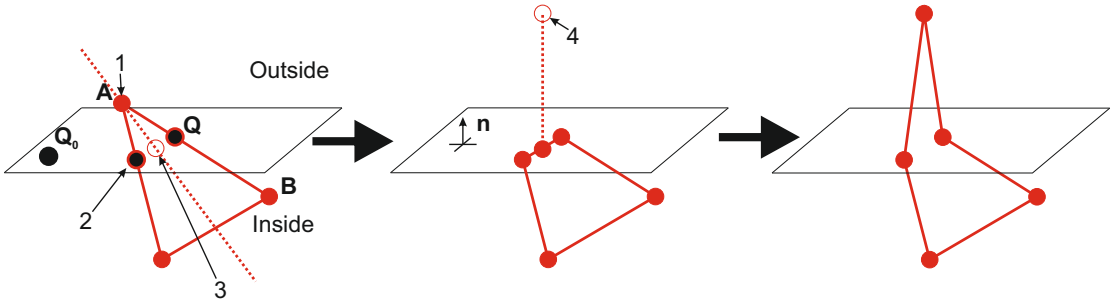


Figure 4.3: The process by which virtual nodes and segments are created when a node exits out of a surface. 1. A node at $\boldsymbol{A}$ which has exited is detected. 2. A new surface node at $\boldsymbol{Q}$ is created where each segment connecting the exited node to an internal node intersects the surface; calculated using equation (4.3). 3. The exited node is moved back onto the surface along the line of the sum of segments connected to it. 4. The exited node is moved to pseudo-infinity along the surface normal. Virtual nodes follow the line of the slip step, positioned at pseudo-infinity and can be directly projected back onto the surface along the surface normal to recover the slip step position.

that nodes are projected consistently from the position of the slip step in the direction of the surface normal. This is because, while there is no effect on the long range displacement fields, the position and shape of the slip step is dependent on the virtual dislocation network being formed correctly. Surface nodes are also restricted so that they may only move on the surface through which the dislocation has exited. If a surface node is no longer connected to any internal dislocations because all connecting segments have exited, then it is projected to pseudo-infinity as part of the virtual network.

The final scenario that must be considered is how to treat nodes that attempt to exit out of a surface where a zero displacement boundary condition is applied (i.e. a fixed surface). Forming a slip step on such surfaces greatly alters the equation for satisfying the boundary conditions, which results in a massive change in the stress state of the simulated volume. Additionally, these surfaces often represent where a test sample joins the bulk of the material it is attached to, such as the base of a micro-pillar or cantilever, and the formation of slip steps on these surfaces may be non-physical. The most practical method is to treat such surfaces as impenetrable. This means that any exited nodes can be traced back to the point where they exited from this surface using the nodal velocity and then fixed in this position.

4.4 Variable Selection

When calibrating DDD models to work coupled with FEM, there are a few additional variable that need to be appropriately calibrated aside from those already mentioned.

The dimensions of the FE elements normally need to be set to balance resolution of the model against performance. More, smaller elements give better resolution while having a greater computational expense. However, when incorporating DDD, and additional factor must be taken into account. If the change in stress state between two elements is too large, then this may cause the time integrator to reduce time interval size in a similar way as discussed with surfaces previously. The exact calibration of this depends on loading conditions but the key factor to keep in mind is that the greater the expected stress gradient is in a given dimension, the smaller the element size must be.

Another consideration to be taken into account is the maximum length a dislocation segment can be, l_{max}. A shape function is used to find the forces acting on nodes as a function of the forces acting on the segments they are connected to. As mentioned previously, the applied stress on a segment is assumed to be that associated with the element in which the midpoint of the segment is positioned. It is important to calibrate l_{max} so that the midpoint is not in a vastly different stress state to that of the node, so that its motion is as a result of the forces acting on it is accurately simulated. This again is dependent on the stress state but if the element size is appropriately calibrated then a good rule of thumb is that segments should not be long enough to cross more than two elements.

Finally, we must take into consideration the maximum size of a time increment, Δt_0. Even if the strain rate is well calibrated, Δt_0 must still be used to ensure that the maximum change in stress in the material is relatively small. This not only depends on the strain rate but in the loading conditions For different loading conditions and different shaped volume, the position of the element with the greatest change in stress will be different. The greatest possible stress change should be used as a reference when calculating suitable Δt_0, so that the stress change in all elements per time increment is appropriately small.

As an example of how these may be calculated, consider a square cross sectioned cantilever, as shown in Figure 4.4.

Beam bending mechanics predict that the stress as a result of the applied load will vary along the x and z axes. Thus it may be appropriate to have smaller element dimensions in these directions than in the y axis in order to optimise performance. Once the element size has been chosen it then becomes simple to assign an appropriate value to l_{max}. In order to calibrate Δt_0, we must now

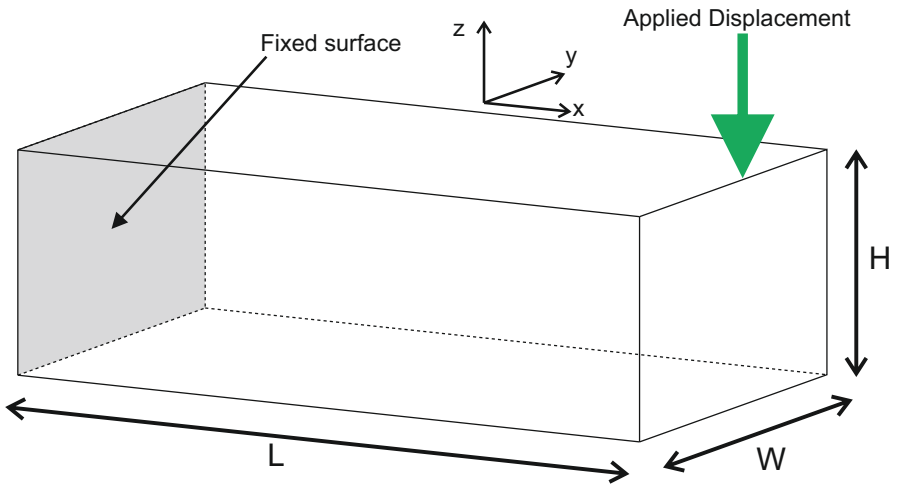


Figure 4.4: A simple cantilever with height H, width W and length L. The cross section is square, so $W = H$.

calculate the maximum change in stress as a function of displacement rate. For a square cantilever, this is:

$$\Delta\sigma_{max} = \frac{3\dot{u}\mu(1+\nu)H}{2L^2} \tag{4.4}$$

where $\dot{u}$ is the applied displacement rate, μ is the shear modulus and ν is the Poisson's ratio.

Now that we know $\Delta\sigma_{max}$, we can set Δt_0 so that $\Delta t_0 \Delta\sigma_{max} < \sigma_{tol}$, where σ_{tol} is some control parameter. The exact value of σ_{tol} depends on the drag coefficients used in the mobility law. Larger drag coefficients mean that larger changes in stress can be tolerated whilst maintaining a relatively small change in the velocity of dislocation segments in a given time period. This is because the drag coefficient is the variable that defines the ratio between real time and simulated time. This means that increasing the loading rate while simultaneously decreasing the drag coefficients by a proportionate factor causes no change to the simulation.

4.5 Summary

We have considered various factors to ensure effective coupling between DDD and FEM techniques in order to build a robust model. With careful calibration of input variables we can ensure the quasi-static framework is maintained for valid use of the DDD method whilst simultaneously using FEM to model the response from the material in which the dislocations are moving. We have also added necessary operations to manage dislocations attempting to exit the simulated volume. It is now important to ensure that we have the means to enforce the boundary

conditions before we can perform a simulation.

Chapter 5

A New Method for Calculating Displacement Boundary Conditions

5.1 Introduction

When using the superposition method to combine DDD with FEM, it is important to properly enforce the boundary conditions for a finite domain. Previously, it has been difficult to properly calculate the displacements that dislocations cause on the surface of a volume for all circumstances. This puts a limit on the full functionality of this type of model, making it difficult to accurately simulate the displacement field when dislocations move out of their glide plane, such as cross-slip, climb and junction formation. However, with a newly developed method, we can accurately and efficiently measure these displacements in order to better satisfy displacement boundary conditions.

5.2 Adapting Barnett's Solution

5.2.1 A Non-Planar Method

In order to calculate the displacement field contribution of a single finite straight dislocation segment, we require an analytic expression that describes the displacement contribution for an arbitrary segment that can be easily implemented into a dislocation dynamics code without the need to store any history information. Starting from the earlier expression in Hirth and Lothe [62], Barnett [66, 72] has derived an expression for the displacement, $\tilde{\boldsymbol{u}}$, produced by a triangular dislocation

loop ABC at a given field point, $\boldsymbol{X}$

$$\boldsymbol{u}(\boldsymbol{X}) = \frac{-\boldsymbol{b}\Omega_{ABC}}{4\pi} - \frac{1-2\nu}{8\pi(1-\nu)}\left[\boldsymbol{f_{AB}} + \boldsymbol{f_{BC}} + \boldsymbol{f_{CA}}\right] + \frac{1}{8\pi(1-\nu)}\left[\boldsymbol{g_{AB}} + \boldsymbol{g_{BC}} + \boldsymbol{g_{CA}}\right] \tag{5.1}$$

$$\Omega_{ABC} = -4 \cdot \mathrm{sgn}(\boldsymbol{\lambda_A} \cdot \boldsymbol{n}) \arctan\left(\sqrt{\tan\left(\frac{s}{2}\right)\tan\left(\frac{s-a}{2}\right)\tan\left(\frac{s-b}{2}\right)\tan\left(\frac{s-c}{2}\right)}\right) \tag{5.2}$$

$$\boldsymbol{f_{AB}} = (\boldsymbol{b} \times \boldsymbol{l_{AB}}) \ln\left(\frac{|\boldsymbol{R_B}|}{|\boldsymbol{R_A}|}\frac{1+\boldsymbol{\lambda_B}\cdot\boldsymbol{l_{AB}}}{1+\boldsymbol{\lambda_A}\cdot\boldsymbol{l_{AB}}}\right) \tag{5.3}$$

$$\boldsymbol{g_{AB}} = [\boldsymbol{b}\cdot(\boldsymbol{\lambda_A}\times\boldsymbol{\lambda_B})]\frac{\boldsymbol{\lambda_A}+\boldsymbol{\lambda_B}}{1+\boldsymbol{\lambda_A}\cdot\boldsymbol{\lambda_B}} \tag{5.4}$$

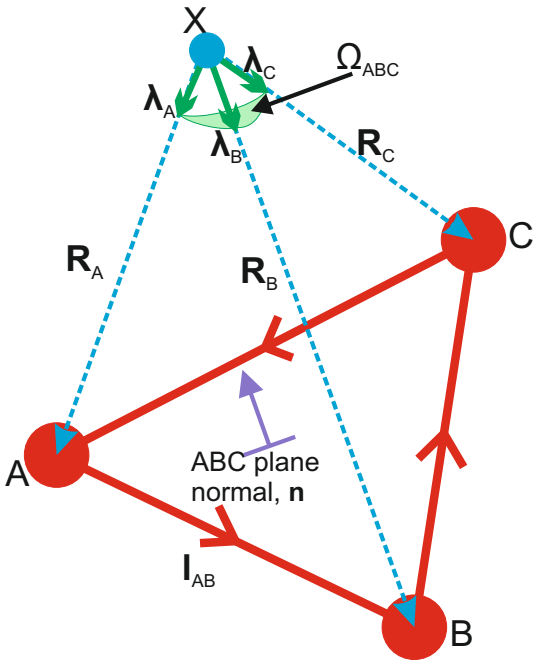


Figure 5.1: Barnett's solution gives the displacement as a result of any triangle, ABC, at any field point, $\boldsymbol{X}$. The Ω_{ABC} term refers to the solid angle at $\boldsymbol{X}$ and $\boldsymbol{f_{AB}}$ and $\boldsymbol{g_{AB}}$ are vectors associated with segment AB.

In these expressions, $\boldsymbol{b}$ is the Burgers vector of the loop, ν is Poisson's ratio for the material, $\boldsymbol{n}$ is the normal to the plane containing ABC, $\boldsymbol{R_A} = \boldsymbol{A} - \boldsymbol{X}$ is the vector from the field point $\boldsymbol{X}$ to $\boldsymbol{A}$, and so on for $\boldsymbol{R_B}$ and $\boldsymbol{R_C}$, and $\boldsymbol{\lambda_A}$, $\boldsymbol{\lambda_B}$ and $\boldsymbol{\lambda_C}$ are their respective normalised vectors, i.e. $\boldsymbol{\lambda_A} = \frac{\boldsymbol{R_A}}{|\boldsymbol{R_A}|}$ as shown in Figure 5.1. The angles between these vectors are a, b and c, where $a = \arccos(\boldsymbol{\lambda_B}\cdot\boldsymbol{\lambda_C})$, $b = \arccos(\boldsymbol{\lambda_A}\cdot\boldsymbol{\lambda_C})$ and $c = \arccos(\boldsymbol{\lambda_A}\cdot\boldsymbol{\lambda_B})$ and s is defined as: $s = \frac{a+b+c}{2}$. The $\boldsymbol{l_{AB}}$ term is the normalised line direction of AB. The two terms $\boldsymbol{f_{AB}}$ and $\boldsymbol{g_{AB}}$ describe the elastic displacement due to segment AB. Note that $\boldsymbol{f_{AB}} = \boldsymbol{0}$ for a pure screw segment and both $\boldsymbol{f_{AB}} \to 0$ and $\boldsymbol{g_{AB}} \to 0$ as the distance between

the field point and the segment becomes much greater than the segment length. Whereas $\boldsymbol{f_{AB}}$ and $\boldsymbol{g_{AB}}$ are both undefined when the field point is on the line segment AB. The Ω_{ABC} term is the solid angle. Mathematically Ω is the area projected by the triangle on a unit sphere located at the field point $\boldsymbol{X}$. Physically the solid angle accounts for the plastic slip step. The slip step on a free surface is left behind by a partially exited loop and is a prominent feature of micromechanical test data. The solid angle, Ω, is in the range $-2\pi \leq \Omega \leq 2\pi$, meaning the first term in (5.1) jumps between $\pm\boldsymbol{b}/2$ either side of the slip plane area bounded by the triangular loop. Note that the displacement due to an isolated segment is ill defined, because of the Ω term. Importantly the Barnett solution outlined in equations (5.1)-(5.4) has an advantage over the solution described by Hirth and Lothe [62]. It does not make the assumption that $\boldsymbol{b}$ lies in the habit plane of the loop (the plane containing $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$) when calculating the solid angle. That is to say the Barnett solution has no restriction on $\boldsymbol{b}$ and so remains valid when the Burgers vector has a prismatic component.

Barnett's solution can be seen to work for any loop by considering the treatment of prismatic loops presented by Fivel and Depres [67]. They posit that a prismatic loop can be assigned a "mirror loop" of opposite sign. These two prismatic loops can be thought of as four shear loops for which the displacement can be calculated. If the mirror loop is then moved to infinity, the displacement calculation for field points local to the original prismatic loop is mathematically equivalent to the same calculation carried out for this loop by itself because of how the $\boldsymbol{f_{AB}}$ and $\boldsymbol{g_{AB}}$ of the mirror loop reduce to zero while the Ω terms are equivalent.

Using Barnett's solution, a contribution to the total displacement from each segment in any closed loop can be found at any given field point by dividing the loop into triangles, shown in Figure 5.2a. Each triangle consists of one real segment which is part of the loop, and two non-real segments which join to a closure point P, common with all other segment triangles. The extra elastic terms cancel, as illustrated in Figure 5.2b, and the solid angle terms for each triangle combine to give the complete solid angle of the whole loop. This works because:

$$\begin{aligned} \boldsymbol{f_{AB}} &= -\boldsymbol{f_{BA}} \\ \boldsymbol{g_{AB}} &= -\boldsymbol{g_{BA}} \\ \Omega_{ABC} &= \Omega_{ABP} + \Omega_{BCP} + \Omega_{CAP} \end{aligned} \tag{5.5}$$

Previously, $\boldsymbol{P}$ has been chosen to be in the plane of the loop [67]. This is problematic when attempting to calculate the result for non-planar loops or multiple loops with varying positions and orientations. Fivel and Depres [67] devised

a method for splitting up loops with arbitrary geometries into triangles using a projection which is on the slip plane along the Burgers vector to find a different closure point for each slip plane so that the entire loop is properly divided. The closure point is arbitrary but the total area of all virtual triangular loops is equal to the area of the original loop. This solution allows for calculation of displacements at any field point for the majority of dislocation structures, including segments which have cross slipped. However this solution can be inconvenient with large dislocation networks since the Burgers vector history of all segments must be tracked in order to correctly split loops that have been involved in dislocation collisions. Therefore we have developed a new method which uses a single closure point, $\boldsymbol{P}$, for all segments in the model as illustrated in Figure 5.2(b).

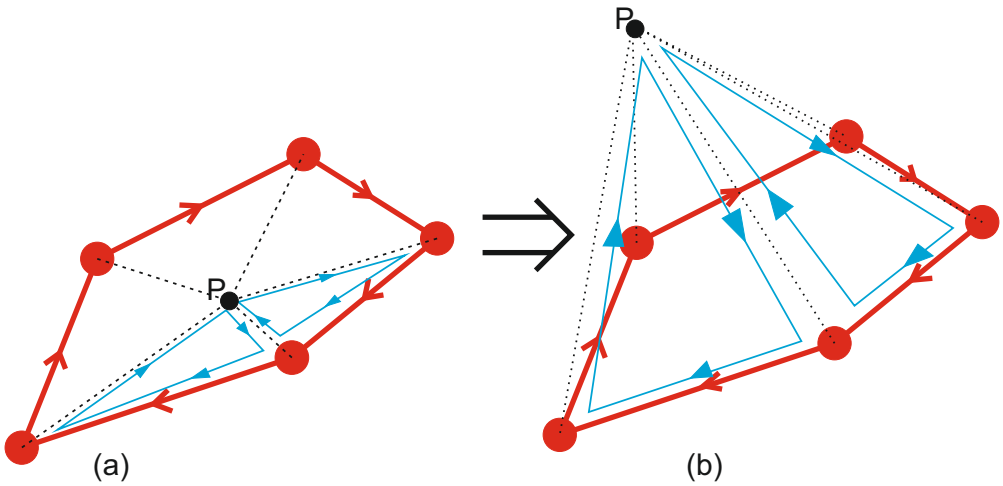


Figure 5.2: An arbitrary loop can be broken into triangles in order to calculate the displacement contribution. This can be done in the plane of the loop as in (a) which requires a different closure point for each slip plane. Or a unique closure point can be used which is not in the habit plane of the loop as in (b). For the correct combination of the triangles, adjacent sides of different blue triangles must be in opposite directions.

Barnett's equation was previously only known to apply to planar loops. Here we validate the use of a single closure point to allow non-planar loops to be evaluated. First, we demonstrate that the closure point, required to convert a segment into a triangular loop, does not need to be in the plane of the segment. Changing the line direction changes the sense of the loop and $\boldsymbol{u} \to -\boldsymbol{u}$. First consider $\boldsymbol{f_{AB}}$:

$$\boldsymbol{l_{BA}} = -\boldsymbol{l_{AB}} \to \boldsymbol{f_{BA}} = -(\boldsymbol{b} \times \boldsymbol{l_{AB}}) \ln \left(\frac{|\boldsymbol{R_A}|}{|\boldsymbol{R_B}|} \frac{1 - \boldsymbol{\lambda_A} \cdot \boldsymbol{l_{AB}}}{1 - \boldsymbol{\lambda_B} \cdot \boldsymbol{l_{AB}}} \right) \tag{5.6}$$

As the vectors are normalised, $\boldsymbol{\lambda_A} \cdot \boldsymbol{l_{AB}}$ and $\boldsymbol{\lambda_B} \cdot \boldsymbol{l_{AB}}$ are the cosines of angles associated with triangle ABX as shown in Figure 5.3. Since $\sin(\pi - \theta) = \sin\theta$, then starting from the sine rule:

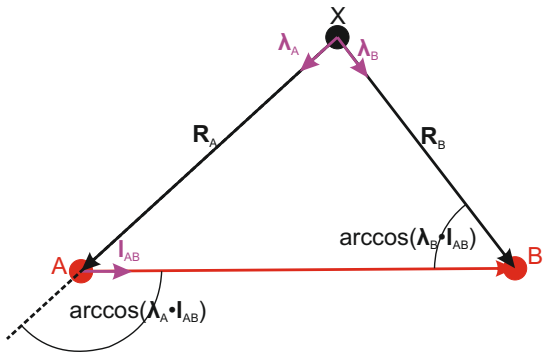


Figure 5.3: triangle formed by dislocation line AB and field point $\boldsymbol{X}$.

$$\frac{\sin(\arccos(\boldsymbol{\lambda_A}\cdot\boldsymbol{l_{AB}}))}{|\boldsymbol{R_B}|}=\frac{\sin(\arccos(\boldsymbol{\lambda_B}\cdot\boldsymbol{l_{AB}}))}{|\boldsymbol{R_A}|} \tag{5.7}$$

$$\rightarrow\frac{\sin^2(\arccos(\boldsymbol{\lambda_A}\cdot\boldsymbol{l_{AB}}))}{|\boldsymbol{R_B}|^2}=\frac{\sin^2(\arccos(\boldsymbol{\lambda_B}\cdot\boldsymbol{l_{AB}}))}{|\boldsymbol{R_A}|^2} \tag{5.8}$$

$$\rightarrow\frac{1}{|\boldsymbol{R_B}|^2}(1-(\boldsymbol{\lambda_A}\cdot\boldsymbol{l_{AB}})^2)=\frac{1}{|\boldsymbol{R_A}|^2}(1-(\boldsymbol{\lambda_B}\cdot\boldsymbol{l_{AB}})^2) \tag{5.9}$$

$$\rightarrow\frac{|\boldsymbol{R_A}|}{|\boldsymbol{R_B}|}\frac{1-\boldsymbol{\lambda_A}\cdot\boldsymbol{l_{AB}}}{1-\boldsymbol{\lambda_B}\cdot\boldsymbol{l_{AB}}}=\frac{|\boldsymbol{R_B}|}{|\boldsymbol{R_A}|}\frac{1+\boldsymbol{\lambda_B}\cdot\boldsymbol{l_{AB}}}{1+\boldsymbol{\lambda_A}\cdot\boldsymbol{l_{AB}}} \tag{5.10}$$

$$\therefore\boldsymbol{f_{BA}}=-\boldsymbol{f_{AB}} \tag{5.11}$$

where equation (5.9) is obtained using $\sin^2\theta=1-\cos^2\theta$. Similarly, for $\boldsymbol{g_{AB}}$:

$$\boldsymbol{g_{BA}}=[\boldsymbol{b}\cdot(\boldsymbol{\lambda_B}\times\boldsymbol{\lambda_A})]\frac{\boldsymbol{\lambda_B}+\boldsymbol{\lambda_A}}{1+\boldsymbol{\lambda_B}\cdot\boldsymbol{\lambda_A}} \tag{5.12}$$

$$=-[\boldsymbol{b}\cdot(\boldsymbol{\lambda_A}\times\boldsymbol{\lambda_B})]\frac{\boldsymbol{\lambda_A}+\boldsymbol{\lambda_B}}{1+\boldsymbol{\lambda_A}\cdot\boldsymbol{\lambda_B}} \tag{5.13}$$

$$=-\boldsymbol{g_{AB}}. \tag{5.14}$$

Therefore if we define

$$\boldsymbol{F_{AB}}=-\frac{1-2\nu}{8\pi(1-\nu)}\boldsymbol{f_{AB}}+\frac{1}{8\pi(1-\nu)}\boldsymbol{g_{AB}}=-\boldsymbol{F_{BA}} \tag{5.15}$$

then a triangular loop ABC connected to closure point P will have terms

$$\boldsymbol{F_{AB}}+\boldsymbol{F_{BP}}+\boldsymbol{F_{PA}}+\boldsymbol{F_{BC}}+\boldsymbol{F_{CP}}+\boldsymbol{F_{PB}}+\boldsymbol{F_{CA}}+\boldsymbol{F_{AP}}+\boldsymbol{F_{PC}} \tag{5.16}$$

$$=\boldsymbol{F_{AB}}+\boldsymbol{F_{BP}}-\boldsymbol{F_{AP}}+\boldsymbol{F_{BC}}+\boldsymbol{F_{CP}}-\boldsymbol{F_{BP}}+\boldsymbol{F_{CA}}+\boldsymbol{F_{AP}}-\boldsymbol{F_{CP}} \tag{5.17}$$

$$=\boldsymbol{F_{AB}}+\boldsymbol{F_{BC}}+\boldsymbol{F_{CA}} \tag{5.18}$$

so the contribution to $\boldsymbol{u}$ from $\boldsymbol{F}$ is independent of the closure point as the virtual segments going to and from $\boldsymbol{P}$ always cancel. We now demonstrate that the solid angle terms can also be calculated for an out of plane closure point.

The solid angle of a polygon as seen from a field point is the projection of the area of the polygon onto a unit sphere centered at the field point. Barnett [66] obtained the following closed form expression for the solid angle of a triangular loop ABC at a field point $\boldsymbol{X}$:

$$\Omega_{ABC} = -4 \cdot \operatorname{sgn}(\boldsymbol{\lambda_A} \cdot \boldsymbol{n}) \arctan\left(\sqrt{\tan\left(\frac{s}{2}\right)\tan\left(\frac{s-a}{2}\right)\tan\left(\frac{s-b}{2}\right)\tan\left(\frac{s-c}{2}\right)}\right) \tag{5.19}$$

$$\boldsymbol{R_A} = \boldsymbol{A} - \boldsymbol{X} \tag{5.20}$$

$$\boldsymbol{\lambda_A} = \frac{\boldsymbol{R_A}}{|\boldsymbol{R_A}|} \tag{5.21}$$

$$\begin{aligned} \boldsymbol{n} = (\boldsymbol{B}-\boldsymbol{A}) \times (\boldsymbol{C}-\boldsymbol{B}) &= (\boldsymbol{B}-\boldsymbol{X}-(\boldsymbol{A}-\boldsymbol{X})) \times (\boldsymbol{C}-\boldsymbol{X}-(\boldsymbol{B}-\boldsymbol{X})) \\ &= (\boldsymbol{R_B}-\boldsymbol{R_A}) \times (\boldsymbol{R_C}-\boldsymbol{R_B}) \\ &= (\boldsymbol{R_A}\times\boldsymbol{R_B}) + (\boldsymbol{R_B}\times\boldsymbol{R_C}) + (\boldsymbol{R_C}\times\boldsymbol{R_A}) \end{aligned} \tag{5.22}$$

$$a = \arccos(\boldsymbol{\lambda_B}\cdot\boldsymbol{\lambda_C}), \quad b = \arccos(\boldsymbol{\lambda_A}\cdot\boldsymbol{\lambda_C}), \quad c = \arccos(\boldsymbol{\lambda_A}\cdot\boldsymbol{\lambda_B}) \tag{5.23}$$

$$s = \frac{a+b+c}{2}$$

In these expressions, $\boldsymbol{n}$ is normal to the plane containing ABC, $\boldsymbol{R_A}$ is the vector from the field point $\boldsymbol{X}$ to $\boldsymbol{A}$, and so on for $\boldsymbol{R_B}$ and $\boldsymbol{R_C}$, and $\boldsymbol{\lambda_A}$, $\boldsymbol{\lambda_B}$ and $\boldsymbol{\lambda_C}$ are their respective normalised vectors. The angles between these vectors are a, b and c. Therefore using equations (5.21) and (5.22)

$$\boldsymbol{\lambda_A}\cdot\boldsymbol{n} = \boldsymbol{\lambda_A}\cdot(\boldsymbol{R_B}\times\boldsymbol{R_C}) \tag{5.24}$$

Now consider a dislocation loop ABC being projected away from $\boldsymbol{X}$ onto an arbitrary plane, Figure 5.4, to form a new triangle $A'B'C'$, such that $\boldsymbol{A'} = \boldsymbol{X} + m_A\boldsymbol{R_A}$, $\boldsymbol{B'} = \boldsymbol{X} + m_B\boldsymbol{R_B}$, $\boldsymbol{C'} = \boldsymbol{X} + m_C\boldsymbol{R_C}$, where m_A, m_B and m_C are positive scalars:

$$\boldsymbol{R_{A'}} = \boldsymbol{A'} - \boldsymbol{X} = \boldsymbol{X} + m_A\boldsymbol{R_A} - \boldsymbol{X} = m_A\boldsymbol{R_A} \tag{5.25}$$

$$\boldsymbol{\lambda_{A'}} = \frac{\boldsymbol{R_{A'}}}{|\boldsymbol{R_{A'}}|} = \frac{m_A\boldsymbol{R_A}}{|m_A\boldsymbol{R_A}|} = \operatorname{sgn}(m_A)\frac{\boldsymbol{R_A}}{|\boldsymbol{R_A}|} = \operatorname{sgn}(m_A)\boldsymbol{\lambda_A} = \boldsymbol{\lambda_A} \tag{5.26}$$

$$\boldsymbol{n'} = m_Am_B\,(\boldsymbol{R_A}\times\boldsymbol{R_B}) + m_Bm_C\,(\boldsymbol{R_B}\times\boldsymbol{R_C}) + m_Cm_A\,(\boldsymbol{R_C}\times\boldsymbol{R_A}) \tag{5.27}$$

Therefore using equations (5.26) and (5.27)

$$\boldsymbol{\lambda}_{\mathbf{A}'} \cdot \boldsymbol{n}' = \boldsymbol{\lambda}_{\mathbf{A}} \cdot \boldsymbol{n}' = m_B m_C \boldsymbol{\lambda}_{\mathbf{A}} \cdot (\boldsymbol{R}_{\mathbf{B}} \times \boldsymbol{R}_{\mathbf{C}}) = m_B m_C \boldsymbol{\lambda}_{\mathbf{A}} \cdot \boldsymbol{n} \tag{5.28}$$

$$\therefore \operatorname{sgn}(\boldsymbol{\lambda}_{\mathbf{A}} \cdot \boldsymbol{n}) = \operatorname{sgn}(\boldsymbol{\lambda}_{\mathbf{A}'} \cdot \boldsymbol{n}') \tag{5.29}$$

$$\therefore \Omega_{A'B'C'} = \Omega_{ABC} \tag{5.30}$$

Where (5.30) follows from (5.29) and (5.2.1) and the fact the the angles a, b, c and s have not changed during the projection. It can also also be seen that the sign of the solid angle changes when the sense of the loop changes since $\boldsymbol{\lambda}_{\mathbf{A}} \cdot (\boldsymbol{R}_{\mathbf{C}} \times \boldsymbol{R}_{\mathbf{B}}) = -\boldsymbol{\lambda}_{\mathbf{A}} \cdot (\boldsymbol{R}_{\mathbf{B}} \times \boldsymbol{R}_{\mathbf{C}})$, therefore

$$\Omega_{ACB} = \Omega_{CBA} = \Omega_{BAC} = -\Omega_{ABC} = -\Omega_{CAB} = -\Omega_{BCA} \tag{5.31}$$

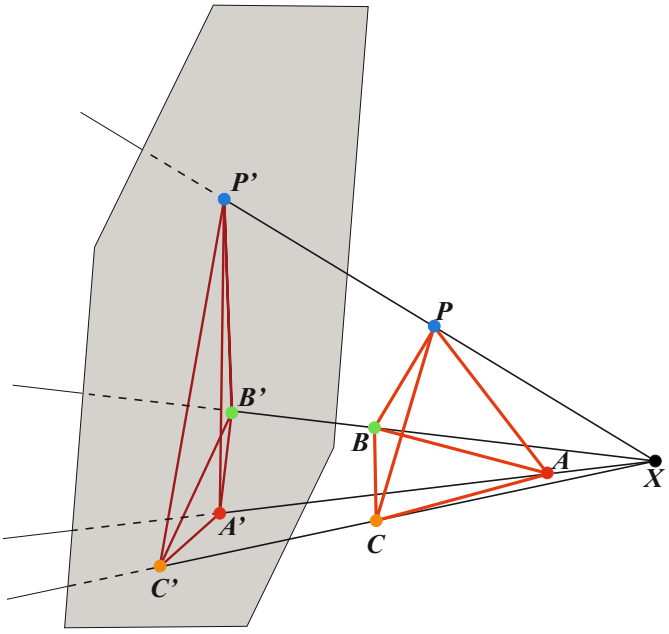


Figure 5.4: A projection of tetrahedron $ABCP$ onto a plane from field point $\boldsymbol{X}$. The $\boldsymbol{R}$ vectors go from $\boldsymbol{X}$ to their respective nodes. It can be seen that there are an infinite number of parallel planes onto which this projection can be made since the only requirement is that the plane intercept the four $\boldsymbol{R}$ vectors while not being inbetween the field point $\boldsymbol{X}$ and the tetrahedron.

If ABC is connected to closure point $\boldsymbol{P}$, we can form three new triangles: ABP, BCP and CAP which form a tetrahedron $ABCP$. For a given field point $\boldsymbol{X}$, we can choose a plane on which to project this tetrahedron as shown in Figure 5.4. If we define the shortest vector from $\boldsymbol{X}$ to the plane as $\boldsymbol{N}$, if $\boldsymbol{\lambda} \cdot \boldsymbol{N} = 0$ then $\boldsymbol{\lambda}$ is parallel to the plane, and if $\boldsymbol{\lambda} \cdot \boldsymbol{N} < 0$ then $\boldsymbol{\lambda}$ is pointing away from the plane,

in either case $\boldsymbol{\lambda}$ will not intercept the plane. Therefore provided:

$$\boldsymbol{\lambda_A} \cdot \boldsymbol{N} > 0 \quad \text{and} \quad \boldsymbol{\lambda_B} \cdot \boldsymbol{N} > 0 \quad \text{and} \quad \boldsymbol{\lambda_C} \cdot \boldsymbol{N} > 0 \quad \text{and} \quad \boldsymbol{\lambda_P} \cdot \boldsymbol{N} > 0 \tag{5.32}$$

m_A, m_B, m_C and m_P are all positive. The solid angle calculation can now be done in plane for triangles $A'B'P'$, $B'C'P'$ and $C'A'P'$. Fivel and Depres demonstrated [67] that, so long as the sense of the loop is maintained, breaking an arbitrary planar loop up into triangles in plane using a common closure point and summing their respective solid angles yields the equivalent solid angle as the original loop. Therefore:

$$\Omega_{A'B'P'} + \Omega_{B'C'P'} + \Omega_{C'A'P'} = \Omega_{A'B'C'} \tag{5.33}$$

However, from equation (5.30), the solid angle of a triangular loop is equal to the solid angle of one face of a tetrahedron. This argument can be extended to all four faces such that:

$$\Omega_{ABC} = \Omega_{A'B'C'}, \quad \Omega_{A'B'P'} = \Omega_{ABP}, \quad \Omega_{B'C'P'} = \Omega_{BCP}, \quad \Omega_{C'A'P'} = \Omega_{CAP}. \tag{5.34}$$

Therefore:

$$\Omega_{ABC} = \Omega_{ABP} + \Omega_{BCP} + \Omega_{CAP}. \tag{5.35}$$

It is always possible to select a plane for which condition (5.32) is satisfied provided $\boldsymbol{X}$ is outside the tetrahedron formed by $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{P}$. If $\boldsymbol{X}$ is inside this tetrahedron then no plane selected can satisfy condition (5.32) and so using this method calculates the wrong solid angle. This demonstrates that if the closure point $\boldsymbol{P}$ is chosen to be anywhere inside a finite volume V then $\boldsymbol{u}$ arising from a triangular loop ABC can be evaluated on the surface of V provided ABC is inside V.

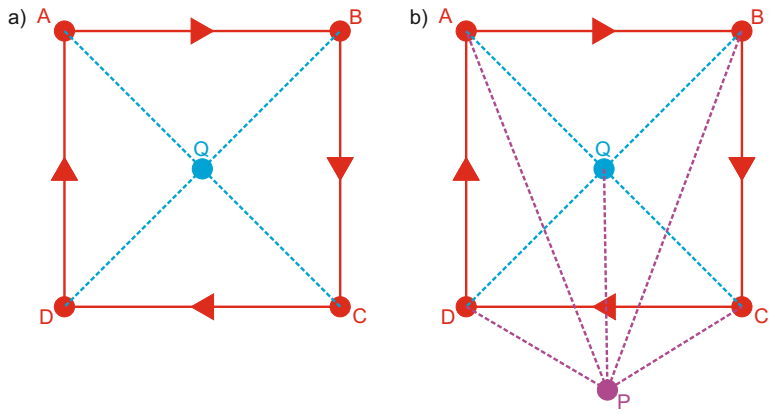


Figure 5.5: a) Square loop $ABCD$ can be divided into four triangles using $\boldsymbol{Q}$, ABQ BCQ CDQ and DAQ, for which each of the displacements can be calculated to give the total displacement of the whole loop when summed. b) The loop can further be divided using out of plane closure point $\boldsymbol{P}$, which renders the use of $\boldsymbol{Q}$ redundant. This approach can be used for any polygonal loop.

This method can be extended beyond triangular loops. Consider a square planar loop $ABCD$ broken up into four triangles in the habit plane of the loop using point $\boldsymbol{Q}$ as shown in Figure 5.5 a). The displacement of these triangles are then calculated using an out of plane arbitrary closure point $\boldsymbol{P}$, shown in 5.5 b). This generates the following $\boldsymbol{F}$ terms:

$$\begin{aligned}&\boldsymbol{F_{AB}} + \boldsymbol{F_{BQ}} + \boldsymbol{F_{QA}} + \boldsymbol{F_{BC}} + \boldsymbol{F_{CQ}} + \boldsymbol{F_{QB}} + \boldsymbol{F_{CD}} + \boldsymbol{F_{DQ}} + \boldsymbol{F_{QC}} + \boldsymbol{F_{DA}} + \boldsymbol{F_{AQ}} + \boldsymbol{F_{QD}} \\ &= \boldsymbol{F_{AB}} + \boldsymbol{F_{BP}} + \boldsymbol{F_{PA}} + \boldsymbol{F_{BC}} + \boldsymbol{F_{CP}} + \boldsymbol{F_{PB}} + \boldsymbol{F_{CD}} + \boldsymbol{F_{DP}} + \boldsymbol{F_{PC}} + \boldsymbol{F_{DA}} + \boldsymbol{F_{AP}} + \boldsymbol{F_{PD}} \\ &= \boldsymbol{F_{AB}} + \boldsymbol{F_{BC}} + \boldsymbol{F_{CD}} + \boldsymbol{F_{DA}} \end{aligned} \tag{5.36}$$

$$\begin{aligned}&\Omega_{ABCD} = \Omega_{ABQ} + \Omega_{BCQ} + \Omega_{CDQ} + \Omega_{DAQ} \\ &= \Omega_{ABP} + \Omega_{AQP} + \Omega_{QBP} + \Omega_{BCP} + \Omega_{BQP} + \Omega_{QCP} \\ &+\Omega_{CDP} + \Omega_{CQP} + \Omega_{QDP} + \Omega_{DAP} + \Omega_{DQP} + \Omega_{QAP} \end{aligned} \tag{5.37}$$

Changing the sense of the loop changes the sign of the solid angle, equation (5.31). So eight terms in equation (5.2.1) cancel to give:

$$\Omega_{ABCD} = \Omega_{ABP} + \Omega_{BCP} + \Omega_{CDP} + \Omega_{DAP} \tag{5.38}$$

This demonstrates that breaking the loop up in-plane using $\boldsymbol{Q}$ is unnecessary and $\boldsymbol{P}$ can be directly used to calculate the Ω_{ABCD} term. This argument can also be extending for a many sided planar polygonal loop.

5.2.2 Colliding and Non-planar loops

Using a single closure point has the dual advantage of working for both non-planar loops and loops which have collided, even if they have differing Burgers vectors. This is because:

$$\tilde{\boldsymbol{u}}(X, \boldsymbol{b}_3) = \tilde{\boldsymbol{u}}(X, \boldsymbol{b}_1) + \tilde{\boldsymbol{u}}(X, \boldsymbol{b}_2) \tag{5.39}$$

where $\boldsymbol{b}_3 = \boldsymbol{b}_1 + \boldsymbol{b}_2$ and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are two different Burgers vectors. We can demonstrate this fairly easily, using a simple example.

Some identities are required to better understand this example. We must first acknowledge that, since $\boldsymbol{b}$ only appears in the prefactor of $\boldsymbol{f}_{AB}$ and $\boldsymbol{g}_{AB}$, reversing the Burgers vector also reverses the resultant displacement vector. Therefore, simultaneously reversing the line direction and Burgers vector has no effect on the resultant displacement vector, as expected. The second identity required is that two dislocation loops occupying the same space but with different Burgers vectors, $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$, is equivalent to one loop with a combined burgers vector of $\boldsymbol{b_3} = \boldsymbol{b_1} + \boldsymbol{b_2}$. Again this follows as $\boldsymbol{b}$ appears in the prefactor of every term in equation (5.1).

Now we can consider our example: two colliding triangular loops, ABC and DBC. They are on separate (arbitrary) planes. The solid angle terms are calculated using arbitrary closure point $\boldsymbol{P}$. If we define

$$\omega_{ABC} = -\frac{\Omega_{ABC}}{4\pi} \tag{5.40}$$

and consider two separate loops just prior to the formation of a junction, as shown in Figure 5.6a, then for any field point outside of the polyhedron formed by $ABCDP$, the displacement is:

$$\begin{aligned}
\boldsymbol{u}(\boldsymbol{X}) &= \boldsymbol{b_1}\omega_{ABC} + \boldsymbol{b_2}\omega_{DBC} \\
&+\boldsymbol{F_{AB}}(\boldsymbol{b_1}) + \boldsymbol{F_{BC}}(\boldsymbol{b_1}) + \boldsymbol{F_{CA}}(\boldsymbol{b_1}) + \boldsymbol{F_{DB}}(\boldsymbol{b_2}) + \boldsymbol{F_{BC}}(\boldsymbol{b_2}) + \boldsymbol{F_{CD}}(\boldsymbol{b_2}) \qquad (5.41) \\
&= \boldsymbol{b_1}\omega_{ABP} + \boldsymbol{b_1}\omega_{BCP} + \boldsymbol{b_1}\omega_{CAP} + \boldsymbol{b_2}\omega_{DBP} + \boldsymbol{b_2}\omega_{BCP} + \boldsymbol{b_2}\omega_{CDP} \\
&+\boldsymbol{F_{AB}}(\boldsymbol{b_1}) + \boldsymbol{F_{BC}}(\boldsymbol{b_1}) + \boldsymbol{F_{CA}}(\boldsymbol{b_1}) + \boldsymbol{F_{DB}}(\boldsymbol{b_2}) + \boldsymbol{F_{BC}}(\boldsymbol{b_2}) + \boldsymbol{F_{CD}}(\boldsymbol{b_2}) \qquad (5.42) \\
&= \boldsymbol{b_1}\omega_{ABP} + \boldsymbol{b_1}\omega_{CAP} + \boldsymbol{b_2}\omega_{DBP} + \boldsymbol{b_2}\omega_{CDP} + \boldsymbol{b_3}\omega_{BCP} \\
&+\boldsymbol{F_{AB}}(\boldsymbol{b_1}) + \boldsymbol{F_{CA}}(\boldsymbol{b_1}) + \boldsymbol{F_{DB}}(\boldsymbol{b_2}) + \boldsymbol{F_{CD}}(\boldsymbol{b_2}) + \boldsymbol{F_{BC}}(\boldsymbol{b_3}) \qquad (5.43)
\end{aligned}$$

It can now be seen that the displacement vector generated by the two loops treated separately is identical to when they are treated as the junction shown in Figure 5.6b.

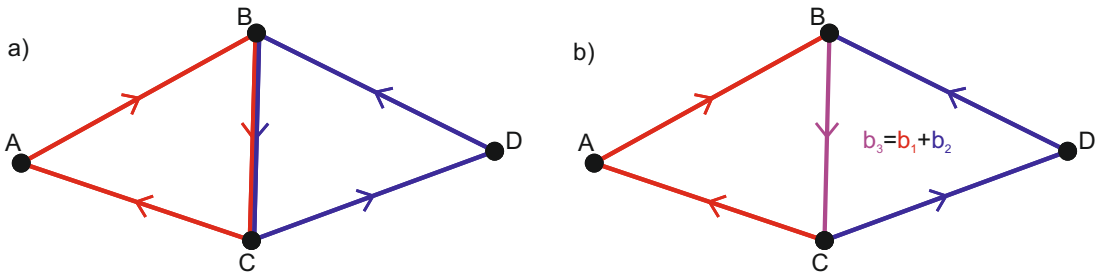


Figure 5.6: Two triangular loops, ABC and DBC, with Burgers vectors of $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ respectively, can be thought of as a) two separate loops or b) a junction when calculating the displacements they cause. The Burgers vector of the merged segment in b) is the sum of the two separate segments in a), $\boldsymbol{b_3}$.

Conservation of Burgers vectors means that the sum of the Burgers vectors entering a node in a dislocation network must equal the sum leaving. In the example above, $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ enter and $\boldsymbol{b_3}$ leaves node B. Thus for the conservation of Burgers vector we must have $\boldsymbol{b_1} + \boldsymbol{b_2} = \boldsymbol{b_3}$. Similarly at point C, BC splits into CA and CD resulting in the same sum. This is consistent with how $\boldsymbol{b_3}$ is calculated when the junction is formed. However, if we change the line direction of one of the colliding loops, the sum changes. Loop ABC colliding with loop DCB

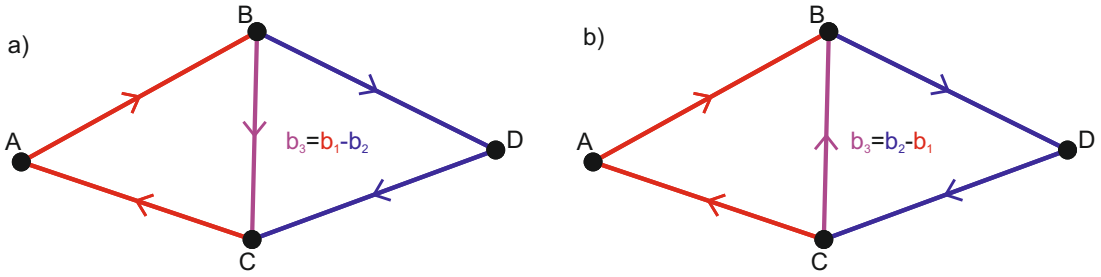


Figure 5.7: Two loops, ABC and DCB, have collided. There are two possible junctions that can form, a) or b). Conservation of Burgers vector means that BC in a) has both opposite line direction and burgers vector to CB in b). This results in the same displacement being calculated for both cases.

instead of DBC, as shown in Figure 5.7. If the junction forms such that at point B line AB enters and lines BD and BC leave, as shown in Figure 5.7a, then the Burgers vector sum is $\boldsymbol{b_1} = \boldsymbol{b_2} + \boldsymbol{b_3}$. Nodal based DD codes such as *ParaDiS* and *DDLab* have been written to account for this conservation of Burgers vector. This is convenient when the displacement is calculated because if two segments have opposite line directions and Burgers vectors then they give identical displacement contributions when using a common closure point. This means that segment BC with a Burgers vector of $-\boldsymbol{b_2}$ is indistinguishable from segment CB with a Burgers vector of $\boldsymbol{b_2}$. This means that this example combines in the same way as the example in Figure 5.6 to give the correct displacement. By extension, the same

can be said of Figure 5.7b where the junction is formed with the other possible configuration. Additionally, this can be used as a proof for the case for a non-planar loop, which can be thought of as a special condition of the collided loop, where $\boldsymbol{b_1} = \boldsymbol{b_2}$ and $\boldsymbol{b_3} = [000]$.

5.2.3 Correctly Using the Method

It can be seen that the closure point now forms a pyramidal "tepee" shape with the dislocation loop as the base and the closure point as the "peak". For planar loops this is simply an irregular oblique pyramid. However this solution is not always correct, for while the elastic terms still cancel, there are situations in which the solid angle term is incorrectly calculated, as explained previously.

Whenever the field point at which the displacement is measured exists inside the polyhedron formed by the loop and the closure point, an exterior solid angle of incorrect sign is calculated instead of the desired result, i.e. $\Omega_{ABC} \neq \Omega_{ABP} + \Omega_{BCP} + \Omega_{CAP}$. If displacement is calculated piecewise by segment, it is not possible to ascertain whether or not a field point exists within such a volume, since there is insufficient information about the loop or loops to which the segment belongs. While this is true for a completely general case, for the surface of a finite volume this problem can be avoided. If a closure point is chosen so that it is inside the volume, all field points on the surface of the volume would be outside of any polyhedron formed by the internal dislocation network and closure point. Therefore $\tilde{\boldsymbol{u}}(X)$ can be correctly calculated for every possible position of X on the surface, provided that no dislocations exit the volume. Fortunately, to couple with FEM using the superposition method, $\tilde{\boldsymbol{u}}$ need only be evaluated on the surface of the simulated volume. Dislocation segments are prevented from exiting through S_u, as this is unphysical and would produce an erroneous $\tilde{\boldsymbol{\sigma}}$ stress to remove the slip step. However dislocations can exit the volume through any part of the surface where tractions are specified, S_t. Therefore we now need to extend the method to handle exited segments because even if a slip step doesn't form on S_u, there can still be a contribution to the displacement field caused by slip step formation.

Before this, it should be noted that Barnett's solution gives the displacement of a loop that has swept out its current area. If we think of a real sample that has been cut and/or polished for testing, there is no displacement on the surface caused by the internal dislocation structure. At the start of a simulation before any load is applied, there should be no displacement on S_u. Subsequent displacements should only be caused by changes in the initial dislocation network that occur during the simulation. Therefore, when calculating $\tilde{\boldsymbol{u}}$ at time t, we must apply the displacement calculation to the dislocation network as it currently is and then

subtract the result of the calculation for the initial dislocation structure such that $\tilde{\boldsymbol{u}}(t) = \tilde{\boldsymbol{u}} - \tilde{\boldsymbol{u}}(0)$. For example consider an initial dislocation loop which shrinks and is annihilated. This process should produce a non-zero displacement, but there is no information in the current dislocation network about the initial loop.

5.3 Calculating Displacements for Exited Segments

Barnett's equation has been derived for a dislocation loop in an infinite isotropic solid. In order to calculate the displacement on the surface of the volume due to an exited dislocation segment, it is convenient to move the exited segment to an infinite distance from the volume whilst maintaining the connectivity of the loop. This issue is discussed in [20]. In a simulation, when a dislocation segment exits, it is moved to a position, that is very far (several orders of magnitude) from the surface relative to the size of the volume, designated as "pseudo-infinity". Extra virtual segments are then generated as necessary to connect the exited segment to the internal segments of the dislocation loop at the surface, as discussed in Chapter 4.

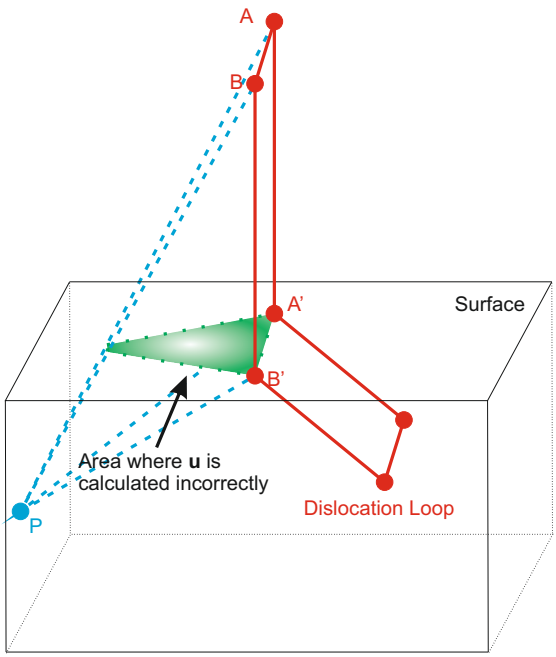


Figure 5.8: When calculating displacements using a common closure point, dislocation segments which have exited the volume have an area on the surface S_t associated with them within which displacements are calculated incorrectly.

Once an exited dislocation node has been projected, there will be areas on the surface where the result of the displacement calculation will be incorrect due

to the volume the dislocation loop creates with the closure point intersecting the surface of the simulated volume as shown in Figure 5.8. In order to avoid this problem, adjustments must be made to the method to give the correct result.

For external parts of a dislocation structure, because it is known that virtual nodes have been projected from the surface, more information about the geometry of the loop is available when looking at individual segments. This is because:

$$\boldsymbol{A} = \boldsymbol{A}' + L\boldsymbol{n} \tag{5.44}$$

where $\boldsymbol{A}$ is the position vector of a virtual node, $\boldsymbol{A}'$ is its corresponding surface node, equivalent to the sum of the surface intersection points of all segments connected to $\boldsymbol{A}$, $\boldsymbol{A}' = \sum \boldsymbol{Q}$, and L is some large factor, much greater than the largest dimension of the simulated volume, approximating infinity and $\boldsymbol{n}$ is the surface normal, as illustrated in Figure 5.8. Since the segments are not inside the material, the elastic displacements caused by the presence of the line of the dislocation ($\boldsymbol{f}_{\mathbf{AB}}$ and $\boldsymbol{g}_{\mathbf{AB}}$) are zero and so can be ignored. This just leaves the solid angle term, Ω, for the displacement arising from the plastic slip step that needs to be calculated. This must be done in such a way that the contribution to Ω from the virtual segments gives the correct value when added to the contributions from the internal segments so that the total Ω for the entire network is the solid angle expected at any given field point on the surface of the simulated volume. To achieve this, the internal dislocation network must be closed at the surface by a virtual surface segment for which only the Ω term is calculated using the internal common closure point $\boldsymbol{P}$. The external part of the loop then has a unique closure point assigned to it and the correct Ω contribution can then be calculated as shown in Figure 5.9a. It should be noted that closure points for external parts of the loop do not have to be unique, they merely must not form a volume that intersects the surface. Assigning a common closure point to each face of the surface a dislocation segment could exit from would also work. However, when it comes to implementing in a DDD code, it was found to be quicker and easier to find a unique closure point for each exited segment than to find the face of the surface a segment has been projected from. Another advantage of assigning a unique closure point to each exited segment is that the algorithm is completely general and independent of the geometry being simulated.

The virtual parts of a dislocation network are not always simple and may consist of multiple segments at pseudo-infinity. Solid angle contributions for each of these virtual segments must be calculated. In order to do this, a sub loop must be found for all segments at pseudo-infinity. All Segments (along the surface normal) joining a virtual segment to the surface can be ignored since their contribution

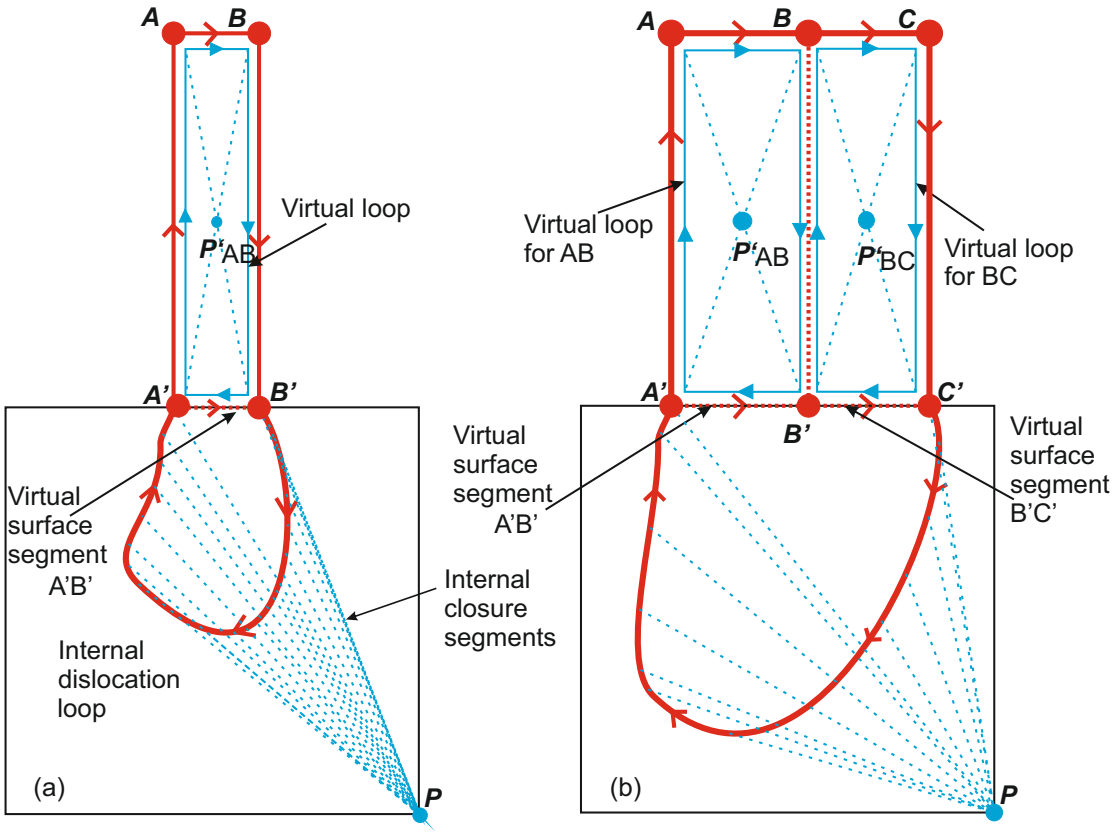


Figure 5.9: (a) One segment AB has exited the surface. Virtual segments are handled separately from the internal segments for the calculation of displacements by introducing a virtual surface segment connecting $\boldsymbol{A}'$ to $\boldsymbol{B}'$ to close the internal loop. Virtual surface segments use the common closure point $\boldsymbol{P}$ and like all virtual segments only contribute to Ω. Each virtual segment at pseudo infinity, e.g. $\boldsymbol{AB}$, has it's own closure point, e.g. $\boldsymbol{P}'_{AB}$. (b) At a later time a second segment BC has exited which has closure point $\boldsymbol{P}'_{BC}$. Every segment at pseudo-infinity has an associate rectangular loop assigned to it. Maintaining the sense of the loop direction ensures that the solid angles combine correctly.

to the displacement will be included in that of the sub-loop associated with the virtual segment (at pseudo infinity) they connect to. Virtual nodes which are not attached to a virtual segment connected to the surface must be projected back on the the surface to find the corresponding corners of the sub-loop. An appropriate virtual surface segment must also be created to ensure the closure of the internal part of the dislocation network as shown in Figure 5.9b.

In the event that a junction exits the surface and there is a node at pseudo-infinity connecting to three other nodes at pseudo-infinity, this approach still functions correctly. The junction node represents the intersection of three sub-loops which will combine to give the correct total displacement provided the Burgers vectors and line direction are maintained. Therefore conservation of Burgers vector ensures the correct handling of any junctions present in the virtual network.

This treatment of exited segments is similar to the technique utilised by Hus-

sein and El-Awady to simulate surface roughness evolution [65]. The key difference is that their technique for analysing surface roughness only takes exited segments into account and ignores the elastic displacement contributions from the internal dislocation segments. This results in a marginal underestimation of the height of the slip step for partially exited loops. Despite this, the technique is a good approximation, since the contribution to the height of a slip step by the internal structure is small in comparison to that of the exited segments. Additionally, the position and orientation of the slip steps is accurate. However, as dislocation density rises, the inaccuracies in $\tilde{\boldsymbol{u}}$ will increase. The method proposed here would provide better results in such a scenario as the elastic displacement from every segment is calculated.

5.4 Implementing the Algorithm

The differences in the displacement fields of real and virtual segments must be properly reflected in the code. The complete algorithm for both internal and exited segments is shown below.

Result: Dislocation displacement $\tilde{\boldsymbol{u}}$ on surface FE nodes
for $\forall$ *FE nodes* $\in S$ **do**
  for $i = 1$ **to** N **do** // $\forall$ dislocation segments
    $A = \boldsymbol{A}(i)$// nodal coordinates of segment
    $B = \boldsymbol{B}(i)$
    if $A \in V \wedge B \in V$ **then** // an internal segment
      $\Omega = \Omega +$ Omega(A, B, P) // Ω_{ABP} using common closure point
      $f = f +$ f(A, B) // $\boldsymbol{f_{AB}}$
      $g = g +$ g(A, B) // $\boldsymbol{g_{AB}}$
    else
      if $A \notin S \wedge B \notin S$ **then** // segment is not connected to surface
        $A' =$ projection(A,S) // project segment onto surface
        $B' =$ projection(B,S)
        $P' =$ centre(A,B,B',A') // unique closure point
        $\Omega = \Omega +$ Omega$(A, B, P') +$ Omega$(B, B', P') +$ Omega$(B', A', P') +$ Omega(A', A, P') // $\Omega_{ABB'A'}$ of the rectangular loop using its closure point $\boldsymbol{P'}$
        $\Omega = \Omega +$ Omega(A', B', P) // calculate $\Omega_{A'B'P}$ for the segment projected onto the surface using $\boldsymbol{P}$
      else
        // virtual segments (parallel to $\boldsymbol{n}$) connecting a virtual node at pseudo infinity to a surface node
        // skip segment. Ignore to avoid double counting.
      end
    end
  end
  $u = \Omega + c_1 f + c_2 g$ // c_1 and c_2 are constants
  $\tilde{u}(t) = u - u(0)$ // subtract off $\tilde{\boldsymbol{u}}_0$ calculated at $t = 0$
end

Algorithm 3: Pseudocode to find the displacement on the surface of a simulated volume. S is S_u during a simulation or $S_u \wedge S_t$ during post processing. The algorithm is amenable to parallelisation over dislocation segments or field points.

Now that we have implemented the theory into a functional method to measure displacements on the surface of a volume, we must demonstrate that it works as expected. Several tests of this method for calculating surface displacements were carried out to verify the method. Firstly, the displacement caused by a triangular loop was calculated using Barnett's solution on the surface of a cube containing the loop. This was compared to the results for when the displacement was calculated using several arbitrary closure points inside the cube. All of the surface displacements calculated were identical, as shown in Figure 5.10.

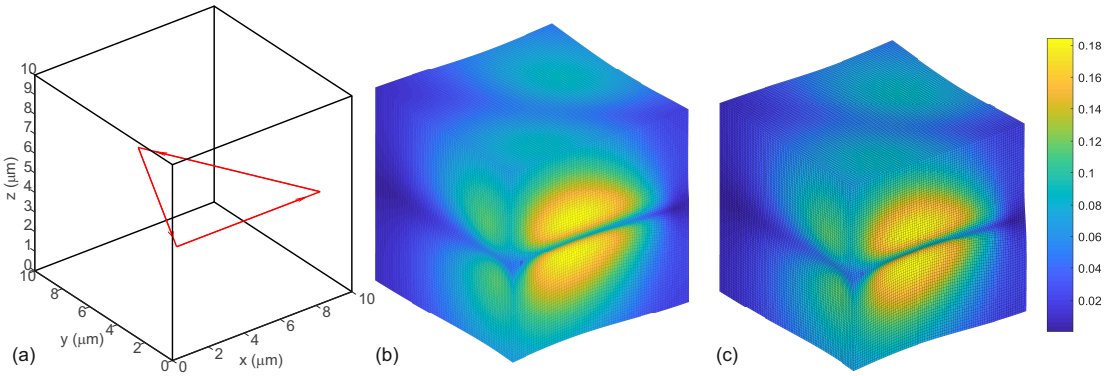


Figure 5.10: A triangular dislocation loop on the (001) plane with a [110] Burgers vector (a) has its displacement field calculated on the surface of a cube using both Barnett's method (b) and using an arbitrary closure point (c). The results are indistinguishable so long as the closure point is inside the cube.

This test was performed with the closure point in various positions inside the cube, including in the plane of the loop and inside the loop itself. This demonstrates that it gives results equivalent with all other displacement codes currently available, since they all use some variation on the implementation of the Barnett solution. further comparisons could be made, although this would be comparing the codes themselves rather than the methodology implemented in them.

The displacement generated by two loops just prior to collision was compared with the displacement after they had collided and formed a junction. There was a slight change in displacement of $0.002b$ on the top surface, consistent with the displacement caused by the movement of the loops. No additional change was observed caused by the formation of the junction, as shown in Figure 5.11.

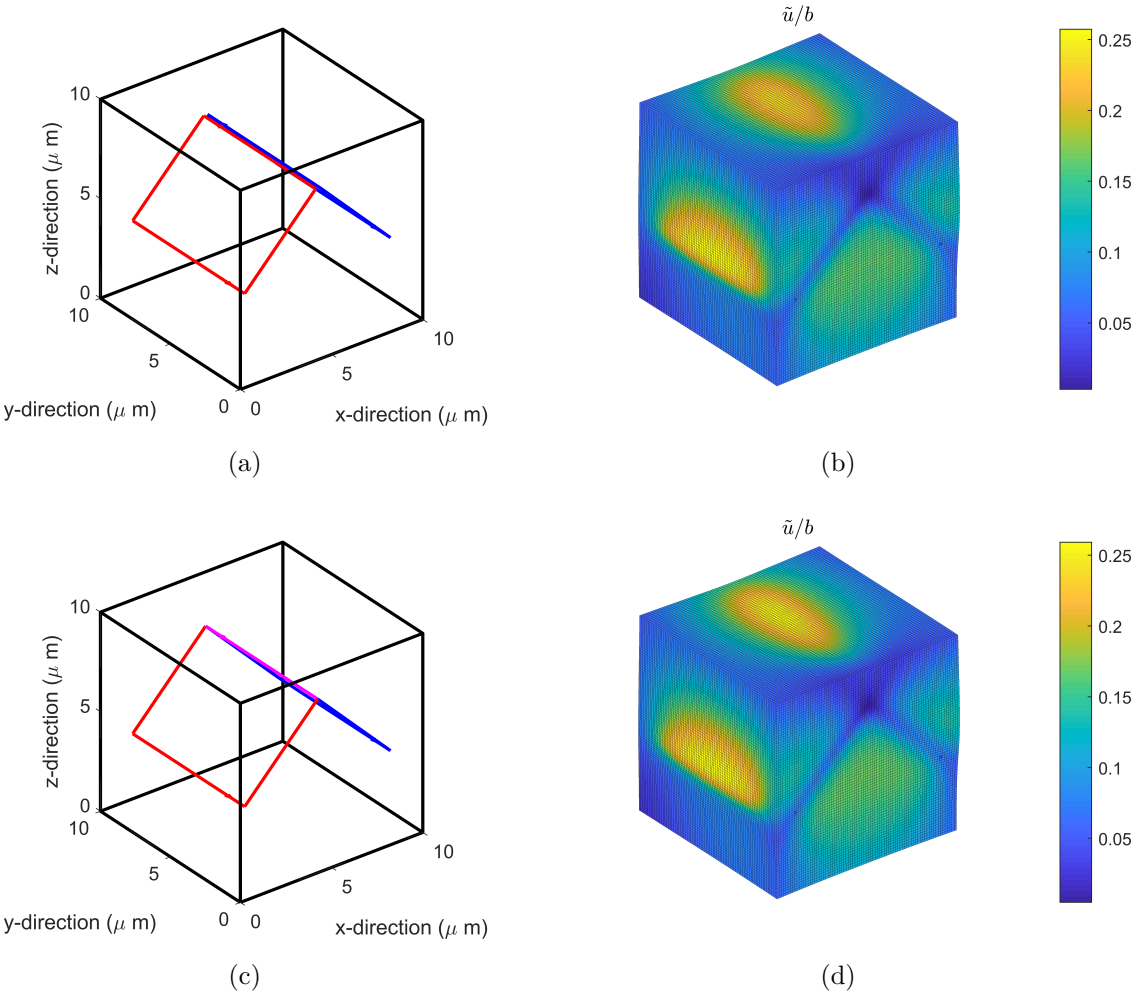


Figure 5.11: A cube aligned along $\langle 100 \rangle$ containing two square dislocation loops just prior to collision shown in (a) and the displacement they generate on the surface of the cube (b). After the the formation of a junction (c), the surface displacement is unchanged (d). The red loop is on the $(\bar{1}01)$ plane with a $\frac{a}{2}[111]$ Burgers vector while the blue loop is on the (101) plane with a $\frac{a}{2}[\bar{1}\bar{1}1]$ Burgers vector. The purple segment at the junction has a $a[001]$ Burgers vector.

The displacement on the surface due to a pair of square prismatic loops where the interstitial loop has exited along its glide prism and the vacancy loop remains inside V is shown in Figure 5.12. This agrees with the displacement obtained by Fivel and Depres using their method [67]

The algorithm described above was used to simulate an end loaded single crystal body-centered cubic micro cantilever beam using *EasyDD*. The full code is available on the Tarleton group GitHub. Identical simulations were carried out, evaluating the dislocation displacement $\tilde{\boldsymbol{u}}$ on S_u to enforce the displacement boundary conditions equation (5.45) and neglecting the dislocation displacements so that the only coupling between DD and FEM is through the traction boundary conditions:

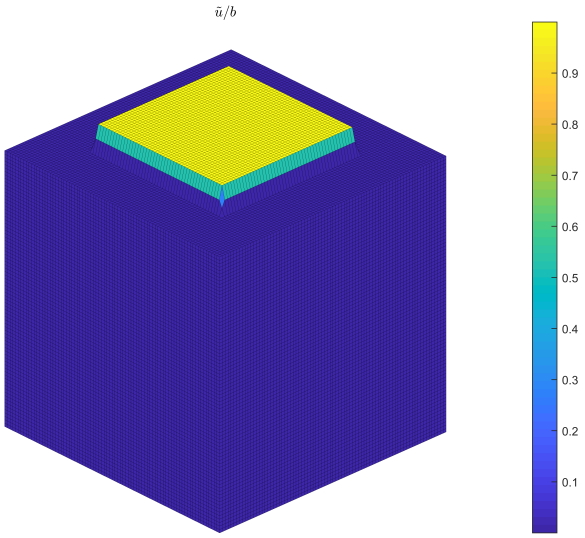


Figure 5.12: displacement on the surface of a cube due to a square (001) [001] prismatic interstitial loop which has exited out of the top of the cube.

$$\left.\begin{array}{ll} \boldsymbol{u} = \boldsymbol{U} & t > 0 \\ \boldsymbol{u} = \boldsymbol{0} & t = 0 \end{array}\right\} \text{ on } S_U \tag{5.45}$$

where U is an applied displacement, t is the simulation time and S_U is the part of the surface where displacement boundary conditions are applied. A BCC crystal structure was chosen with Poisson's ratio and bulk modulus approximating that of tungsten. 60 dislocation sources, 10 on each $\{110\}$ slip plane, were then scattered randomly throughout a single crystal cantilever with height and width of 5 and length of 18μm. The sources themselves must be closed loops rather than the commonly used isolated Frank-Reed source segments as this would produce erroneous $\tilde{\boldsymbol{u}}$ values due to Ω being ill defined. Physically this is correct as a dislocation segment can not terminate in a perfect lattice. The cantilevers were fixed at one end with a displacement of 0.22μm applied to the other end. It should be noted that the fixed end must not have any dislocations exiting through it which was achieved by pinning dislocations at this surface when they reached it. The resulting load-displacement curves are shown in Figure 5.13 and the dislocation networks that evolved are shown in Figure 5.14. There was an observable difference in both the load/displacement curves and final dislocation structures.

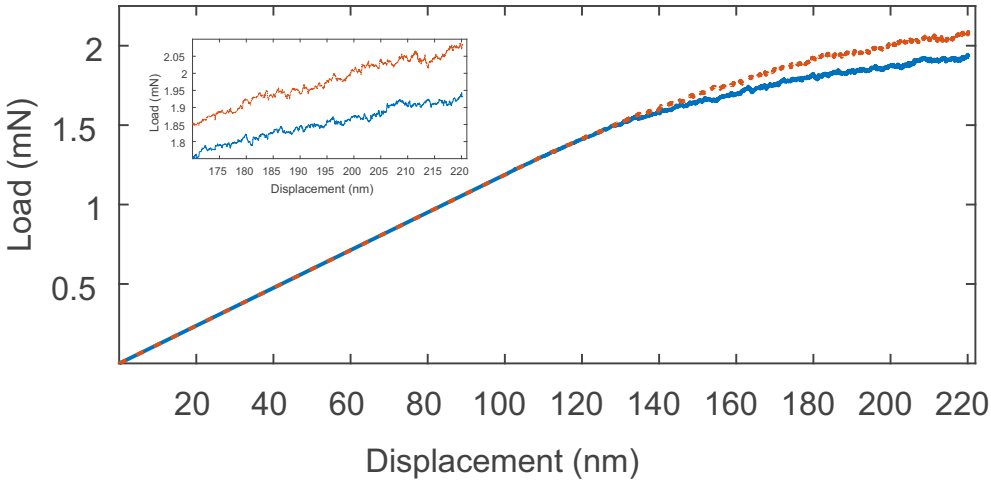


Figure 5.13: Simulated load displacement curve for two identical microcantilver simulations, with modified traction boundary conditions only, equivalent to $\tilde{\boldsymbol{u}} = 0$, (red dashed line) and with modified traction and displacement boundary conditions $\tilde{\boldsymbol{u}}$ (blue line). A greater plastic response is observed when displacement boundary conditions are included in the simulation.

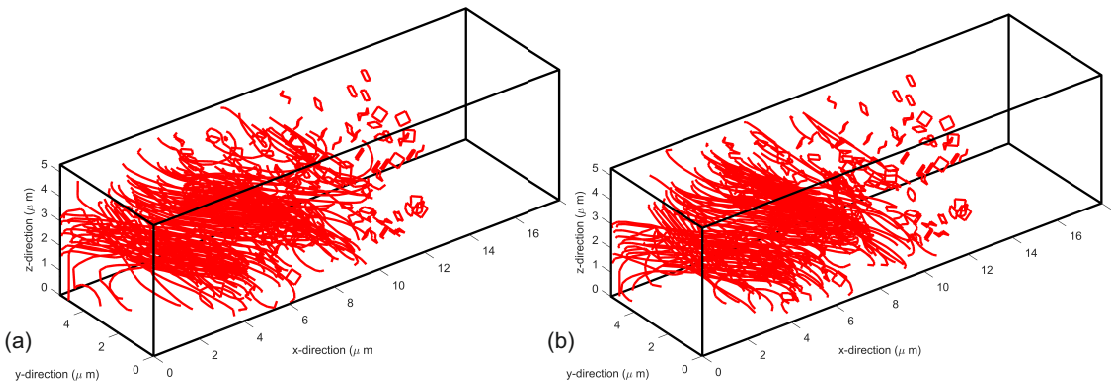


Figure 5.14: Dislocations network in single crystal beam bending simulations where (a) $\tilde{\boldsymbol{u}}$ is neglected and (b) $\tilde{\boldsymbol{u}}$ is used to modify the FEM boundary condition on S_u. A greater dislocation density and pile up at the neutral axis is observed in (a), despite the plastic response observed being smaller.

This method may also be used in post-processing to evaluate the displacement on the entire surface $S = S_u \wedge S_t$ of a volume. A rectangular cantilever, with width and height of 5 and length of 30μm was simulated which was fixed at one end with a displacement of 0.6μm applied at opposite end. The initial dislocation structure was 60 randomly distributed dislocation loops, 10 on each {110} slip plane, similar to the above case. After post processing, slip steps were observed on S_t where dislocations had exited the surface as can be seen in Figure 5.15. The complete final dislocation network consisted of 5736 segments totaling slightly more than 1mm in length (ignoring segments of pseudo-infinite length), of which

907, with a total line length of approximately 150μm, had exited. Both glide and cross-slip occurred.

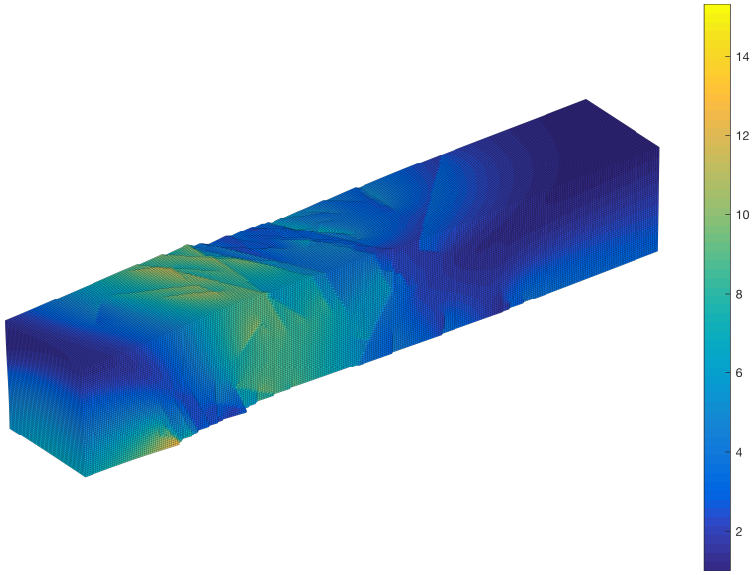


Figure 5.15: The dislocation displacement field evaluated on the surface of the simulated volume. Colour corresponds to the displacement magnitude. The fixed end is on the left of the image.

5.5 Summary

The implementations of this new method for measuring surface displacements has an all round positive effect on the quality of the data gained from a simulation, as well as providing additional information about changes to the surface of a material caused by dislocation motion. Additionally, it can be readily paralellised for improved performance when running larger scale simulations. Although it cannot provide accurate measurements of displacements caused internally, it is likely that this is the best picture of displacements that can be gained without recording additional dislocation history or loop configuration. With this in mind, we can now test the *EasyDD* code by running simulations of simple mechanical tests.

Chapter 6

Simulating a Tungsten Cantilever with *EasyDD*

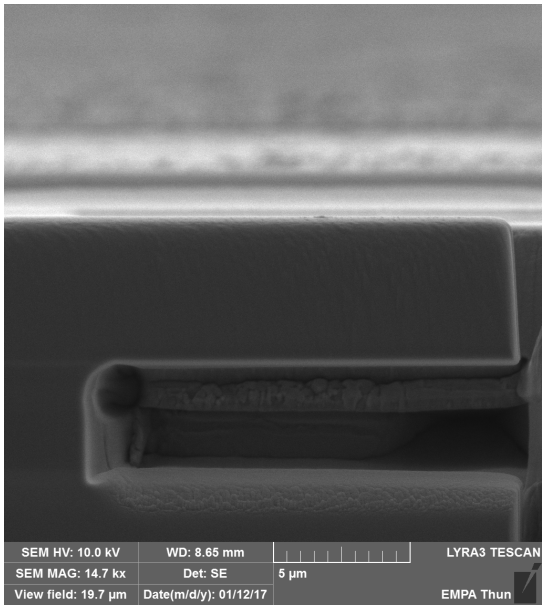
6.1 Introduction

Now that we have a robust model for simulating dislocation behaviour in a finite model, we can simulate a real experiment in order to test the quantitatively validity of the model. The bending of a pure tungsten micro-cantilever by a nanoindenter makes for a good test case for relatively simple performance. The aim is to simulate a real experiment and compare the data, similar to the work done by Tarleton et al. in 2015 [73], but in three dimensions instead of two.

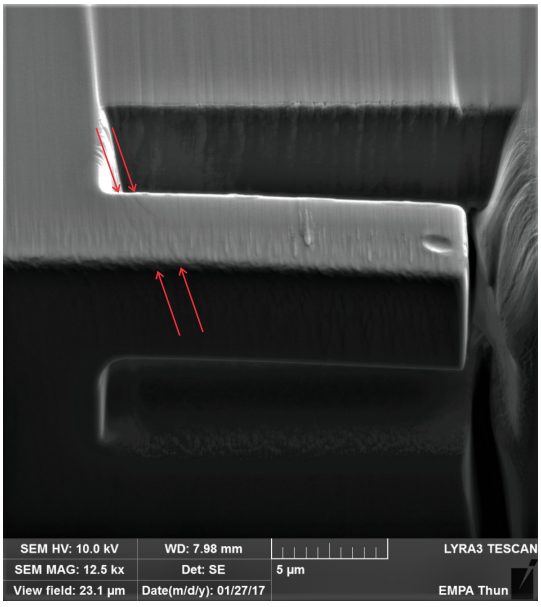
6.2 Simulation Setup

The experimental data used for comparison was provided by Johannes Ast of the Friedrich Alexander University of Erlangen Nuernberg. The original test was preformed on an unnotched, oblong cross-sectioned micro-cantilever 14.3 microns long, measured from the indenter tip to the bulk material, 5.3 microns in height and 5.4 microns in width. Force/displacement data as well as EBSD and SEM imaging of the sample were provided. The shear modulus, μ, of the material was estimated as 135 MPa from the force displacement data. Other material constants were taken from AZoM [74] giving the Poisson's ratio, ν, of 0.28 and the lattice parameter, a, of $318pm$. The cantilever was carved using a focused ion beam (FIB), along the crystallographic axes, so that the edges of the beam all lie in $\langle 100 \rangle$ type directions. The cantilever was loaded under displacement controlled conditions to a total vertical displacement of $2\mu m$ and then unloaded. The results are shown in Figure 6.1.

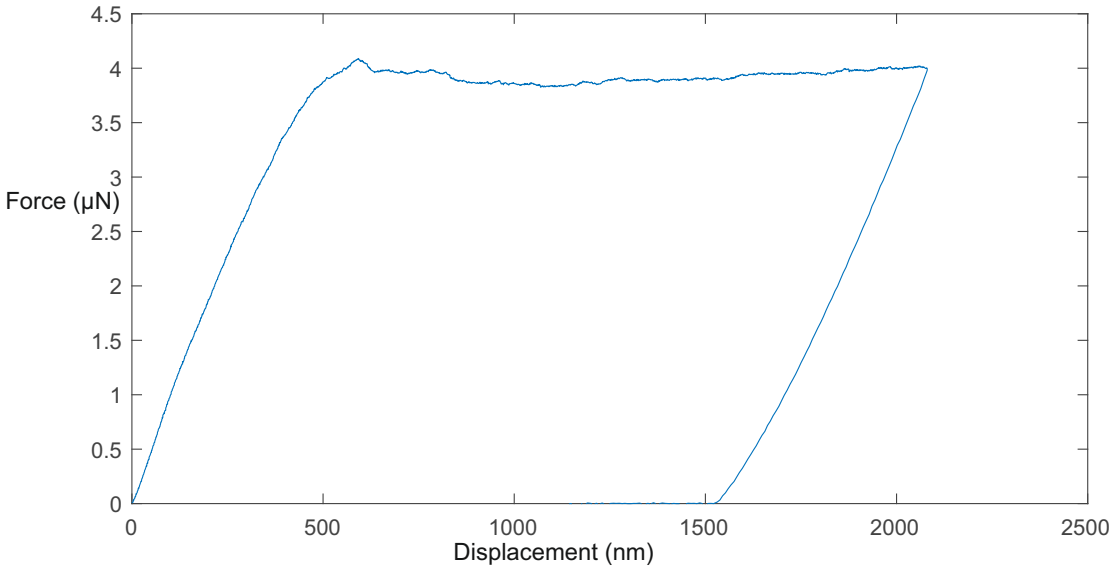
Using the quantitative details from the indenter experiment, it is possible to



(a)



(b)



(c)

Figure 6.1: A tungsten cantilever (a) Before loading, from the side. (b) After unloading, from above, with slip steps marked with arrows. (c)Force/displacement curve. Images and experimental data courtesy of J. Ast.

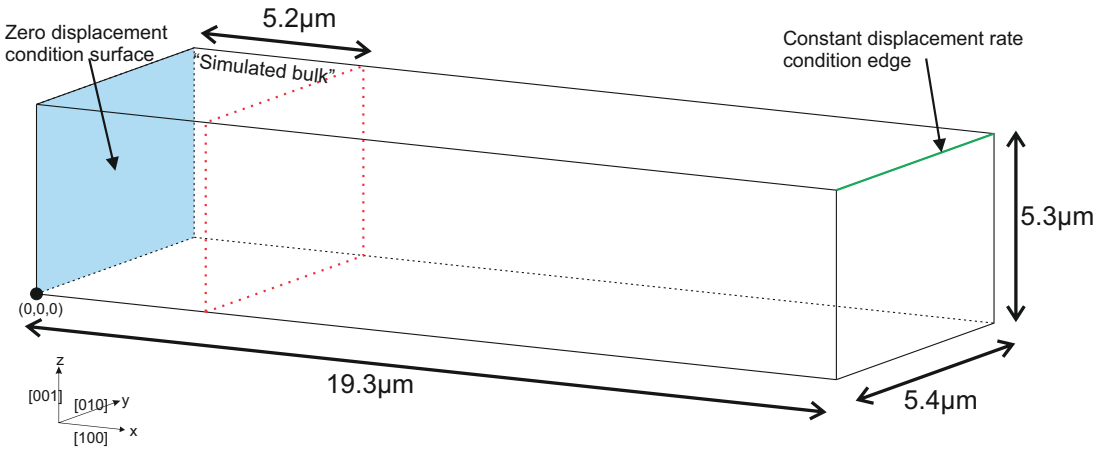


Figure 6.2: The setup of the FEM volume in which the dislocation behaviour is simulated.

construct a model with these details as input parameters. The basic physical properties can be input as quoted. The next important step is designing the cuboid volume used to simulate the cantilever. To do this correctly, the loading conditions must be considered. Thanks to the orientation in which the cantilever was cut in relation to the crystallography of the material, no rotation needs to be considered and the edges of the simulated volume can be aligned with the axes in space, as shown in Figure 6.2. The longest edges lie parallel to the x-axis, which is also in the [100] direction. The edges spanning width of the cantilever are parallel to the y-axis and the [010] direction. Finally, the vertical edges are parallel to the z-axis in the [001] direction. The origin, coordinates $(0, 0, 0)$, is positioned at the back-right-bottom vertex of the cuboid. The back face is where the cantilever would connect to the bulk material and is designated as the surface with a zero displacement condition, i.e. the fixed end. The top edge at the front face of the cantilever is where the constant displacement is applied. Together, this face and edge make up S_U. The edge combined with all other faces make up S_T, where a zero traction boundary condition is enforced. The volume is divided into elements, with 40 in the x direction and 8 in the y and z directions each, for a total of 2,560 elements. This is now a basic model capable simulating purely elastic behaviour. However, using the dimensions of the cantilever as quoted will not give the correct elastic response. This is because during a real mechanical test the fixed end is not completely immobile as the simulation assumes. There may be slight elastic deformation of the bulk of the material on the top surface, making the cantilever appear less stiff than it actually is. To compensate for this, an addition is made to the length of the cantilever of $5.2\mu m$. The first $5.2\mu m$ into the cantilever from the fixed end act as a sort of "simulated bulk", the size of which is calibrated so that the elastic response of the simulation matches that of the experiment.

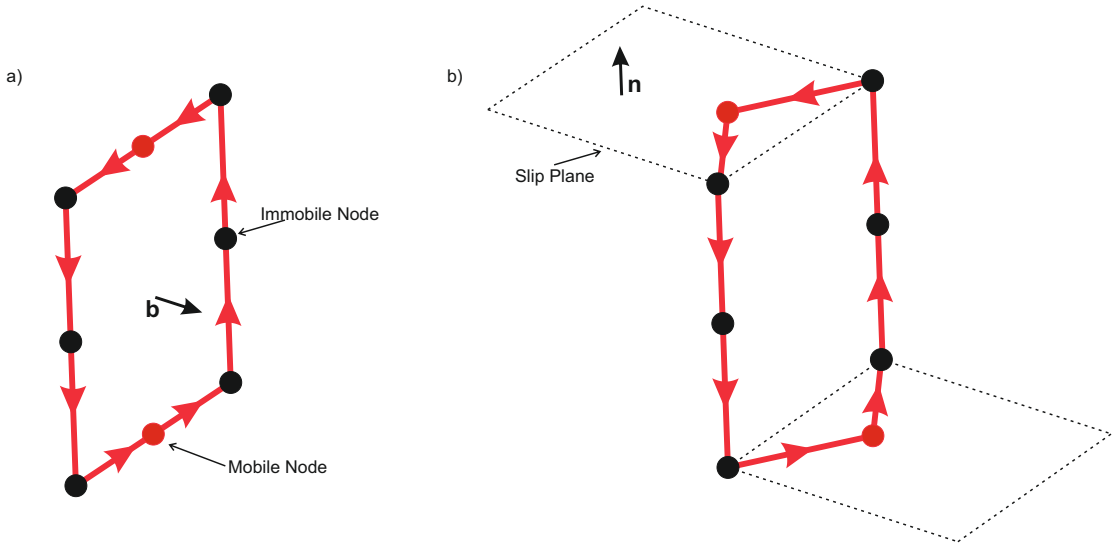


Figure 6.3: A square prismatic loop can be used as a dislocation source. (a) is the configuration at the start of the simulation and (b) is how the sources start to evolve once activated. The Burgers vector, $\boldsymbol{b}$, is parallel to the prismatic loop habit plane normal and the slip plane normal, $\boldsymbol{n}$, is parallel to the fixed segments.

Once the cantilever was set up as required, an initial dislocation structure had to be included. What is desired are sources of dislocations capable of continuously providing more dislocations as the simulation progresses. A good mechanism for producing this in *EasyDD* is the prismatic source. This is a square prismatic eight segments. Two sides are fixed in position whilst the other two are allowed to move freely, as shown in Figure 6.3. This essentially creates a pair of Frank-Read sources whilst maintaining complete connectivity of the structure so that there are no incomplete dislocation loops. Some of the algorithms in *EasyDD* are designed to work with complete loops, so prismatic sources are preferred over standard fixed end Frank-Read sources. The orientation and Burgers vector of the loop determine the slip system of the sources. The plane normal of the habit plane of the loop is parallel to the Burgers vector, whilst the fixed sides are parallel to the slip plane normal of the mobile sides. This allows for procedural generation of sources for any desired slip system. Additionally, the two sources on any loop will have opposite line directions. This means that at any given applied stress the sources will bow out in opposite directions. This is convenient because one direction may be moving into a location of higher stress in the volume than the other, allowing for faster initial growth of the dislocation loop. Therefor when using prismatic loop sources there is always one source expanding in the advantageous growth direction. It should be noted that the minimum dimension for the side of a source is the maximum length of a segment permitted in the simulation, l_{max}. If the source is any smaller than this, remeshing will impede the evolution of the source.

In order to improve simulation performance, it is important to strategically

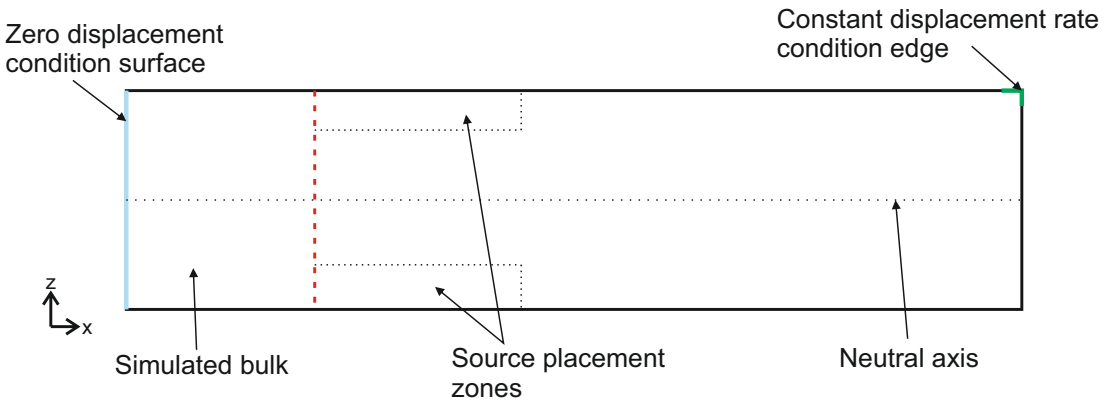


Figure 6.4: The positioning of the sources at the start of the simulation.

position dislocation sources. The convergence study discussed in Chapter 4 used a similar setup to that shown here. The dislocation sources were scattered throughout the volume but the majority of them were never activated. The stress in the cantilever is low far from the fixed end and there is a neutral axis on the x/y plane at the center of the cantilever. The stress is highest closer the the top and bottom surfaces and closer to the fixed end. For the duration of the simulation, only sources positioned here will activate, so there is little point adding more sources elsewhere since they will consume computing power without contributing meaningfully to the simulation, However, sources cannot be positioned in the simulated bulk portion of the cantilever. This is because it is the part of the simulated cantilever with the highest stress but the bulk in the experiment has a lower stress than in the cantilever and very quickly becomes negligible away from the cantilever. This means that any dislocation motion that occurs in the simulated bulk is not physical. Not placing sources in the bulk is a good way of reducing the presence of any such dislocations in the model. Further restricting the placement of sources to the top and bottom fifths of the cantilever gives a placement as shown in Figure 6.4.

The loops are scattered randomly throughout the placement zones by generating random numbers for the x, y and z coordinates of the center of each loop. If any loops intersect or are too close to each other then the starting configuration is regenerated. Sources are generated for each of the $\frac{a}{2}\langle 111\rangle\{110\}$ slip systems. There are 12 total slip systems each with 4 prismatic sources, 2 close to the top surface and 2 close to the bottom surface, for a total of 48 prismatic sources or 96 Frank-Read sources. The sources measure $100nm$ along each side. Not all of these slip systems will be activated by the applied stress, but it is assumed that sources present in a real material will be of random orientation, so sources are similarly generated so that any effects caused by local stress variation can be captured. Since tungsten is a BCC metal, the improved mobility law discussed in Chapter 3

is used to calculate dislocation velocities, while the displacement boundary condition corrections are calculated using the new method discussed in Chapter 5 and the traction boundary conditions are similarly solved using the method developed and implemented by D. Celis Garza, available on the Tarleton group GitHub. A convergence study was run using the same initial setup to find the equilibrium loading rate by monitoring the elastic displacement component, as mentioned in Chapter 4. The drag coefficient for edge dislocations, B_{edge}, was set to 1. This means that, even though no reliable value for the drag coefficients could be found in the literature, the conversion of the time units should be simple once a value for the drag coefficients is calculated. It should be noted that while the the cantilever in the experiment was loaded and unloaded, the current version of the *EasyDD* code takes too long to measure this amount of plastic behaviour and so the goal is only to model the initial stages of plastic deformation. Given this fact, it is also assumed that the sources placed in the simulation only represent the lowest strength sources present in the real material.

6.3 Results

There are three types of data output by the simulation that can be directly compared with observations from the experiment. These are:

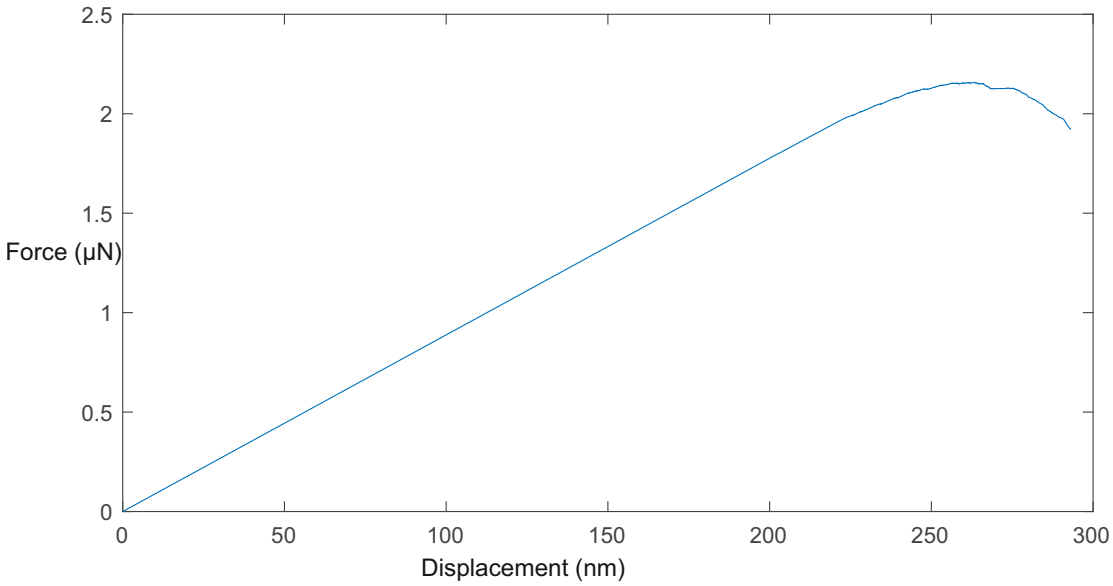
- Force/displacement data, which can be directly compared with the equivalent data from the experiment.
- The dislocation structure, which can be compared to the EBSD imaging of geometrically necessary dislocations (GNDs) in the test sample.
- Surface displacement data, which can be compared to the SEM imaging of the sample post test, with slip steps present.

The force displacement data is shown in Figure 6.5. By itself, the data appears typical of a beam bending test. However, when compared with the experimental data it is immediately apparent that the simulated cantilever starts to yield much earlier than expected. The simulated cantilever reaches its proportionality limit at around $202nm$ and $221nm$, whereas the proportionality limit for the experiment sample is at $323\mu m$ or $385\mu m$, although this is more difficult to determine given that the cantilever in the experiment does not perfectly elastically deform. However the point at which plastic deformation starts to occur is only one aspect of plastic behaviour. It is also possible to compare the way the cantilever plastically deforms after the proportional limit has been exceeded. By examining the plastic behaviour of both cases using the same aspect ratio, it becomes

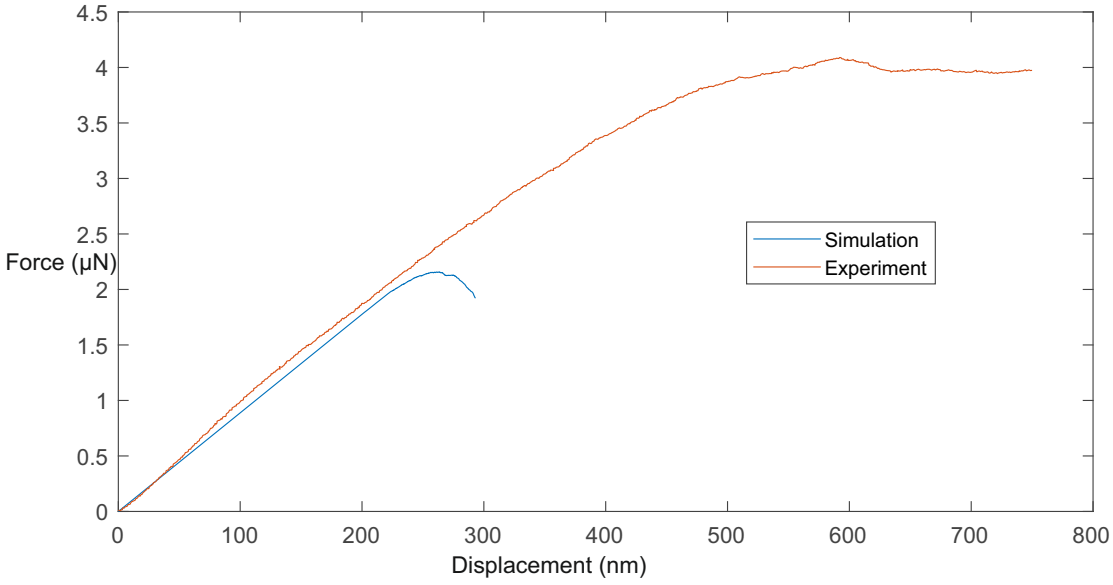
possible to do a qualitative comparison of yield behaviour. As Figure 6.6 shows, immediately after yield both cases show similar behaviour, The force required for additional displacement continues to increase until the elastic limit is reached and then the force required decreases. Eventually the force stops decreasing and remains roughly constant as the cantilever continues to deform. It is at this stage that the behaviour starts to differ. During the experiment, plastic deformation continues in a stable manner until the cantilever is unloaded, with relatively small changes occurring. Although there is noticeable softening followed by hardening, this occurs after large displacements that are not relevant to the initial plastic behaviour modelled by the simulation. Conversely, after a relatively short period of stable deformation, the simulation starts to show drastically increased plastic deformation and the force required to cause further displacement begins to drop quickly. Unfortunately, there was not time available to continue to run the simulation to see whether the plastic behaviour would stabilise again, given that the already large amount of time required to run the simulation increases even further as the total number of dislocations and their densities increases.

EBSD imaging of the experimental sample makes it possible to easily qualitatively compare the positioning of of dislocations in the simulation and the experiment after loading, as shown in Figure 6.7.

When comparing the arrangement of dislocations between the simulation and the experiment, there are a lot of similarities to be observed. The regions in which there are the most dislocations are not only in the same range but are of very similar shape. Additionally, there are lines of high dislocation density in the EBSD map that align with the planes that dislocations in the simulation are moving along. There are a few notable differences. The simulation has significantly more relative dislocation activity in the "bulk" material. The experiment has a higher density of dislocations close to the bottom face than close to the top face, while the inverse is true for the simulation, albeit to a lesser degree. There also appears to be more dislocation pile-up along the neutral axis in the simulation than occurred during the experiment. For additional, more quantitative comparison, it is possible to estimate the dislocation density by calculating the total length of dislocations present in the simulated bulk, $737.0\mu m$, and dividing by the volume they occupy, approximately $113.4\mu m^3$. This gives an average dislocation density of $6.50E+12m^{-2}$. This is about two orders of magnitude less the the densities of dislocations quoted on the EBSD map. While this is to be expected given that the experiment underwent over 20 times the plastic displacement of the simulation, the value for the dislocation density in the simulation is still a little small given that the amount of plastic displacement is proportional to the area swept by dislocations.



(a)



(b)

Figure 6.5: Force/displacement data showing (a) the full behaviour of the model and (b) the behaviour of the model in comparison to the experiment. Experimental data courtesy of J. Ast.

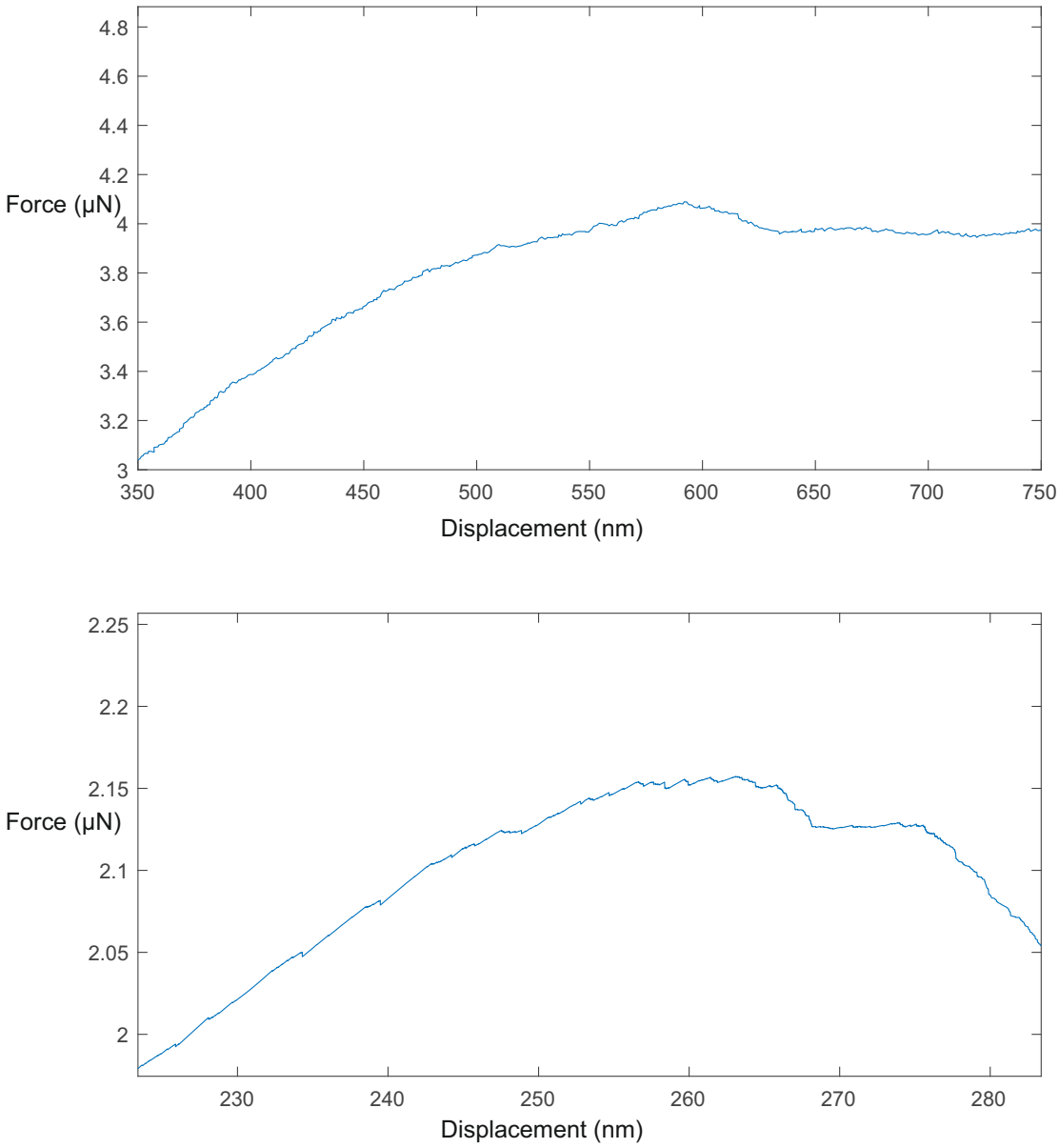


Figure 6.6: A qualitative comparison of the plastic behaviour of the (a) experiment with the (b) simulation by displaying their respective plastic responses within the same aspect ratio.

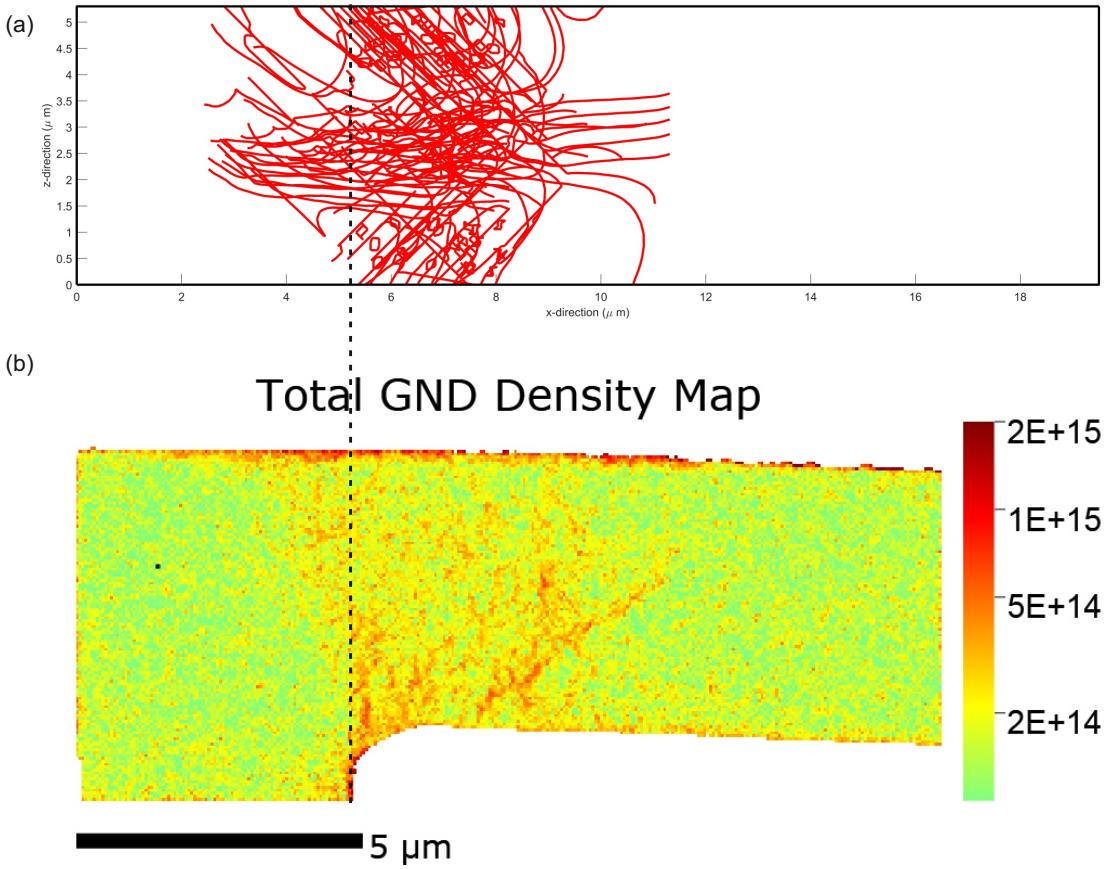


Figure 6.7: (a) A view of the simulated volume down the y-axis/[010] direction after the simulation and (b) An EBSD image of the cantilever tested in the experiment, taken along the same direction. The dotted line represents the division between the cantilever and the bulk. The units of GND density are m^{-2}. Image courtesy of J. AST.

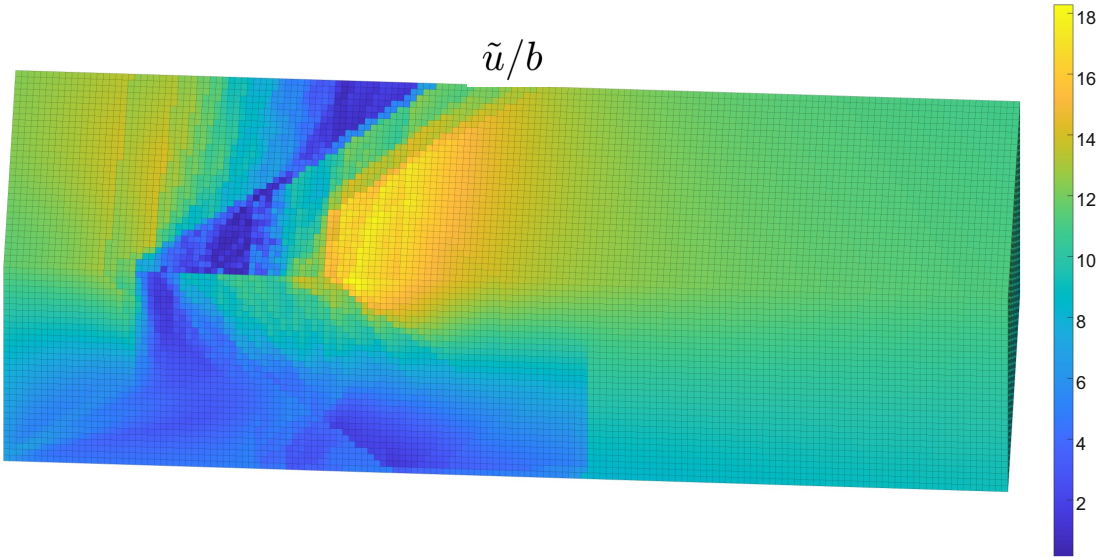


Figure 6.8: Post processing of the simulated volume with the surface displacement code reveals a slip step. The fixed end is on the left of the image and the scale is in units of the Burgers vector, $\boldsymbol{b}$. Apparent discontinuities are artefacts caused by surface element resolution.

The final point of comparison is that of slip steps on the surface of the bulk. Careful observation of the Figure 6.1 (b) reveals two slip steps in the top of the cantilever, which span the entire width of the beam. They appear to lie more or less parallel to the $[1\bar{1}0]$ direction, running from close to the bulk, on the side of the cantilever furthest from the viewer, along the top face to the edge and then continuing in the $[10\bar{1}]$ direction down the side of the beam visible in the image. This suggests the steps were caused by dislocations with $\frac{a}{2}[\bar{1}11]$ type Burgers vector. We can contrast this with the simulation by using the surface displacement code for post processing, giving Figure 6.8.

Figure 6.8 shows the simulated cantilever from the same angle as the cantilever in Figure 6.1 (b) and also has a prominent slip step in a similar position, However instead of being of the $\frac{a}{2}[\bar{1}11]$ type, this slip step seems to have a $\frac{a}{2}[11\bar{1}]$ type Burgers vector instead. There are many similarities between the experiment and the simulation, however their results are clearly not them. How valid is the *EasyDD* model then?

6.4 Discussion

It is important to examine the reasons for the differences and similarities between the behaviour of the simulation and that observed during experimentation. Such examination improves understanding of the validity of the model and aids in the discovery of new ways that it can be further improved upon.

The first difference to be addressed should be the fact that the experiment did not demonstrate pure elastic behaviour during the initial loading phase. This occurred because of plastic deformation at the indenter tip, which is clearly visible in Figure 6.1 (b), and is a known limitation of using *EasyDD* to simulate cantilever bending. It is possible to use *EasyDD* to simulate indenter behaviour, as demonstrated by Yu et al. [75], and a few adjustments to the cantilever model could enable this to be accounted for. To do this would require expanding the simulated volume to the full dimensions of the experimental sample, altering the loading conditions so that a small area of the top surface has a constant displacement instead of an edge and adding dislocation sources to the loaded end of the cantilever. This would essentially create two simulations in one but comes with many drawbacks. The additional dislocation behaviour would increase the runtime of the simulation and may even make the cantilever nature of the simulation redundant, since it would take so much time to model the indenter behaviour that dislocation motion close to the fixed end would not be simulated in any practical time-frame. Given that there is no real interaction between dislocations at the indenter tip and those close to the fixed end, adding the indenter behaviour to the cantilever model for the purpose of better understanding dislocation behaviour offers no noticeable advantage. The inaccuracy in the elastic response of the cantilever is therefore left as a known factor and allowed to persist in order to gain more information about dislocation behaviour close to the fixed end. It will become apparent that correctly gauging the balance between accuracy and practicality is key when using *EasyDD* to construct models of dislocation behaviour.

The next major difference between the simulation and the experiment is the onset of dislocation motion. Dislocation motion starts to allow plastic deformation much earlier in the simulation than is observed experimentally. The most likely cause of this is the size of the dislocation sources used. The magnitude of the shear stress, τ, required to activate a Frank-Read source is given in Equation 6.1

$$\tau = \frac{2Gb}{l} \tag{6.1}$$

where G is the shear modulus of the material, b is the magnitude of the Burgers vector and l is the distance between the fixed points in the source. It immediately becomes obvious that smaller sources require greater stresses to activate them, and therefor can only allow plastic displacement to occur after a larger total displacement has been reached. Looking at the difference between when plastic deformation was initiated for the simulation and the experiment, the sources can be estimated to be too large by an approximate factor of 2. It seems simple enough to merely decrease the source size and attempt the simulation again, but this has

its issues. As mentioned previously, the minimum size a source can be is l_{max}. This means that if we wish to model smaller dislocation sources, we must decrease l_{max}. This results in more segments being used to make up the dislocation network, which exponentially slows the progress of the simulation due to the extra segment-segment force calculations that need to be carried out. It was for this reason that the sources used were chosen, since they were the smallest sources known that could be simulated by the code with the hardware available in a reasonable amount of time. Another possible solution is to move sources closer to the neutral axis, so that a larger displacement is required before the stress acting on the sources becomes large enough to activate them. This has the disadvantage that it is introducing one non-physical aspect to offset another. In reality, dislocations start moving close to the top and bottom surface first in a mechanical bending test. To artificially prevent this from happening could allow for non-physical dislocation networks to form, meaning that the results of the simulation still do not match reality, merely in a different way. Modelling smaller sources is the more accurate way to perform the simulation and the most recent version of the code is most likely capable of it. However it should be expect to require a significantly longer period of time to model an equivalent amount of plastic behaviour.

This brings us to the plastic behaviour simulated using the current setup. While initially it seems to mimic the behaviour of the experiment at a lower and narrower range of applied displacements, additional dislocation action appears to cause softening in the simulation to a much larger degree than in the simulation. There are two possible reasons for this. The first is once again that it is due to the source size. It is possible that the larger sources require a small enough activation stress that the presence of dislocations nearby is enough to activate them, allowing more slip systems to activate that if the sources were smaller. It is also possible that the observed softening behaviour was simply the same type of softening observed in the experimental data but compressed over a smaller range of applied displacements. The only way to confirm this however would be to continue running the simulation for extended periods of time. The second reason is the existence of the simulated bulk. Dislocation have been observed moving into the simulated bulk, which as mentioned previously has a higher stress state than anywhere in the cantilever tested during the experiment at an equivalent total displacement. The behaviour of any dislocations that have moved into the simulated bulk must be written off as non-physical and taken into account when considering the meaningfulness of any results.

Another area where the existence of the highly stressed simulated bulk causes issues is when it comes to comparisons with the dislocation density map. The movements of dislocations in the cantilever seem to match well with what is ob-

served in the EBSD image, however there is much greater presence of dislocations in the simulated bulk that there are in experiment. Dislocations were allowed to enter the simulated bulk because it was judged that an impenetrable interface preventing them from doing so would cause nonphysical pile up phenomena. The simulated bulk is not the only aspect that causes issues with the compliance of the simulation with the EBSD map. The increased density of dislocations close to the bottom face of the experimental cantilever is likely cause by the corner at the bottom where the cantilever interfaces the bulk. This formation causes a local concentration of stress, a phenomenon that cannot be replicated with the current setup of the cantilever model. There is also the issue of the lower than expected dislocation density in the simulation. This is most likely cause by an insufficient number of sources in the simulated volume, resulting in fewer dislocations being formed, While adding more sources would cause the simulation to take more time to run, it would not be a catastrophic increase in run-time and could partially be offset with a larger loading rate due to the greater dislocation density, as discussed in Chapter 4. It should be noted that this would require additional convergence studies to be run.

The final difference between the simulation and the experiment that must be addressed is the type of slip step that each formed. Both types of slip step are formed by dislocation action close to the fixed end of the cantilever, which is to be expected as this is where the stress is highest. Dislocations exit the cantilever at various places but these slip steps become noticeable because of the number of dislocations exiting from the same part of the surface and it is the higher local stress relative to other parts of the cantilever that is the cause of this. However, in the experimental cantilever, as you pass the fixed end and move into the bulk the stress decreases. This results in the development of a slip step formed by dislocations moving from the fixed end into the cantilever. Conversely, in the simulation the part of the simulated volume designated as the bulk has an increased stress, causing more dislocations to move into the bulk than into the cantilever proper, resulting in the major slip step being formed with a different orientation to that in the simulation. Given the number of problems the simulated bulk causes, it would be beneficial to alter the model to reduce the impact it has. Unfortunately, this is easier said than done. The most obvious step to take would be to make the plane between the main body of the cantilever and the simulated bulk impenetrable to dislocations. However, the high local stress would cause dislocations to pile up at this newly formed interface, which would obstruct the action of any sources positioned close to the fixed end. This in turn would alter the plastic response of the simulation, since the sources closer to the fixed end contribute more to plastic displacement. The ideal solution would be to add to

the simulation so that the bulk can be simulated physically. Since the simulated volume cannot be concave, this would require and additional FE volume between which dislocations could move, essentially coupling two simulations together. The capability of the code to accurately calculate boundary conditions makes this theoretically possible, but further work is required for implementation.

In conclusion, the dislocation behaviour observed in the simulation is consistent with that observed in the experiment but the plastic response is not. It is therefor justifiable to say that the evolution of the dislocation network from its original configuration was modelled physically but the original configuration was not a good representation of the cantilever in the experiment before loading occurred.

6.5 Summary

EasyDD was used to simulate a tungsten cantilever with mixed results. While the motion of the dislocations was accurately modelled, the overall situation was not due to the initial dislocation structure being too far removed from reality. *EasyDD* has various strengths and weaknesses and it is important to play to its strengths while reducing the influence of its weaknesses in order to get the best results from simulations.

Chapter 7

Conclusions and Further Work

7.1 Retrospective

Through the progression of this thesis, work has been done to improve the capabilities of the DDD code *EasyDD*. Building on the *DDLab* framework, by streamlining previous methods and adding new ones, the performance of the code has reached new heights. Problems that would have been fatal for a simulation before have been removed and the code has become faster, more reliable and more versatile. That said, there are still limits to its capabilities.

The improvements discussed in Chapter 2 may have only been foundational work, but they were key for further development of the code. Not only did these improvements enhance the code's performance, but by improving the robustness of these basic building blocks it makes it possible to add greater functionality later without the code breaking. Such future-proofing is a vital part of creating a general purpose code so that new features can be added as and when they are required. The topological operations and time integrator are the foundation on which the rest of the simulation depends, so ensuring their smooth operation was key to all work moving forward.

Ironing out the kinks in the basic framework for the mobility laws was a vital step in ensuring that *EasyDD* performed consistently and physically. The different mobility laws are the main part of the code that make it material specific and ensuring that they are as close a representation of reality as possibly makes simulating real simulations more accurate. Additionally, it worked to improve performance so that simulations can run faster as well as more smoothly. Perhaps most crucially, these mobility laws allow for accurate physical representations of dislocation behaviour in more complex stress states, allowing for the possibility of creating a greater variety of simulation types as well as allowing the code to handle dislocation structures to develop to a greater level of complexity, increasing

the amount of plastic behaviour it is possible to simulate.

Understanding how the DDD part of the code works with the FEM part allows us to better know the full capabilities and limitations of the model. Realising how the dislocation network behaves at the spatial and temporal boundaries of the model allows us to properly construct simulations so that they have meaningful and physical results. The current handling of virtual segments works well for tracking where segments have exited the simulated volume, however the process of convergence studies by which appropriate loading and displacement rates are found is somewhat time consuming and increases the time required to set up new simulations.

Developing a fast reliable way to measure displacements caused by dislocations on the surface of the simulated volume has greatly improved the potential of the *EasyDD* code. This method allows for the modelling of more complex behaviour, removes a source of error and provides a post processing technique that makes it easier to compare simulations with real world examples, increasing the relevancy of DDP codes in the wider scientific community. It also opens the door to simulating more complicated scenarios by ensuring displacement boundary conditions can always be defined.

The attempt at simulating a tungsten cantilever demonstrated how the code performs when pushed to its current limits. While it is fairly good at modelling the motion and interactions of dislocation once yield has occurred, there are still resolution limits that can prevent sources of a sufficiently small size from being used. This means that, for some materials, it is not possible to accurately predict the yield point the the code as it currently exist without introducing an artificial activation stress for sources.

7.2 Further Work

Great strides have been made in improving the *EasyDD* code, but there are still limitations to be overcome and additional functionality that may be desired in order to simulate a wider range of scenarios.

Creating a new process by which collision and separation events are handled is a necessary step for simulating situations with large dislocations densities because such improved functionality is needed when many dislocations are interacting in close proximity over a short period of time. Without this improvement, attempting to simulate such situations will more often than not result in the simulation becoming stuck and no progress being made. This would also have the added benefit of improving the performance of other types of simulation.

While the new mobility laws work much better than the previous ones, there is

still room for improvement. Developing a better interpolation between drag coefficient for the BCC mobility law, for example, would allow for smoother modelling of cross-slip and more reliable simulations, since faults in the mobility law can still cause simulations to get stuck or give non-physical results. Additionally, mobility laws for other crystal structures need to be improved to the current standard, since this thesis focused primarily on BCC structures.

There are a few avenues for improvement when it comes to the process of coupling DDD models with FEM. The greatest achievement in this area would certainly be developing an analytic method for determining appropriate force or displacement rates, thus removing the need for lengthy convergence studies and greatly increasing the convenience of using codes such as *EasyDD*. Unfortunately the complexity of this problem is such that it may serve as the topic of an entire thesis by itself. A more reachable goal would be to improve the remeshing of virtual segments, so that the number of segments required to store slip step information is reduced without losing any information regarding the shape of slip-steps formed.

Now that it is possible to accurately measure the displacement caused by dislocations on any convex surface, complex shared boundary conditions between multiple FE problems can be implemented, allowing for greater possibilities when running a simulation. More complicated shapes could be modelled, such as ones involving concave surfaces, or previous sources of inaccuracy could be removed, such as properly simulating the bulk material that a cantilever is attached to as mentioned in Chapter 6.

The largest deficiency when simulating a tungsten cantilever was caused by the source size. This is the result of a resolution issue with the code. There are two possible further avenues of work that could remedy this. The first is to add amendments to the remeshing and force calculation code specifically for the handling of sources. This would be the easiest but may not be as robust in terms of future-proofing. The second is to rework the dislocation-dislocation force calculations so that they apply to elements instead of segments, allowing for sources to be of sizes of the same order of magnitude as the core radius. This has the added benefit of potentially improving the performance of the code by reducing the time required to calculate the forces acting on dislocations, however developing this new technique would be much more involved.

There are many additional areas of functionality that can be introduced to *EasyDD* in order to model more interesting phenomena. Work has already been done to model dislocation climb[76], creep behaviour [77], nano-indentation [75] and the effect of hydrogen diffusion on plasticity using this code.

7.3 Final Conclusions

EasyDD has become a powerful tool for modelling dislocation behaviour in finite objects. It can be applied to various scenarios and its versatility makes it unique among DDP codes, with a modular aspect that means further functionality can be readily added in the future. Although there are some scenarios it is difficult to model because of the computational costs, continual improvements are making more possible and many problems that previously seemed like intractable limitations to these modelling techniques have already had inroads made into solving them or been completely overcome. When combined with improving computer hardware, it seems that codes like *EasyDD* becoming commonplace for anyone investigating plasticity is just a matter of time.

All input files are available on the Tarleton group Github:
https://github.com/TarletonGroup/EasyDD

Bibliography

[1] B. Bromage, E. Tarleton, Calculating dislocation displacements on the surface of a volume, Modelling and Simulation in Materials Science and Engineering 26 (8) (2018) 085007. doi:10.1088/1361-651X/aae404.
URL http://stacks.iop.org/0965-0393/26/i=8/a=085007?key=crossref.5a96f3f9f84d0b74f27d39ed999ad16c

[2] J. B. Pethica, W. C. Oliver, Ultra-Microhardness Tests on Ion Implanted Metal Surfaces, in: Materials Research Society Symposia, 1982, pp. 373–379.

[3] G. R. Johnson, W. H. Cook, Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures, Engineering Fracture Mechanics 21 (1) (1985) 31–48. doi:10.1016/0013-7944(85)90052-9.

[4] V. Volterra, Sur l'equilibre des corps elastiques multiplement connexes, Ann. Ec. Norm. (Paris) 24 (1907) 401–517.

[5] G. I. Taylor, The Mechanism of Plastic Deformation of Crystals (1934).

[6] S. Amelinckx, Etchpits and Dislocations Along Grain Boundaries, Sliplines and Polygonization Walls, Acta Metallurgica 2 (6) (1954) 848–853. doi:10.1016/0001-6160(54)90039-5.

[7] J. W. Allen, K. C. A. Smith, XLVII. Electron Microscopy of Etched Germanium Surfaces, Journal of Electronics and Control 1 (4) (1956) 439–443. doi:10.1080/00207215608961431.
URL http://www.tandfonline.com/doi/abs/10.1080/00207215608961431

[8] I. M. Robertson, P. J. Ferreira, G. Dehm, R. Hull, E. A. Stach, Visualizing the Behavior of Dislocations - Seeing Is Believing, MRS Bulletin 33 (February 2008) (2008) 122–131. doi:10.1557/mrs2008.25.

[9] P.-O. Esbjørn, E. Jørgen Jensen, Computer studies of dislocation properties using two-dimensional model systems, J. Phys. Chem 37 (5) (1976) 1081–1091. doi:10.1016/0038-1098(76)90725-0.

[10] W. G. Hoover, N. E. Hoover, W. C. Moss, Steady State Dislocation Motion Via Molecular Dynamics * 63 (3) (1977) 324–326.

[11] W. G. Hoover, N. E. Hoover, W. C. Moss, Nonlinear dislocation motion via nonequilibrium molecular dynamics, Journal of Applied Physics 50 (2) (1979) 829–837. doi:10.1063/1.325997.

[12] V. Vitek, Theory of the core structures of dislocations in body-centred-cubic metals, Crystal Lattice Defects 5 (1974) (1974) 1–34.

[13] A. S. Bakenov, V. V. Kirsanov, Behaviour of Edge Dislocation Core Under Applied Stress, Physics of Metals and Metallography 4 (59) (1985) 74–79.

[14] T. Harry, D. J. Bacon, Computer simulation of the core structure of the screw dislocation in alpha iron containing copper precipitates part 1b 50 (2002) 195–208.

[15] M. Ghazisaeidi, D. R. Trinkle, Core structure of a screw dislocation in Ti from density functional theory and classical potentials, Acta Materialia 60 (3) (2012) 1287–1292. doi:10.1016/j.actamat.2011.11.024.
URL http://dx.doi.org/10.1016/j.actamat.2011.11.024

[16] J. Lothe, J. P. Hirth, Dislocation Dynamics at Low Temperatures, Physical Review 115 (3) (1959) 543–550. doi:10.1103/PhysRev.115.543.
URL http://link.aps.org/doi/10.1103/PhysRev.115.543

[17] E. Nadgornyi, Dislocation dynamics and mechanical properties of crystals, Progress in Materials Science 31 (C) (1988) 1–530. doi:10.1016/0079-6425(88)90005-9.

[18] A. N. Gulluoglu, C. S. Hartley, Simulation of dislocation microstructures in two dimensions. I. Relaxed structures, Modelling and Simulation in Materials Science and Engineering 1 (1992) 1–17. doi:10.1088/0965-0393/1/4/004.
URL http://iopscience.iop.org/0965-0393/1/1/001

[19] H. J. Frost, M. F. Ashby, Motion of a dislocation acted on by a viscous drag through an array of discrete obstacles, Journal of Applied Physics 42 (13) (1971) 5273–5279. doi:10.1063/1.1659936.

[20] G. Po, M. S. Mohamed, T. Crosby, C. Erel, A. El-Azab, N. Ghoniem, Recent Progress in Discrete Dislocation Dynamics and Its Applications to Micro Plasticity, Jom 66 (10) (2014) 2108–2120. doi:10.1007/s11837-014-1153-2.

[21] J. E. Dorn, J. Mitchell, F. Hauser, Dislocation dynamics, Experimental Mechanics 5 (11) (1965) 353–362. doi:10.1007/BF02326081.

[22] L. V. Brutzel, Dislocation-dynamics method (2015) 277–281.

[23] R. Madec, L. P. Kubin, Second-order junctions and strain hardening in bcc and fcc crystals, Scripta Materialia 58 (9) (2008) 767–770. doi:10.1016/j.scriptamat.2007.12.032.

[24] H.-J. Chang, Analysis of Nano indentation Size effect based on Dislocation Dynamics and Crystal Plasticity, Ph.D. thesis (2009).

[25] D. Gomez-Garcia, B. Devincre, L. P. Kubin, Dislocation patterns and the similitude principle: 2.5D mesoscale simulations, Physical Review Letters 96 (12) (2006) 8–11. doi:10.1103/PhysRevLett.96.125503.

[26] R. Madec, The Role of Collinear Interaction in Dislocation-Induced Hardening, Science 301 (5641) (2003) 1879–1882. doi:10.1126/science.1085477.
URL http://www.sciencemag.org/cgi/doi/10.1126/science.1085477

[27] C. Despres, Modélisation physique des stades précurseurs de l'endommagement en fatigue dans l'acier inoxydable austénitique 316L, Ph.D. thesis (2004).

[28] M. A. Shehadeh, E. M. Bringa, H. M. Zbib, J. M. McNaney, B. A. Remington, Simulation of shock-induced plasticity including homogeneous and heterogeneous dislocation nucleations, Applied Physics Letters 89 (17) (2006) 10–13. doi:10.1063/1.2364853.

[29] V. V. Bulatov, L. L. Hsiung, M. Tang, A. Arsenlis, M. C. Bartelt, W. Cai, J. N. Florando, M. Hiratani, M. Rhee, G. Hommes, T. G. Pierce, T. D. de la Rubia, Dislocation multi-junctions and strain hardening, Nature 440 (7088) (2006) 1174–1178. doi:10.1038/nature04658.
URL http://www.nature.com/doifinder/10.1038/nature04658

[30] N. O'Dowd, Computational Continuum Mechanics (2005). doi:10.1017/CBO9781139059992.

[31] S. S. Shishvan, S. Mohammadi, M. Rahimian, E. Van Der Giessen, Plane-strain discrete dislocation plasticity incorporating anisotropic elasticity, International Journal of Solids and Structures 48 (2) (2011) 374–387. doi:10.1016/j.ijsolstr.2010.10.010.
URL http://dx.doi.org/10.1016/j.ijsolstr.2010.10.010

[32] E. F. Holländer, The geometric equation of dislocation dynamics, Czechoslovak Journal of Physics 12 (1) (1962) 35–47. doi:10.1007/BF01688395.

[33] E. V. D. G. a. N. N. H. M. Cleveringa, Comparison Continuum of Discrete Dislocation and Plasticity Predictions for a Composite Material, Acta mat. 45 (8) (1997) 3163–3179.

[34] J. Huang, M. Meyer, V. Pontikis, Core structure of a dissociated easy-glide dislocation in copper investigated by molecular dynamics, Physical Review B 42 (9) (1990) 5495–5504. doi:10.1103/PhysRevB.42.5495.

[35] H. Yasin, H. M. Zbib, M. A. Khaleel, Size and boundary effects in discrete dislocation dynamics: Coupling with continuum finite element, Materials Science and Engineering A 309-310 (2001) (2001) 294–299. doi:10.1016/S0921-5093(00)01731-7.

[36] V. Shenoy, R. Miller, E. Tadmor, D. Rodney, R. Phillips, M. Ortiz, An adaptive finite element approach to atomic-scale mechanics-the quasicontinuum method, Journal of the Mechanics and Physics of Solids 47 (1999) 611–642. doi:10.1016/S0022-5096(98)00051-9.

[37] L. Kubin, A Guide to 3D DD Simulations, in: Dislocations, Mesoscale Simulations and Plastic Flow, Oxford University Press, Oxford, 2013. doi:10.1093/acprof:oso/9780198525011.003.0004.
URL http://www.oxfordscholarship.com/10.1093/acprof:oso/9780198525011.001.0001/acprof-9780198525011-chapter-04

[38] L. P. Kubin, G. Canova, The modelling of dislocation patterns, Scripta Metallurgica et Materiala 27 (8) (1992) 957–962. doi:10.1016/0956-716X(92)90456-O.

[39] M. Verdier, M. Fivel, I. Groma, Mesoscopic scale simulation of dislocation dynamics in fcc metals: Principles and applications, Modelling and Simulation in Materials Science and Engineering 6 (6) (1998) 755–770. doi:10.1088/0965-0393/6/6/007.
URL http://stacks.iop.org/0965-0393/6/i=6/a=007?key=crossref.17db6c78a1e2c279c62be2a7a81e0581

[40] C. S. Shin, M. C. Fivel, M. Verdier, S. C. Kwon, Numerical methods to improve the computing efficiency of discrete dislocation dynamics simulations, Journal of Computational Physics 215 (2) (2006) 417–429. doi:10.1016/j.jcp.2005.10.025.

[41] C. S. Shin, M. C. Fivel, M. Verdier, C. Robertson, Dislocation dynamics simulations of fatigue of precipitation-hardened materials, Materials Science and Engineering A 400-401 (1-2 SUPPL.) (2005) 166–169. doi:10.1016/j.msea.2005.01.079.

[42] M. Fivel, Tridis: 3D Discrete Dislocation Dynamics (2017).
URL http://www.numodis.fr/tridis/

[43] C. Lemarchand, B. Devincre, L. P. Kubin, Homogenization method for a discrete-continuum simulation of dislocation dynamics, Journal of the Mechanics and Physics of Solids 49 (9) (2001) 1969–1982. doi:10.1016/S0022-5096(01)00026-6.

[44] B. Devincre, mM a Dislocation Dynamics Simulation (2011).
URL zig.onera.fr

[45] H. M. Zbib, M. Rhee, J. P. Hirth, On plastic deformation and the dynamics of 3D dislocations, International Journal of Mechanical Sciences 40 (2-3) (1998) 113–127. doi:10.1016/S0020-7403(97)00043-X.
URL http://linkinghub.elsevier.com/retrieve/pii/S002074039700043X

[46] S. Groh, H. M. Zbib, Advances in Discrete Dislocations Dynamics and Multiscale Modeling, Journal of Engineering Materials and Technology 131 (4) (2009) 041209. doi:10.1115/1.3183783.
URL http://materialstechnology.asmedigitalcollection.asme.org/article.aspx?articleid=1428540

[47] H. M. Zbib, T. A. Khraishi, Chapter 84 Size Effects and Dislocation-Wave Interaction in Dislocation Dynamics, Dislocations in Solids 14 (2008) (2008) 207–249. doi:10.1016/S1572-4859(07)00004-6.

[48] K. W. Schwarz, Simulation of dislocations on the mesoscopic scale. I. Methods and examples, Journal of Applied Physics 85 (1) (1999) 108–119. doi:10.1063/1.369429.

[49] X. H. Liu, K. W. Schwarz, Modelling of dislocations intersecting a free surface, Modelling and Simulation in Materials Science and Engineering 13 (8) (2005) 1233–1247. doi:10.1088/0965-0393/13/8/003.

[50] V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, K. Yates, T. Arsenlis, Scalable line dynamics in ParaDiS, in: Proceedings of the ACM/IEEE SC 2004 Conference: Bridging Communities, 2004. doi:10.1109/SC.2004.53.

[51] W. Cai, A. Arsenlis, C. R. Weinberger, V. V. Bulatov, A non-singular continuum theory of dislocations, Journal of the Mechanics and Physics of Solids 54 (3) (2006) 561–587. doi:10.1016/j.jmps.2005.09.005.

[52] V. V. Bulatov, W. Cai, Computer Simulations of Dislocations, 1st Edition, Vol. 7, Oxford University Press, 2006. doi:10.1017/CBO9781107415324.004.

[53] M. Peach, J. S. Koehler, The Forces Exerted on Dislocations and the Stress Fields Produced by Them, Physical Review 80 (3) (1950) 436–439. doi:10.1103/PhysRev.80.436.

[54] A. Lehtinen, F. Granberg, L. Laurson, K. Nordlund, M. J. Alava, Multi-scale modeling of dislocation-precipitate interactions in Fe: From molecular dynamics to discrete dislocations, Physical Review E 93 (1) (2016) 1–9. doi:10.1103/PhysRevE.93.013309.

[55] A. Arsenlis, ParaDiS: Dislocation Simulations (2012).
URL paradis.stanford.edu

[56] S. Aubry, M. Rhee, G. Hommes, V. V. Bulatov, A. Arsenlis, Dislocation dynamics in hexagonal close-packed crystals, Journal of the Mechanics and Physics of Solids 94 (2016) 105–126. doi:10.1016/j.jmps.2016.04.019.
URL http://dx.doi.org/10.1016/j.jmps.2016.04.019

[57] E. V. d. Giessen, A. Needleman, Discrete dislocation plasticity: a simple planar model, Modelling and Simulation in Materials Science and Engineering 3 (5) (1995) 689–735. doi:10.1088/0965-0393/3/5/008.
URL http://stacks.iop.org/0965-0393/3/i=5/a=008?key=crossref.dbde9af8e79635c0f3fa40f1430db225

[58] A. Vattré, B. Devincre, F. Feyel, R. Gatti, S. Groh, O. Jamond, A. Roos, Modelling crystal plasticity by 3D dislocation dynamics and the finite element method: The Discrete-Continuous Model revisited, Journal of the Mechanics and Physics of Solids 63 (1) (2014) 491–505. doi:10.1016/j.jmps.2013.07.003.
URL http://dx.doi.org/10.1016/j.jmps.2013.07.003

[59] M. C. Fivel, G. R. Canova, Developing rigorous boundary conditions to simulations of discrete dislocation dynamics, Modelling and Simulation in Materials Science and Engineering 7 (5) (1999) 753–768. doi:10.1088/0965-0393/7/5/308.

[60] D. Weygand, L. H. Friedman, E. V. D. Giessen, a. Needleman, Aspects of boundary-value problem solutions with three-dimensional dislocation dynamics, Modelling and Simulation in Materials Science and Engineering 10 (4) (2002) 437–468. doi:10.1088/0965-0393/10/4/306.

[61] M. C. Fivel, A. A. Ei-Azab, Linking continuum mechanics and 3D discrete dislocation simulations, in: Journal De Physique. IV : JP, Vol. 9, 1999, pp. 9–261. doi:10.1051/jp4:1999926.

[62] J. P. Hirth, J. Lothe, Theory of Dislocations, 1982.

[63] S. Queyreau, J. Marian, B. D. Wirth, A. Arsenlis, Analytical integration of the forces induced by dislocations on a surface element, Modelling and Simulation in Materials Science and Engineering 22 (3) (2014) 35004. doi:10.1088/0965-0393/22/3/035004.

[64] R. de Wit, The Continuum Theory of Stationary Dislocations, in: Solid State Physics - Advances in Research and Applications, Vol. 10, 1960, pp. 249–292. doi:10.1016/S0081-1947(08)60703-1.
URL http://linkinghub.elsevier.com/retrieve/pii/S0081194708607031

[65] A. M. Hussein, J. A. El-Awady, Surface roughness evolution during early stages of mechanical cyclic loading, International Journal of Fatigue 87 (2016) 339–350. doi:10.1016/j.ijfatigue.2016.02.022.
URL http://dx.doi.org/10.1016/j.ijfatigue.2016.02.022

[66] D. M. Barnett, The displacement field of a triangular dislocation loop, Philosophical Magazine A 51 (3) (1985) 383–387. doi:10.1080/01418618508237562.
URL http://www.tandfonline.com/doi/abs/10.1080/01418618508237562

[67] M. Fivel, C. Depres, An easy implementation of displacement calculations in 3D discrete dislocation dynamics codes, Philosophical Magazine 94 (28) (2014) 3206–3214. doi:10.1080/14786435.2014.949326.

[68] D. Weygand, L. H. Friedman, E. Van Der Giessen, A. Needleman, Discrete dislocation modeling in three-dimensional confined volumes, Materials Science and Engineering A 309-310 (2001) (2001) 420–424. doi:10.1016/S0921-5093(00)01632-4.

[69] T. Jogi, S. Bhattacharya, Evolution of Interfacial Dislocation Networks in Particle-Strengthened Alloy Systems During High Temperature Creep: A Discrete Dislocation Dynamics Study, Transactions of the Indian Institute of Metals 69 (2) (2016) 507–512. doi:10.1007/s12666-015-0774-7.

[70] F. Ferroni, E. Tarleton, S. Fitzgerald, GPU accelerated dislocation dynamics, Journal of Computational Physics 272 (2014) 619–628. doi:10.1016/j.jcp.2014.04.052.
URL http://dx.doi.org/10.1016/j.jcp.2014.04.052

[71] A. Arsenlis, W. Cai, M. Tang, M. Rhee, T. Oppelstrup, G. Hommes, T. G. Pierce, V. V. Bulatov, Enabling strain hardening simulations with dislocation dynamics, Modelling and Simulation in Materials Science and Engineering 15 (6) (2007) 553–595. doi:10.1088/0965-0393/15/6/001.

[72] D. M. Barnett, R. W. Balluffi, The displacement field of a triangular dislocation loop-a correction with commentary, Philosophical Magazine Letters 87 (12) (2007) 943–944. doi:10.1080/09500830701601748.

[73] E. Tarleton, D. Balint, J. Gong, A. Wilkinson, A discrete dislocation plasticity study of the micro-cantilever size effect, Acta Materialia 88 (2015) 271–282. doi:10.1016/j.actamat.2015.01.030.

[74] CiDRA Precision Services, Machining of Materials. "Tungsten". AZoM. (2 2020).

[75] G. Li, Introduction to the Finite Element Method and Implementation with MATLAB, Cambridge University Press, 2020.
URL https://books.google.co.uk/books?id=n5nsDwAAQBAJ

[76] M. J. Aschwanden, R. W. Nightingale, D. Alexander, A. R. Winebarger, H. P. Warren, F. Liu, A. C. F. Cocks, S. P. A. Gill, E. Tarleton, An improved method to model dislocation self-climb You may also like Evidence for Nonuniform Heating of Coronal Loops Inferred from Multithread Modeling of TRACE Data An improved method to model dislocation self-climb (2020). doi:10.1088/1361-651X/ab81a8.
URL https://doi.org/10.1088/1361-651X/ab81a8

[77] F. X. Liu, A. C. F. Cocks, E. Tarleton, Dislocation dynamics modelling of the creep behaviour of particle-strengthened materialsdoi:10.1098/rspa.2021.0083.