

MATERIALS SIMULATIONS AT THE
ATOM-CONTINUUM INTERFACE: DISLOCATION
MOBILITY AND NOTCHED FRACTURE INITIATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Nicholas Patrick Bailey

January 2003

MATERIALS SIMULATIONS AT THE ATOM-CONTINUUM INTERFACE:
DISLOCATION MOBILITY AND NOTCHED FRACTURE INITIATION

Nicholas Patrick Bailey, Ph.D.

Cornell University 2003

We have solved three problems with a common theme of interfacing atomistic models with continuum models. The first is measuring the Peierls barrier for dislocation glide in a two dimensional material. The key features of this work are (1) efficient extrapolation of the infinite system limit from small simulations, through the use of multipole relaxation at the atom-continuum interface, and (2) the representation of the dependence on external parameters (in this case applied stress) in a compact way using a physically motivated functional form. The second problem is the initiation of fracture at sharp notches in single crystal silicon, a problem of current experimental interest in microfabrication. It is found that when expressed in atomic-scale units the critical stress intensity factor is almost independent of notch opening angle, as long as the interatomic potential does, in fact, produce brittle fracture. The third problem is the challenge of incorporating atomistic simulations in an adaptive manner in large scale continuum (finite element) simulations. Our method involves embedding such simulations within elements in an overlapping sense, and avoids some of the complexity associated with alternative methods. We solve these three problems through the development of a flexible, modern, powerful molecular dynamics package, known as DigitalMaterial. We describe the design of the software, which is fully object-oriented. What makes this

package different from others is the use of a component-based approach based on software engineering methods known as Design Patterns. The interfaces for these components are very clearly defined, allowing components to be interoperable and to be easily driven from a high level scripting environment.

Biographical Sketch

Nicholas Bailey was born in 1974 in Dublin, Ireland, to Mary and Patrick Bailey. He attended the Irish speaking school *Scoil Bhríde* before entering the Jesuit-run Gonzaga College SJ, at which time his knowledge of the Irish language started to decrease linearly with time, although he was happy to learn French, Latin and Greek, and become interested in science, in particular physics. In 1992 he entered University College Dublin's science Faculty, graduating in 1996 with a B. Sc. in Mathematical Physics and Experimental Physics. Having been accepted into Cornell University to pursue graduate study, Nicholas arrived in Ithaca, NY on August 23, 1996, one day after completing his last undergraduate examination. After five semesters of teaching, which he enjoyed very much, especially the classes for physics majors, he joined Professor James Sethna's group doing software development for multiscale modeling of materials. Much programming in C++ and Python took place, along with learning how to design a large scientific code properly, as well as thinking about dislocations and cracks and finite elements.

During all this time spent being educated he participated in a range of contact sports, starting with rugby (Gonzaga College Senior Cup Team 1991-1992, Old Belvedere R. F. C. McCorry Cup team 1992-1994), adding during college Shotokan karate (ITKF 2nd *kyu* (brown) 1995) and in graduate school competitive ballroom

dance (collegiate competitions, gold level in Spring 2002). His interest in dance extends also to swing, modern, tap and more.

Nicholas has accepted a post-doctoral position in the Technical University of Denmark (DTU) working with Karsten Jacobsen and others in the Center for Atomic Scale Materials Physics (CAMP). He will be leaving for Denmark within a day or two of defending this thesis, thus echoing the manner in which he first arrived in Ithaca.

To my parents, and to Wendy

Acknowledgements

In the last six years many people have contributed to the great time I have had here at Cornell. First I should mention my adviser Jim Sethna, who has kept me interested in my work, making it exciting all the time, always having new ideas, to the point that I sometimes felt like my current projects were like a large pile of books I was carrying in my arms, barely able to see over the top of them, but more or less managing to hold onto them; encouraging me when I questioned the validity of my work and helping me believe in myself. The members of our group, the post-docs Thierry Cretegy and Drew Dolgert, with whom I collaborated intensely and productively on software development, as well as Markus Rauscher, who helped me understand how to be a scientist; and students Lance Eastgate and Connie Chang with whom I had many useful conversations. I owe a debt to Chris Myers of the Cornell Theory Center for teaching me about software design in general and the benefits of Python and SWIG in particular, as well as always being available to help with sticking points regarding their use. I remember quite distinctly the moment when I found I could understand Chris's discourses on software design notions. In my extended research group I would like to mention Tony Ingraffea, who is probably most responsible for my appreciation of engineering problems and methods, as well as Paul (Wash) Wawrzynek, Gerd Heber and Erin Iesulauro for

helpful advice and discussions.

In the rest of the physics department I would especially like to mention former house-mates Tom Glickman, Cayce Butler, Kevin O'Neill, Mike Quist and Aash Clerk and all the people who regularly appeared in the donut room, E18 Clark Hall, between 4.00 and 4.30 every weekday (except Monday and Tuesday during the semester) and participated in the most random conversations about everything. I think 90% of my understanding of the America and Americans is based on those conversations, possibly along with the many Friday nights spent at the Statler bar drinking pitchers of Saranac.

I would like to thank my parents and brother and sister for regular phone calls , occasional visits, and constant support. I would like to thank my girlfriend Wendy McRae for much advice and support during times of self-doubt, and particularly for her careful proof-reading and editing of most of chapters 4 and 5 in the late stages of their preparation. One large group of people which should definitely not go unmentioned or unthanked is the Cornell Ballroom Dance Club, a group of people and an activity which most definitely helped keep me sane while working hard. I truly discovered a new side of myself through them, and something I will enjoy doing forever: ballroom dance, whether in the context of social dancing, competing or teaching.

Finally I would like to thank the weather, for not being as bad, in winter, as I initially feared when coming from Ireland.

Table of Contents

Biographical Sketch	iii
Dedication	v
Acknowledgements	vi
Table of Contents	viii
List of Tables	xii
List of Figures	xiii
1 Introduction	1
1.1 Multiscale modeling	4
1.2 Parameter Passing: small scales to large	12
1.2.1 Examples: some specific defects	14
1.3 Mixed atomistic/continuum modeling: to couple, to embed, to extend	17
1.3.1 Coupling to finite elements	17
1.3.2 The quasi-continuum method	19
1.3.3 FE meets Digital Material: OFE/MD	21
1.4 Object-oriented, Design Patterns-enabled molecular modeling: Digital Material	22
1.4.1 Python drives C++ creates power with control	25
2 Dislocation Mobility: Accurate Peierls Barriers	28
2.1 Introduction : mobility of dislocations	28
2.1.1 Flexible boundary conditions	31
2.2 Boundary kinematics: Flex-S with moving center	34
2.2.1 Linear elasticity: general solution in 2D	35
2.2.2 Multipole coefficients and center as boundary degrees of freedom	42
2.3 Energetics	45
2.3.1 Atomistic energy: “cut-Lennard-Jones” interatomic potential	46
2.3.2 Continuum energy: linear analysis	49
2.3.3 Continuum energy: nonlinear analysis	52
2.3.4 Blending of atomistic and continuum energy functionals	54
2.3.5 Accounting for external stress	57

2.4	Dynamics of embedded dislocation system	61
2.5	Computing the energy barrier	64
2.5.1	Initial and final states	65
2.5.2	Formulation of Nudged Elastic Band for extended system	67
2.5.3	Running the simulation	75
2.6	Representing the data: Functional Forms	77
2.6.1	Symmetry	79
2.6.2	Singularity	80
2.6.3	Simple model	81
2.6.4	Subtleties, extra physics	83
2.6.5	Dependence on σ_{xx} and σ_{yy}	84
2.6.6	Fitting procedure	84
2.7	Results and discussion	85
2.7.1	Size dependence;	85
2.7.2	Stress dependence	88
2.A	2D versus 3D elasticity	93
2.B	Lagrangian versus Eulerian coordinates	95
2.C	Transforming Eulerian to Lagrangian coordinates in continuum energy analysis	98
2.D	Relating changes in elastic energy to work done on boundaries	101
3	Fracture of Notched Single Crystal Silicon	104
3.1	Introduction	104
3.1.1	Elastic fields near a notch	105
3.2	Simulation	108
3.2.1	Geometry	108
3.2.2	Potentials	110
3.2.3	Boundary Conditions	111
3.2.4	Critical stress intensities	113
3.3	Results	114
3.3.1	Observed fracture behavior	114
3.3.2	Critical stress intensities	121
3.4	Discussion	123
3.4.1	Critical stress intensities	123
3.5	Summary	129
3.6	Acknowledgments	129
3.A	Units and Conversions	130
3.B	Stroh formalism for notches	131
3.C	Subtleties associated with taking powers of complex numbers	134
3.D	Crystal orientations	137
3.E	Subtlety in defining mode I/mode II in crack case (no reflection symmetry)	137

4	Digital Material, A Modern Molecular Dynamics Code	141
4.1	Introduction: The complexities of today’s simulations, as driven by multiscale materials modeling	141
4.1.1	Digital Material design goals	143
4.2	Components	146
4.2.1	ListOfAtoms	146
4.2.2	Potential	149
4.2.3	Mover and Transformer	151
4.2.4	NeighborLocator	155
4.2.5	BoundaryConditions	159
4.2.6	Constraint	161
4.2.7	AtomsInitializer	167
4.2.8	ListOfAtomsObserver	170
4.3	Infrastructure	173
4.3.1	Parallelization	173
4.3.2	Serialization	179
4.3.3	Graphics/Visualization	185
4.4	Summary	189
5	Overlapping Finite Elements/Molecular Dynamics (OFE/MD)	190
5.1	Motivation and purpose	190
5.2	Principles of the coupled OFE/MD model	194
5.2.1	Kinematics	194
5.2.2	Energetics	196
5.2.3	From energy to forces	198
5.3	Dynamics and algorithms	201
5.3.1	Zero temperature algorithm	203
5.3.2	Finite temperature algorithm	204
5.3.3	Applying the displacement field to core atoms	205
5.4	Model geometries	206
5.4.1	One-Brick; sphere of atoms	206
5.4.2	Two-Brick; layer of atoms	209
5.4.3	Cracked silicon plate	209
5.4.4	Cracked Cube	211
5.5	Technical details	212
5.5.1	Partial Stiffness Matrices	212
5.5.2	Linear solve step	228
5.5.3	Subtleties	230
5.6	Infrastructure	238
5.6.1	Software engineering	238
5.6.2	Interfacing with a continuum model via data base	240
5.6.3	Diagnostics, visualization and feature detection	244

5.7	Future applications of OFE/MD	245
5.7.1	Cohesive law extraction	246
5.7.2	Dislocations/surface/grain boundary interactions	247
5.7.3	Continuum crack growth	248
5.A	Quadratic shape functions for hexahedral (brick) elements	249
5.B	Quadratic shape functions for wedge elements	252
5.C	Quadratic shape functions for tetrahedral elements	253
5.D	Transformations between natural coordinates in cracked-cube model	255
5.E	Symmetry considerations for internal relaxation parameter in a diamond-cubic lattice	259
5.F	Nodal forces due to symmetric force dipole at the center of a cubic element.	260

List of Tables

1.1	Defects and their dimensionalities.	8
2.1	Cut-Lennard-Jones parameters.	47
2.2	Quadratic and cubic elastic constants for our cut-Lennard-Jones potential and for that of Holian et al.	49
2.3	Displacement, strain, energy density and total energy within a radius R for different multipoles and pairings of multipoles—linear analysis.	51
2.4	Simulation parameters for Figs. 2.7 and 2.8.	76
2.5	Fit parameters for CLJ potential.	92
3.1	Surface energies for silicon according to mSW, EDIP and MEAM potentials.	112
3.2	Critical stress intensity values for different geometries and potentials, including experimental data from Refs. [66, 67].	124
3.3	Angles of slip planes and crack planes and ratio of shear to normal stress for different geometries.	127
3.4	Unit conversion factors for K	131
4.1	Methods for Serializable, DMWriter and DMReader.	182
5.1	Internal relaxation parameters K_{123} and ζ for Si potentials.	233
5.2	Transformation vectors and matrices for transforming tetrahedron natural coordinates.	255
5.2	Transformation vectors and matrices for transforming tetrahedron natural coordinates.	256
5.2	Transformation vectors and matrices for transforming tetrahedron natural coordinates.	257
5.2	Transformation vectors and matrices for transforming tetrahedron natural coordinates.	258

List of Figures

2.1	Division of atomic model into different regions when using fixed boundaries, original Flex-S scheme and extended Flex-S scheme. r_c is the cutoff distance of the potential.	31
2.2	Plots of the $n = 1$ displacement fields; clockwise from top left: $j = 0, 1, 3, 2$	41
2.3	Transition function with limits $ \vec{X} = 6$ and $ \vec{X} = 16$	55
2.4	Undeformed and dislocated lattice with the lattice vectors, and lines indicating the choice of inserted half-planes, as well as the mirror plane.	64
2.5	Effective potential for dislocation glide in (a) infinite crystal, (b) “centered” simulation and (c) “offset” simulation.	67
2.6	Effect of using climbing image (CI) technique in the Nudged Elastic Band method. The number of MDmin iterations was 1000 in both cases.	74
2.7	Typical barrier energy profile from simulation.	77
2.8	Parameter values along transition path.	78
2.9	Schematic of dislocation potential energy under shear stress σ	79
2.10	Plot of barrier versus stress for sinusoidal potential.	83
2.11	Dependence of zero-stress barrier height on core region size, for different numbers of boundary parameters (CLJ potential), without continuum energy.	85
2.12	Zero-stress barrier height vs. core region size, far field energy included.	86
2.13	Zero stress barrier with and without continuum energy terms, with eight boundary parameters.	88
2.14	Contour plot of Peierls barrier $(\sigma_{xx} - \sigma_{xy})$	90
2.15	Contour plot of Peierls barrier $(\sigma_{yy} - \sigma_{xy})$	90
2.16	Contour plot of Peierls barrier $(\sigma_{xx} - \sigma_{yy})$	91
2.17	Contour plot of Peierls barrier (pressure- σ_{xy}).	91
2.18	Histogram of fractional errors in multidimensional nonlinear fit to barrier data.	92
2.19	When the dislocation glides a distance b , the work done on the boundary of squares $S1$ and $S2$ is $0.58\sigma_{xy}b^2$, that on circles $C1$ and $C2$ $2\sigma_{xy}b^2/3$. On rectangle $R1$ it is very nearly $\sigma_{xy}b^2$	102

3.1	(a) Notch schematic and notation; (b) silicon crystal with a notch; darker layer is fixed boundary atoms.	106
3.2	Normal (a) and shear (b) stresses on radial planes as functions of plane angle, for $\gamma = 70^\circ$	109
3.3	mSW-crack.	115
3.4	EDIP, crack.	116
3.5	MEAM, crack.	116
3.6	mSW- 70°	117
3.7	EDIP- 70°	118
3.8	MEAM- 70°	119
3.9	mSW- 90°	119
3.10	EDIP- 90°	120
3.11	MEAM- 90°	120
3.12	mSW- 125°	121
3.13	EDIP- 125°	122
3.14	MEAM- 125°	123
3.15	Computed critical stress intensities for the three potentials and experiment.	125
3.16	Det(\mathbf{K}) versus λ using difference of logs.	136
3.17	Det(\mathbf{K}) versus λ using log of the quotient.	136
3.18	Angular dependence of σ_{yy} for the crack, using Sih and Stroh formalisms. The Stroh curve has the higher peaks.	140
5.1	Two-dimensional atomistic region embedded entirely within a single element, and “natural coordinates” of the element.	195
5.2	Meshes for “OneBrick” and “TwoBrick” models.	207
5.3	Stress-strain curves for One-Brick model, first with x -displacements only, then with lateral relaxation allowed.	208
5.4	Stress-strain curves for Two-Brick model, first with x -displacements only, then with lateral relaxation allowed.	208
5.5	2D Mesh for Silicon thin plate, in deformed configuration.	211
5.6	Mesh for cracked-cube model.	213
5.7	15-noded wedge element with quarter-points.	218
5.8	Natural coordinates and node numbering for wedge element.	220
5.9	Wedge configuration in real space, and natural coordinates correspondingly rotated.	221
5.10	Natural coordinates and node numbering for a tetrahedron, and possible locations of the atomistic region.	224
5.11	Coordinate systems for first four tetrahedron orientations.	226
5.12	Class diagram for OFE/MD simulation software.	241
5.13	Use of a relational data base.	242
5.14	Web interface to OFE/MD simulations.	243
5.15	Simple cohesive zone model, with expanded view of the cohesive zone.	247

5.16	A crack growth step. Moving the actual node means changing how things look even in undeformed coordinates.	249
5.17	Node-numbering for 20-noded wedge elements.	251
5.18	Node-numbering for 15-noded wedge elements.	252
5.19	Node-numbering for 10-noded tetrahedron elements.	254

Chapter 1

Introduction

In this thesis we present two physics problems and two problems in software for multiscale modeling of materials. The first physics problem, the subject of chapter 2, is the computation of the Peierls barrier of a single dislocation in an infinite medium as a function of applied stress. This quantity is a crucial ingredient in a rule for specifying the velocity of a dislocation as a function of stress, such as would be required in a dislocation dynamics simulation. Our focus is on getting results as accurate as possible from simulations as small as possible. The ability to use small simulations, containing of the order of hundreds of atoms, is significant because it makes possible the use of *ab initio* methods such as the local density approximation (LDA) to compute these quantities, thus achieving accuracy in the sense of obtaining a believable representation of real materials. For the purpose of developing the technique, we have chosen to work with a two-dimensional crystal and use a very simple interatomic force law, the Lennard-Jones pair potential. The motivation for studying small systems stems also in part from a wish to avoid competing with those groups whose specialty is very large systems. The state of the art in atomistic simulations involves billion atom simulations on massively parallel

computers. These are impressive, beautiful simulations, but given the computing resources and time needed to perform just one simulation, it is not generally possible to perform a systematic study of a particular system under all possible conditions. A whole computer budget can be used up getting a result for one particular material, in one particular geometry with one particular set of external conditions (temperature, applied stress, etc.) It is not clear what useful results can be taken away from the outcome of a single such simulation.

The second physics problem, the subject of chapter 3, is a computation of the threshold for fracture initiation from notched samples of single crystal silicon. The threshold refers to a quantity called the *stress intensity factor* which characterizes the linear elastic state (displacement and stress fields) near a notch, in a manner analogous to standard fracture mechanics (a crack is just the limiting case of a notch whose opening angle is zero). This has been inspired by some recent experiments, some recent theoretical work seeking to explain the apparent large dependence, of critical stress intensity factor, K_c , on notch opening angle, and the observation by Sethna that because the units of the stress intensity factor actually depend on the opening angle, the apparent angle dependence is largely a consequence of using inappropriate units. When atomic-scale units are used, such as eV and \AA , the plot becomes almost flat. This turns out to be more the case in the simulation data than in the experimental data, mainly because of one point in the experimental data noticeably deviating from the apparent almost-flat curve. In this work the limitations of interatomic potentials for silicon are made clear, in particular regarding the question of whether a potential produces brittle fracture, like real silicon, or not. Only one of the three potentials used produces satisfactorily realistic behavior.

Chapter 4 is a fairly detailed presentation of the design principles of the molecular dynamics (MD) software package we have developed, called DigitalMaterial. We have attempted to make as much use as possible of modern software engineering ideas, and to think carefully about the key components of an MD code, with a view to providing maximum flexibility to the user. Flexibility means allowing not just different interatomic potentials, but different types of boundary conditions, different kinds of constraints, different kinds of dynamics and different kinds of tools for computing quantities of interest. A user of the code should rarely have to delve into the inner workings of it, but only have an understanding of what the various components are, and how their interfaces operate.

The work described in chapter 5 is of a somewhat hybrid nature, its aim being partly to develop a new technique for coupling finite elements to atomistic simulations, and partly to develop software which is “adaptive”, meaning that the code allows atomistic simulations to be dynamically incorporated into continuum simulations. The state of development of this work is such that it is poised to applied to interesting computational materials problems.

In the rest of this chapter we provide background for the field of multiscale modeling of materials, starting from general concepts and proceeding to a description of the “parameter passing” paradigm in the context of specific defects. We then describe some approaches to coupling atomistic simulations with finite element models, including our own approach, *Overlapping Finite Elements/Molecular Dynamics* (OFE/MD). Finally we introduce the ideas constituting our approach to software design: object-orientation, design patterns and the use of high level scripting languages.

1.1 Multiscale modeling

In recent times two fields of scientific endeavor have emerged as grand arenas of interdisciplinary research: materials science and biology. Materials science has been interdisciplinary since its emergence as a field; biology, the oldest science apart from astronomy, has become so in recent decades. Both increasingly count physicists, mathematicians, computer scientists and engineers among their practitioners, as advances in experimentation and computation have begun to permit detailed analysis at atomic scales in both fields. In fact the distinction between the fields, for practical purposes, might well be gone in another twenty years —by then we could be designing materials almost as complex and structured as living tissue.

We are not at that stage yet, but there is already enough structure—as much by design as by nature—in real materials to make modeling them a challenge. The realistic modeling of materials is important for innumerable engineering applications, ranging from semiconductor devices to aircraft fuselages. It is desired to understand the response of the material under the various conditions that the engineered part will be subject to in its lifetime. Specifically one needs to be able to answer the questions: (1) Will the part perform in the manner desired? and (2) Under what circumstances is the part likely to fail (and how can we then maximize its lifetime)? The kinds of properties that are relevant in a real application include mechanical, chemical, electronic, thermal, thermodynamic, etc. We will restrict our focus to mechanical properties. In large scale engineering applications, these are typically the most important and/or the most difficult to reliably account for. Engineers are plagued by the variability of mechanical properties such as strength and toughness.

To illustrate this point, consider a steel airplane wing. Suppose we have a design for the wing, specifying the shapes and sizes of all the components. Let us ask the question: what is the total mass of the wing/bridge? We know the density of steel and can add up the volumes of the pieces to get the total volume, then multiply by the density to get the mass. Suppose next we ask the question: How much will the structure bend under its weight, or subject to some applied force? This is little harder, but knowing the elastic moduli of the material we can use a finite element program to do a stress analysis of structure subject to the load and compute displacements. This can be done fairly reliably—the main error source, namely, the discretization into elements, can be systematically improved by refining the finite element mesh (under we run out of computer power). What if we now ask the question: If the structure is subjected to a periodic applied force, how many cycles can we expect to take place before the structure will fail by fracture? Or even without considering cyclic loading, suppose a small crack somehow appears somewhere on the structure, how fast will it grow—and can the plane land before this happens? Given our experience with density and elastic properties, we might guess at the existence of a property called *fracture toughness*, which might be more than one number, characterizing the growth of small or large cracks, under cyclic or monotonic loading. We would do a stress analysis, employing the concepts of fracture mechanics (which explains how to characterize the stress state near a crack and relate it to fracture toughness).

Fracture toughness is indeed a property that engineers talk about. The problem is that one cannot find a value characteristic of, say, a particular kind of steel, such that a fracture mechanics analysis will then give a reliable prediction. It is very hard to measure fracture toughness experimentally; fracture mechanics texts[4]

devote many pages to specifying precise rules for the geometry to be used in measuring toughness, because the measured values depend rather more on geometry than a true material property ought to. This is very unlike the case for elastic constants. Because of this poorly understood dependence, when it comes to designing structures, engineers have to use the so-called *safety factor* approach: rather than design the structure so that the load during its lifetime will never exceed the load that will cause failure, design so that the load will never exceed, say, one third (safety factor = 3) of the load needed to cause failure, or equivalently design so that the load required to cause failure is three times the maximum load anticipated during the lifetime. Including the safety factor compensates for the fact that the estimate of the load required to cause failure is uncertain because our understanding of the mechanical properties of the materials is limited. This approach works for the most part, but there are two problems. First, over-engineering in this way often means using more material or more expensive material than might actually be necessary, so that the cost of the structure is significantly more than it could otherwise safely be. Second, even with an apparently reasonable safety factor, structures still fail at loads below the design load (see [4], chapter 1, for examples). Thus it is very desirable to be able to accurately understand and model mechanical properties of materials for the purpose of predicting failure.

Let us now consider why there is such variability between samples. This is the point where we need to start looking deep inside the material, on various length scales. Different length scales present different material processes, and contribute differently to the observed macroscopic behavior. First, the smallest scale of interest is the atomic scale, where distances are measured in nanometers and the motion of individual atoms is under consideration. The interaction between

atoms is important for determining this motion. From the atomic interactions come macroscopic properties such as the density, elastic constants and melting point. At a scale where a relatively small number of atoms are under consideration, the material looks the same from one point to another. For this reason the elastic constants of the macroscopic body are quite well defined¹. We say this property is dominated by the atoms. However for a property such as fracture toughness of a metal, it is not the atoms as such, but rather the *defects* that dominate the behavior.

Let us consider metals as our main engineering materials of interest. An atomic description of a metal is in terms of a crystal, but materials (metals) used in engineering applications are never single crystal; they are polycrystal. This introduces questions involving the size distribution of the individual crystals—now called *grains*—and the properties of their boundaries. Furthermore even the individual grains are not perfect single crystals, because they have point defects: vacancies (missing atoms) and interstitials (extra atoms) or impurities (atoms substituted with atoms of a different element) as well as line defects—dislocations. Finally, most engineering metals are not pure, but rather are alloys. A steel for example, is iron with a range of other elements making up a small percentage of the total composition. The other elements often are not spread uniformly throughout the host metal (matrix) but are clumped together as relatively large (on the atomic scale) particles known as inclusions or precipitate particles (since the process of their forming is known as precipitation). We may think of these as volume defects.

¹It does matter, though, whether it is a single crystal or polycrystal—in the former case, the anisotropy of the elastic constants will be apparent on the macroscale, whereas in the latter case the macroscopic elastic constants will be an average over orientations.

Table 1.1: Defects and their dimensionalities.

Dimension	Defect types
0	vacancy, interstitial, impurity atom
1	dislocation (edge/screw), grain boundary junction, crack front
2	grain boundary, surface (e.g. from a crack)
3	precipitate particle, voids

We now have defects of all dimensionalities, as illustrated in table 1.1.

Now for the key issue: metal that is used in engineering applications is full of all of these kinds of defects. They make up a structure within the metal known as the *microstructure*. From a macroscopic point of view the microstructure may be considered as the internal state. Its existence is the cause of perceived *history dependence* of a material, namely the history of its processing. If we think that we only need a few macroscopic variables (stress, temperature, material constants) to describe a material and notice a history dependence in the relationship between them, what this tells us is that our description of the state is not complete: the microstructure is not accounted for. This is in contrast to the statistical mechanics of say a gas, where a few macroscopic variables, and an equation of state, really are enough to describe the macroscopic behavior. A (crystalline) solid is different because it has defects, which are generally non-equilibrium phenomena, and particularly because on experimental time scales they do not equilibrate away. The microstructure may be quantified in terms of densities of defects (dislocation density, density of precipitate particles), or an orientation distribution function (for grains). A key issue for developers of continuum theories of microstructure evolution is to determine what the appropriate variables are for describing the

microstructure. This is very non-trivial because simple averages (such as concentration) are not necessarily sufficient: defect-driven processes tend to be governed by extreme-value statistics, where the tails of distributions are more important than the means. For example, crack initiation might be determined by the size of the largest microcrack present in the material, rather than the mean microcrack size.

It is important to realize that although we use the word “defects”, it should not be assumed that defects are bad. In fact materials are often engineered to have a particular microstructure, for example a specific density of precipitate particles of a specific size, or a particular grain size, or dislocation density. In general such things increase the hardness (resistance to plastic flow) and fracture toughness, etc. of the material (thus pure metals are never used in structural engineering). For example, if the grains are small then dislocations can travel less distance before hitting a grain boundary, which impedes their motion. Thus the material is harder. Another cause of hardness in a material, at least in a metal, is the interactions between dislocations. Dislocations will move under applied stress, giving rise to macroscopic plasticity. However at large strains when the dislocation density is high, they eventually become tangled in each other, which drastically impedes further motion. Because this process happens as the material is “worked”, it is called *work hardening*. A simple example of work hardening can be observed when you try to straighten out a paper clip.

We see that the dynamics of a material on microscopic scales are very complicated. This is the challenge of materials modeling: the number of processes that take place, on many length scales. There are two contexts, roughly speaking, in which materials scientists seek to usefully model material microstructure. First is

the processing of the material, which may consist of various treatments involving heating and working the material (“heat ’n’ beat”). The aim of processing is to produce a microstructure which gives desirable material properties such as high toughness, or creep resistance, etc. The second context is modeling the material as part of an engineered structure in order to ascertain its performance and likelihood of failure under working conditions. Clearly if one had the power to model both the processing and operational behavior of a material one could then design materials at will using a computer. This is the holy grail of materials science. Without it, there is as much artistry as science. New materials are developed by experiment, literally creating and testing hundreds or thousands of variations in composition and processing in order to find the optimal values.

In the last five years the so-called *multiscale modeling* has been much publicized as the key to modeling materials realistically. In its broadest sense it simply means any method of modeling a system which explicitly takes into account the disparate length scales ranging from the atomic (or even electronic) at one end to the macroscopic/engineering length scale at the other end. Explicitly taking the different length scales into account generally involves the use of different descriptions of the material at different length scales. This for example means the positions and velocities of atoms at the atomistic scale, the positions and configurations of point, line and surface defects at the mesoscopic scale, and continuum fields at the macro-scale. Fracture is the archetypal multiscale process, since the defining events, namely the separation of atoms (“bond-breaking”) are atomic-scale phenomena, while the source of energy driving these events is large scale elastic deformation.

Two paradigms of multiscale modeling have emerged. The first, called by some

the *Coupling of Length Scales* refers to simulations which use different simulation techniques simultaneously to model different parts of the system in a hierarchical manner. For example a simulation of fracture involving tight-binding (quantum-mechanical) molecular dynamics for the atoms closest to the crack tip, classical potential molecular dynamics for a much larger layer of atoms surrounding these, and continuum mechanics (involving finite element calculations) to deal with the outermost and largest part of the system. The second kind of modeling has been referred to as *parameter-passing sequential modeling* (see *MRS Bulletin, March 2001, Editorial*), or more frequently as *the transfer of information across length scales*. In this case a simulation only involves one length scale and one description of the material at a time. The output of one simulation is used to give parameters for another simulation at a larger length scale. Given a simulation of a material at some relatively small scale, the “micro-scale”, what information is needed for a complete description at the larger length scale (which we shall call the meso-scale—although either or both scales might be termed micro- or meso- in a different context)? There are two aspects to consider: choosing a description at the meso-scale, for example a continuum description in terms of fields and defects, and devising rules for the evolution of these quantities—field equations (PDEs), or equations of motion for defects (ODEs). These rules should contain as completely and as accurately as possible the physics from the micro-scale, to be *extracted* somehow from micro-scale simulations. In many cases what it comes down to is choosing a functional form to fit data from the lower scale simulation (in some cases the data come from experiment) for use in the upper scale simulation.

1.2 Parameter Passing: small scales to large

Matter, upon sufficiently close examination, is seen to consist of atoms. In metals these are not arranged randomly, but for the most part are arranged in a regular lattice, or crystal. Atoms in a crystal² do not give rise to particularly complicated macroscopic behavior; it is not hard to tune an empirical law for the atomic interactions (usually known as an *interatomic potential*) to yield the correct (i.e. experimental) density/lattice constant, first order (linear) elastic constants, etc. at least at zero temperature. It takes a little more work to get correct temperature dependence, nonlinear elastic constants, phonon dispersion curves etc. However these quantities are all readily available from experiment and it is straightforward to construct continuum theories from them. For atoms in a crystal, the atomic description offers little advantage. This is particularly the case given that from a macroscopic point of view there are way too many atoms for practical simulation.

As we emphasized in the previous section, it is the defects of a crystal, of various kinds, that tend to dominate material deformation, fatigue and failure beyond linear elasticity. This is the case even though the number density of defects is much smaller than that of atoms. It seems that we need an atomic description to satisfactorily describe defects—since their very definition is terms of atoms—but given that most of the atoms in a material are in regions of homogeneous or nearly homogeneous deformation relative to a perfect crystal, we would still seem to be wasting a lot of computational effort in simulating them. There are ways around this conundrum. One is to retain an atomic description close to defects, and use a continuum description far away. This is the first type of multiscale modeling mentioned above, which will be discussed further in the next section.

²assuming the crystal to be large so that surface effects are small

The other type of multiscale modeling is based on idea that the various defects can be represented as various lower-dimensional entities which can be located within the continuum and assigned their own dynamics: rules of formation, motion, interaction, destruction, etc. An important point to which we shall return again and again in this thesis, is that the rules must include dependence on all possible parameters. This includes the geometry of the defect as well as background fields like stress/strain, temperature, electric field, impurity concentration etc. Such information is carried *implicitly* by the atoms in the atomistic description, but we have to make it *explicit* in the continuum description. Thus we make a transition or paradigm shift from a defect being an *emergent* property from an atomistic description to being a *primary entity* in a continuum simulation. There are several common features of this transition:

1. Though we are reducing the description by leaving out atoms, the description of a continuum entity is usually more complicated geometrically necessitating functions of many variables. For example, diffusion along a grain boundary involves three functions (three diffusion constants in 2D) of five variables (5 parameters needed to specify a grain boundary).
2. The continuum description is mathematically more precise, requiring for example, one to define the *exact* location of the vacancy, dislocation or crack tip, which can only be obviously identified to within a lattice constant in the atomistic description. Subtleties emerge due to this ambiguity which require care in the construction of rules.
3. There is generally some *singularity* in the continuum description, usually associated with the basic physics of the defect.

4. Efficient *functional forms* are needed to represent rules. These functional forms are fit to data from atomistic simulations. By efficient is meant that the form has a minimum number of fitting parameters, capable of being determined with a minimum number atomistic calculations. Often, efficiency is achieved by building the form of the singularities into the function.
5. One needs to pay attention to possible *new physics* associated with the singular points of the continuum description.

1.2.1 Examples: some specific defects

1. *Vacancies.* A vacancy is a missing atom. In an atomistic simulation its presence can be inferred by identifying a set of surrounding atoms as bounding a “perfect” subset of the crystal and comparing the actual number of atoms within the boundary with the number expected from the crystallography. At not too high temperatures this can be done quite close to the vacancy—almost one unit cell, thus allowing the location to be specified to within a lattice constant. This leads to the continuum description of a vacancy as a point object. Atomistic calculations are needed then to determine the rate of formation of vacancies³, the energy barrier (which gives the rate of diffusion of the vacancy), the elastic fields (which determine interactions with other defects such as dislocations) and the rate of annihilation with an interstitial—all as a function of external stress, temperature, etc.
2. *Dislocations.* A dislocation is defined by a nonzero Burgers vector, which is the amount by which a path around the dislocation line, in locally “good”

³actually vacancy-interstitial pairs

parts of the crystal, fails to close when translated to a perfect crystal. Again, this path can be brought quite close to the dislocation line allowing a specification of its location to within a lattice constant. Parameters of the continuum description include a rate of dislocation nucleation, various energy barriers determining the motion of the dislocation (e.g. Peierls barrier, kink formation and migration barriers), the elastic fields carried by the dislocation (which determine its interaction with other defects) and a rate of annihilation with another dislocation or with a surface—all a functions of external stress, temperature, orientation, curvature etc. In chapter 2 of this thesis we present work on 2D dislocations whose aim is to extract a accurate description of the Peierls barrier as a function of external stress. One of the features of that work is the flexibility built into the boundary conditions by use of elastic multipoles, whose purpose is to accelerate the convergence (with system size) of the measurement of this particular quantity. The ability to obtain accurate results with a minimal system enhances our ability to most efficiently extract information from one length scale and communicate it to the next.

3. *Cracks.* A crack is a curve bounding two internal free surfaces of the material. It can be parameterized as a curve along with a unit vector defined at each point on the curve specifying the local normal to the crack surface. The principal rule that is desired in a meso- or macro- scale simulation of a crack is the crack growth law. Present understanding of crack growth laws is not complete enough to be able to systematically obtain parameters from atomistic simulations. Some of the relevant parameters are known from fracture mechanics [4]: the stress intensity factor (which characterizes the

stress concentration near the crack front) or energy release rate (related to the stress intensity factor: defined in terms of how much elastic energy is released per unit growth of the crack). However these are properties of the large scale state of the crack—they characterize how much energy is available to advance the crack from far away loading. The challenge is then to know what material properties are to be combined with the loading information to make a prediction for crack growth. For a *brittle* material, the caricature at least is that atomic planes *cleave*, and all of the elastic energy released goes into the appropriate surface energy. In this idealization a knowledge of surface energies of different crystallographic orientations along with barrier heights for crack advance in different directions within a plane (lattice trapping) is in principle enough to predict the growth law; real brittle materials do not necessarily cleave perfectly, but their damage zone is small—of order atomic length scale. In a ductile material, a large amount of dislocation nucleation and motion—plastic flow—takes place, leading to very damaged crack surfaces—the damage zone might be of order microns. Also, it means that a lot of released elastic energy—most of it, in fact—goes into plastic flow and not into surface energy, making ductile materials much tougher in general than brittle ones. “Toughness” measures the energy release rate when a crack actually grows⁴. Because ductile fracture involves large-scale dislocation activity, typically spread over cubic microns or more, it is not practical to extract crack growth laws directly from atomistics; intermediate-scale simulations involving dislocations are necessary.

⁴If the loading is such that the energy release rate is lower than than the toughness, then the crack will not grow; the meaning of energy release rate is then how much elastic energy would be released by a virtual extension of the crack.

In this section I have concentrated solely on passing from atomistic descriptions to defect-based descriptions, because this thesis is mostly concerned with these two levels. However the concepts also apply to parameter passing at larger scales. In this case one seeks to get rid of even the defects and represent them by fields, such as dislocation density replacing discrete dislocations, or an orientation distribution function (also known as *texture*) replacing grain boundaries. The methodology here is much less well developed; it is still far from clear what the appropriate field description is in many cases. It also not understood how best to extract the parameters—the constitutive laws—from defect simulations to pass to purely field-based simulations.

1.3 Mixed atomistic/continuum modeling: to couple, to embed, to extend

The other side of multiscale modeling involves keeping atoms, but only some of the atoms. One maintains an atomistic description where necessary (in the vicinity of defects), and uses a coarser description elsewhere. The coarser description can be either a standard continuum modeling method such as finite elements, or a description deriving directly from coarse-graining the atomistic description. I will briefly mention examples of both.

1.3.1 Coupling to finite elements

An example of coupling of standard (time dependent) finite elements to empirical potential MD (which itself was coupled further in to tight-binding MD) is the work of Abraham et al. [2, 1, 14]. The system is a piece of silicon containing a inter-

nal crack. The central region containing the crack is treated atomistically. It is surrounded by a finite element region. The sample volume represented by finite elements is more than an order of magnitude greater than that represented by atoms, but only one quarter the number of degrees of freedom (nodes/atoms). There are two aspects to any coupling algorithm: kinematics and energetics; these can be associated with two independent approximations made in going from an atomistic model to a continuum/coarse-grained model, namely the *kinematic approximation*, by which the number of degrees of freedom is reduced, and the *energetic approximation* which is whatever approximation is made to avoid calculating the energy of the continuum/coarse-grained region in a fully atomistic manner. The kinematics are coupled in this case by making the finite element mesh refine to the atomic length scale in the vicinity of the boundary; then the nodes become coincident with atoms. Their energetic approximation is to use the linear elastic energy computed from the continuum displacement field (given by finite element nodal displacements and shape functions). The energetic coupling consists of kind of averaging of linear elastic contributions and interatomic potential contributions to finite element cells at the boundary. The most important feature of this averaging is that there is indeed a well defined Hamiltonian function of the entire system, from which forces are derived by differentiation. This importance is emphasized in Refs. [2, 1, 14].

The advantage with such coupling is that one can make use at least of existing and standard finite element techniques, and possibly of existing finite element software. Going further one could imagine making use even of existing simulations—that is having a finite element simulation adapt in real time by introducing an atomistic simulation. Our work in this context is the subject of chapter 5.

The drawback in implementations such as that of Abraham et al. is that the finite element mesh has to be designed with the atomistic simulation in mind because it needs to be refined to the atomic length scale near the boundary. This makes it somewhat difficult though not impossible to adapt a finite element simulation to make a hybrid finite-element/atomistic simulation.

1.3.2 The quasi-continuum method

The quasi-continuum method developed by Tadmor et al. [70, 69, 51] is described as being all atomistic, in that at no point is a constitutive law used to calculate energies—all energies are purely atomistic. It is derived as a coarse-graining of standard atomistics a subset of atoms, say of a crystal, is identified as being *master atoms*, whose positions are independent degrees of freedom. These atoms are taken to form the nodes of a finite element mesh, the positions of other atoms, known as *slave atoms* being given by interpolation using standard finite element shape functions. Fully atomistic regions are identified as regions where all atoms are master atoms, typically near defects. Clearly the kinematic coupling between coarse-grained and fully refined regions is of the same type as with Abraham et al, namely a refining of the mesh in the coarse-grained region to the atomic length scale, although the boundary is more of an emergent property than an explicit one in this case. In its energetics, the is atomistic everywhere. However it is still necessary to avoid computing forces on every atom in the standard MD way (force calculations, after all, are what dominate the time in standard MD). This is done by ignoring the nonlocal nature of the interatomic potential in coarse-grained regions, and assigning each element an energy equal to its volume times the energy density of an infinite crystal homogeneously strained according to whatever the

strain in that element is (this being a function of the master atoms' positions only). Thus atoms in the coarse-grained region are called “local atoms”, and those in the fully refined region “non-local atoms”. Some care is taken to match and weight contributions from boundary atoms. However it is found that the ideal lattice is not the ground state of the resulting energy functional. This is because a non-local atom near a boundary with local atoms (where “near” means “within a potential cutoff-distance of”) does not receive the same force contributions from the non-local atoms on one side of it as from local atoms on the other and thus is not balanced. Corrective forces are therefore added, known as “ghost forces” in order to balance the discrepancy and stabilize the perfect lattice; however these do not derive from a energy functional, which is the principal weakness of the method. In particular it makes dynamics (as opposed to statics—incremental loading and relaxation) impossible because energy is not conserved and standard algorithms go unstable[41].

In the quasi-continuum method there is no clear separation between coarse-grained and fully-refined regions, they are more or less intermingled. When a dislocation wants to leave the fully refined region, a refining of the mesh ahead of it, to convert coarse-grained material into refined material, must take place. Then, to avoid having trails of fully-refined material everywhere, wasting computer time, the material must be coarsened again after the dislocation has passed through, which turns out to be almost as computationally intensive as leaving it refined [19].

1.3.3 FE meets Digital Material: OFE/MD

In our implementation of a coupling between MD and finite elements described in chapter 5, we seek to overcome the problems in the above two methods by using a somewhat simpler approach. One of the motivating factors in the work is the idea that an MD simulation could be created within a previously existing continuum (finite element) simulation. This would happen when some detection mechanism locates a local hot-spot of stress or temperature or both, from which it is anticipated that new physics at the atomistic level might play a significant role in the further evolution of the material. To be specific, for testing, the continuum model is a finite element mesh with a crack. The MD simulation is to be placed in a small region on the crack front. In contrast to the other hybrid methods, the mesh is not refined to the atomic scale—it cannot be, since it existed before the MD simulation did, and we wish not to have to change it to accommodate the MD simulation. In fact, the MD region overlaps the continuum region in that this part of space is both occupied by atoms and covered by parts of finite elements. The Hamiltonian of the combined system is carefully constructed, using a *transition function* to control the weighting of the contributions to the energy density from the atomistic and continuum models. Note that in contrast to the quasi-continuum method we do have a well-defined Hamiltonian. Ghost forces are still apparent, but their effect on atomistic processes is reduced because the boundary between local and non-local energy is typically far from the region of interest, and is smoothed out by the transition function.

1.4 Object-oriented, Design Patterns-enabled molecular modeling: Digital Material

A key part of this work is the attention paid to software development. We have developed a powerful molecular dynamics (MD) package, known as Digital Material, which will be described in some detail in chapter 4. The main reason for adopting a modern software engineering approach to designing an MD package is that recently, in particular, MD has become a lot more than it once was. Traditional MD simulations, as described for example by Allen and Tildesley [3], involve mostly using periodic boundary conditions and doing time averages of various quantities to compute thermodynamic properties of solids and liquids. However, multiscale modeling requires much more than this. Coupling to continuum models requires different kinds of boundary conditions and constraints. Understanding long time dynamics involves identifying local minima of the potential, computing energy barriers, transition states and paths. For example, the method of *Temperature Accelerated Dynamics* of Voter and Sorenson [61] combines running molecular dynamics at a high temperature, identifying transitions from one local minimum to another, reflecting trajectories back into the initial well (in order to sample more barriers) and computing energy barriers between minima, in order to extrapolate the low temperature long time dynamics. Thus the method combines standard MD, minimization and techniques for finding barriers (several techniques exist; see [30]), as well as infrastructure-operations such as making copies of a system, comparing two states of a system, etc. Furthermore the barrier finding methods themselves often involve making copies, modifying forces and other non-standard manipulations. Thus in the context of materials modeling using MD or more gen-

erally, *atomistics*⁵ we see that flexibility is needed on many fronts, given the variety of:

- Ways we want to organize the atomic variables, meaning mainly the position and velocity arrays (for the purposes of having different subsets for boundary atoms or core atoms, or for making copies for the Nudged Elastic Band method, etc.)
- Types of boundary conditions (periodic, or embedded in continuum)
- Constraints that may be put on the system
- Adjustments to be made to the forces (e.g. for the Nudged Elastic Band method)
- Ways of evolving the system (accelerated rather than real time)
- Analyses that can be carried out, including visualization, computing statistical averages, stress intensity factors (for cracks).

We have always needed a variety of interatomic potentials to represent different materials. In old-style scientific programming, this was achieved by substituting in one subroutine for another, and recompiling. The variety of combinations now possible, and the need for a large degree of interoperability, mandates an object-oriented programming (OOP) approach, paying great attention to the design of the software. In the last decade the software-engineering community has realized how good OOP techniques tend to recur in different contexts. This has led to

⁵Many people take MD to mean strictly standard dynamics, which does not include for example, minimization techniques and barrier finding techniques, which are naturally all part of the same package. When I feel the need to be explicitly general I will use the term “atomistics” rather than MD.

the concept of *Design Patterns*, introduced by the book of the same name [24]. A design pattern is an abstraction of a successful OOP technique. It is a named protocol for what classes should exist, and what relationships they should have with each other in order to solve a certain class of problem, in a way that optimizes code readability, maintainability and re-usability. In the context of named patterns, a programmer can more quickly identify which one will best suit his needs for a particular situation—without the name as a handle for a previously tried and tested method, he or she may have eventually reached a similar design, but not as quickly. The naming and categorizing of successful techniques is a powerful aid.

In chapter 4 of this thesis, we describe our MD package, DigitalMaterial, that has been designed with these principles in mind. It is important to note that no performance compromise has been made; this is due to the use of vectorized array operations at the lowest level whenever possible, although a user of the code generally does not need to be aware of how this is done. Two main examples of Design patterns that have been used include the *Strategy Pattern* in the context of the *NeighborLocator* component and the *Observer* pattern in the context of measurements and analyses to be made on the system, including visualization. Others include Singleton, Factory Method, Iterator and Mediator. As well as providing a variety of interatomic potentials, boundary-conditions, “mover”-algorithms, we also have provided tools for setting up simulations, classes which fill a given shape in space with atoms occupying sites of an arbitrary lattice. We have seen codes which were fixed to only deal with one particular crystal orientation, or similar, and have found this kind of flexibility of great use when starting a new application.

1.4.1 Python drives C++ creates power with control

While the use of object-oriented languages such as C++ has been growing steadily in the last decade, a relatively more recent development is the recognition of the advantages offered by scripting languages, particularly Python[53]. Scientists in the past have used for example Perl to manage simulation jobs, dealing with input and output files etc. Python, like Perl, is a scripting language, but with full object-oriented capability, as well as a much more user-friendly syntax. There is a large and growing list of available modules for Python, including Numeric Python (NumPy), offering fast array and matrix operations. But by far what most makes Python useful for our purposes is the existence of David Beazley's program SWIG (Simplified Wrapper and Interface Generator) [68], which allows easy interfacing of Python to C or C++ code. In particular, classes in a C++ library, such as our Digital Material library, can be *shadowed* in Python thus allowing almost real-time control using the python interpreter. The ability to directly manipulate the C++ objects through python is of great benefit for debugging applications, because the compile cycle is avoided. Apart from this, applications which use the underlying C++ library can be quickly developed using Python.

Two other MD packages which combine Python with C/C++, that have been developed recently are SPaSM (Scalable Parallel Short-range Molecular dynamics) by Beazley and Lomdahl of Los Alamos National Lab [11]and MMTK (Molecular Modeling Toolkit) by Hinsen [33]. SPaSM was originally written in C, and hence is not object-oriented. Its primary strength is that it has been optimized for large scale parallel simulations. From what the author has seen of its source code, it does not seem to particularly flexible in terms of boundary conditions or the range of algorithms that can be applied. Wrappers for Python were subsequently

added—in fact this was the origin of SWIG, David Beazley being an author of SPaSM. MMTK had quite a different story: it was written primarily in Python, with performance critical parts being written as C extensions. It is designed in a fully object-oriented manner, and incorporates several of the design principles that Digital Material does, such as separating the code for calculating forces from the code that implements specific algorithms for dynamics or energy minimization, separating out the boundary conditions (“universes”) and using third-party libraries when possible for standard tasks (e.g. LAPACK for linear algebra and NetCDF for binary storage, both of which are used by Digital Material). The data structures in MMTK reflect the chemistry background of its author, which facility included for atoms, groups, peptides, and proteins. The interatomic potentials it has reflect this type of application, and do not include the potentials commonly used for metals or other materials (e.g. silicon) of interest to the materials modeling community.

In designing Digital Material, we have chosen to put most of the work into C++ code, providing a library that can either be used by a pure C++ application [the user provides a `main()` function], or as a module to be imported into Python [the user writes a Python script⁶]. The amount of Python code in applications in the author’s experience is between 500 and 1000 lines. The latter figure echoes a target declared at a project meeting for another materials-modeling/engineering software project [5], where the stated goal of the project was to develop a set of software tools such that a new graduate student with no prior experience could within a month, using less than 1000 lines of code, write an application that could

⁶Indeed the process of converting a C++ main function to a python script is almost trivial: the addition/removal of semi-colons, the replacement of periods by arrows or vice versa, and the altering of syntax for `for` loops are most of the work.

solve engineering problems involving complex geometry, coupled solid and fluid simulations, involving a variety of mathematical models.

As an example of the power offered by Digital Material, the author was asked by an experimentalist to run simulations of scattering of atoms by a surface. A version of the Python script which worked, to some extent, took two hours to complete. A re-implementation of the “Safari” algorithm [27] for efficient sampling of impact parameters took only another few hours. There was of course a certain amount of debugging to be done, but the limits at this point were more to do with the science than with programming; and this is exactly what we seek: to allow a researcher to concentrate on the novel aspects of a problem rather than be spending time re-implementing and debugging code which has been written many times before in a different context.

Chapter 2

Dislocation Mobility: Accurate Peierls Barriers

2.1 Introduction : mobility of dislocations

The work described in this chapter falls within the “parameter-passing” kind of multiscale modeling. It concerns the accurate calculation of dislocation mobility from atomistic simulations. Dislocation mobility has particular importance in multiscale modeling, in view of its fundamental role in plasticity, which is often the key ingredient in multiscale phenomena. Dislocation dynamics simulations[21, 56, 75] model dislocations explicitly as points in 2D or discretized curves in 3D, with rules for calculating the force on each dislocation or part of a dislocation from the local stress, rules for specifying the velocity in terms of the forces and rules for dealing with the creation, destruction and intersections of dislocations. The motion of a dislocation is governed by configurational changes near the core, and hence atomistic modeling is the natural technique to study it. The aim of such modeling is to determine velocity (or equivalently mobility, the ratio of velocity to

force) of dislocations in response to an applied stress at a particular temperature, when subject as well to other fields: electric field, impurity concentration etc.

The velocity of a dislocation moving under an applied stress field is generally assumed to depend only on the component of the stress which shears the slip plane, called the resolved shear stress (RSS). It is known that the *driving force* on a dislocation depends only on the RSS, but this does not preclude a dependence of the *mobility* on other components of the stress tensor—the distinction being one between kinetics and thermodynamics. Experimentally, the dependence of dislocation velocity on stress is typically fit with an equation of the form

$$v = v_0 \left(\frac{\tau}{\tau_0} \right)^m \quad (2.1)$$

where τ is the RSS, τ_0 is called the *critical resolved shear stress* (CRSS) and v_0 is the velocity when $\tau = \tau_0$. The fitting parameters are τ_0 , v_0 and the exponent m and these in turn depend on temperature as well as the crystallographic slip system[42]. Furthermore different values are measured for a crystal in tension than for one in compression; this is known as the tension-compression asymmetry, and is a sign of the dependence of the mobility of dislocations other parts of the stress field.

It is well known[34] that at low temperatures and stresses the dynamics of dislocation are primarily due to glide and in some elements are dominated by the existence of an energy barrier, the Peierls barrier, between neighboring configurations (some transition metals, like Cu, have absurdly small Peierls barriers). In 3D the glide process cannot happen along the whole dislocation line at once, but occurs as a result of kink-pair nucleation and kink-migration. However in 2D, where the dislocation is a point-like object, overcoming the Peierls barrier is a single step

process, and the resulting mobility is well characterized by an Arrhenius law:

$$v(\sigma, T) = b\nu \exp(-E_B(\sigma)/K_B T) \quad (2.2)$$

where b is the Burgers vector, ν is an attempt frequency, and σ is the applied stress tensor. We must calculate the energy barrier E_B as a function of σ , and present the information in a compact functional form. We have focused on dislocations in two-dimensional crystal (which are necessarily edge dislocations) so that we could develop a flexible boundary method of evaluating energy barriers in the context of a relatively simple system.

There is another aspect of modeling defect dynamics directly which has only been minimally included in existing simulations. This is the effect that the defect has on its environment. In a dislocation dynamics simulation it is important to be able to calculate the stress field due to a dislocation. To leading order this is given by the standard solution from linear elasticity, known as the Volterra solution, which gives the stress field due to a single dislocation at the origin:

$$u_x^{(V)} = \frac{b}{2\pi} \left[\tan^{-1} \frac{y}{x} + \frac{xy}{2(1-\nu)(x^2+y^2)} \right], \quad (2.3)$$

$$u_y^{(V)} = -\frac{b}{2\pi} \left[\frac{1-2\nu}{4(1-\nu)} \log(x^2+y^2) + \frac{x^2-y^2}{4(1-\nu)(x^2+y^2)} \right]. \quad (2.4)$$

However this may not be enough when two dislocations approach close to one another—the stress field close to a dislocation is not necessarily the simple leading order formula but may have higher order terms (i.e. terms which fall off as a higher power of r). In principle the complete description of a dislocation, or of any defect, when represented as a primitive object in a simulation, must include all terms of the stress field. If the stress field is expressed as a multipole series, then the

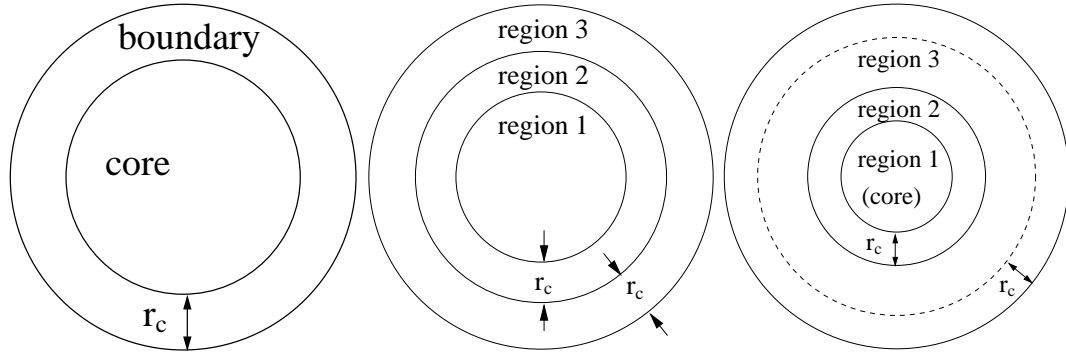


Figure 2.1: Division of atomic model into different regions when using fixed boundaries, original Flex-S scheme and extended Flex-S scheme. r_c is the cutoff distance of the potential.

coefficients of the terms in the series should be considered as extra numbers ‘carried’ by the dislocation as it moves in the simulation, in addition to the Burgers vector. This notion has influenced our choice of boundary conditions in our atomistic simulations. In embedding our atomistic model in an infinite continuum, we do indeed employ a multipole series to represent the elastic displacement field, rather than for example use a Greens function technique. The purpose is to be able to quote the values of the coefficients as properties of the dislocation (which may depend on external fields).

2.1.1 Flexible boundary conditions

The earliest work on dislocation core structure [39, 26] employed boundary conditions consisting of a surrounding layer of atoms fixed in positions determined by the anisotropic linear elastic (Volterra) displacement field of a dislocation (see the first diagram in Fig. 2.1.1). Details of the core structure turned out to depend

significantly on the size of the system and did not converge until the system was quite large. This is despite the fact that the displacements at the boundaries are accurately described by the Volterra solution even when the system size is small. For example, the Volterra solution displacements are off by 1% of a lattice constant at distances of five lattice spacings, and 0.1% at ten lattice spacings, but the computed energy barrier for Lennard-Jones potentials changes by a factor of three (partially because the true barrier is rather small).

Sinclair[59] was among the first to employ flexible boundary conditions in dislocation core-structure calculations. He started with a general solution of linear anisotropic elasticity[23] consisting of the logarithmic ($O(r^0)$) Volterra terms as well as a series of terms going like r^n where n ranges over all integers. The terms with $n > 1$ do not appear in an infinite system; the $n = 1$ term, which gives rise to a constant stress at infinity, to be interpreted as the applied stress, which we use but Sinclair did not. By including a truncated set of the negative- n terms in the formulas for the boundary displacements, and allowing the coefficients of these terms to vary as part of the minimization to find the core structure, Sinclair was able to get accurate results with a much smaller system. In Sinclair's method there are three regions, numbered I, II and III (middle diagram in Fig. 2.1.1). Region I contains unconstrained "core" atoms. Regions II and III contain constrained boundary atoms, each region having a thickness equal to the cutoff distance. The energy is taken to be the atomistic energy of the infinite crystal; forces on the boundary parameters being determined by weighted sums of the forces on all constrained atoms in the crystal. Because the constraining displacement field satisfies linear elasticity, atoms outside region II are assumed to be in equilibrium and therefore have zero force, thus the sum needs to include only region II atoms. The

purpose of region III atoms is to provide a full set of neighbors for region II atoms. Because of the approximation of zero forces outside region II; the generalized forces acting on the displacement coefficients are not derived from a well-defined energy.

Following this, Sinclair and other workers[25, 60] developed several alternative schemes for employing flexible boundaries with names such as *Flex-I*, *Flex-II* etc. The latter, Flex-II, has been more widely used[54] as it was found to be more computationally efficient. Rather than represent the continuum displacement field in terms of a general solution to the elasticity equations, Flex-II at each step in the minimization uses a Green's function technique to compute corrections to the displacement field arising from force imbalances at the coupling region (the boundary-equivalent to region II). The assumption regarding forces outside region II also features in the Green's function methods. The original method, known as *Flex-S*, has gone somewhat out of fashion for the purposes of dislocation core structure calculations.

The aim of the present work is to compute the energy barrier associated with the transition from one local minimum of energy to a neighboring one; this is the Peierls barrier. Since now we consider motion of the dislocation it is clear that at the very least, the boundaries should be flexible enough to allow for the motion of the dislocation. From the point of view of the elastic displacement field this appears as a change in the center with respect to which the distance and angle appearing in the formulas are computed. Ohsawa and Kuramoto[50] implemented a Flex-S scheme in order to measure the Peierls stress for a dislocation. They kept a fixed nominal center of the dislocation—all the variation of the boundary, including effective motion of the center was in the coefficients of the higher order

terms¹.

2.2 Boundary kinematics: Flex-S with moving center

For our Peierls barrier calculations we have employed a Flex-S type scheme. However our method differs from Ohsawa and Kuramoto or Sinclair in that we allow the center of the dislocation, that is the center with respect to which the terms in the displacement field are calculated, to move. The motivation for this was discussed above. Since the series expansion of the displacement field is a general solution without varying the center—translation of the field configuration can be represented by appropriate changes in the coefficients of the terms—this introduces two extra unnecessary parameters into the description of the continuum field. In order to have well defined parameters, we need to place two constraints on the higher order coefficients. This is an arbitrary choice, and thus quoted coefficients must always refer to this choice, although ultimate quantities, such as the mobility, should be independent of it. This is somewhat analogous to a gauge transformation in electromagnetism. In addition to the “negative- n ” terms in the general solution (those which die away with distance), we include the first positive- n term, which is the constant strain term, which couples to the applied stress at infinity. Even higher n terms could be included for the purpose of simulating a nearby surface for example. Our method also differs in the way in which the dynamics of the boundary is derived; in particular there is a well-defined energy functional which

¹Note that the coefficient of the Volterra term cannot vary, as it is determined by the Burgers vector.

blends atomistic energy from the core of the dislocation with continuum elastic energy from the far-field region. Region III still defines the boundary of atoms represented in the simulation, but region II only exists for a practical reason² and no longer has a role in defining the energetics. The boundary of atomistically counted energy is a separate boundary one cutoff distance within the boundary of region III (see the third diagram in Fig. 2.1.1).

In the following sub-sections we review the general solution to two-dimensional elasticity and describe the kinematic details of how this is used to define atomic positions. In subsequent sections we describe our atomistic potential, our continuum elasticity analysis to determine the far-field contribution to the energy, our coupling technique and details on how barriers are computed (some adjustments to the Nudged Elastic Band method are required to deal consistently with the boundary parameters.). Then we describe our method of compactly representing the barrier data for many stresses before presenting results and discussion.

2.2.1 Linear elasticity: general solution in 2D

Since we have chosen to use a simple pair potential (Lennard-Jones) in two dimensions, the lattice is triangular and the linear elastic constants are isotropic³. Sinclair and other workers modeling dislocations used potentials with anisotropic elastic constants and thus used the general solution developed by Eshelby et al.[23] for the case of linear elasticity in a three dimensional material but where there is a direction along which fields do not vary. However it is not straightforward to apply

²namely, to make dynamics of core atoms more efficient by excluding atoms beyond one potential cut-off distance.

³The triangular symmetry forces the lowest order elastic constants to be isotropic

the anisotropic formulas to the case of an isotropic material. Solutions which are different in the anisotropic case merge and become the same when elastic constants corresponding to isotropy are inserted, and thus half of the solution space is missing. Thus we use a separate solution presented by Timoshenko for the isotropic case. It has the following form⁴ (see appendix 2.A for a discussion on the relations between elastic solutions and elastic constants in 2D and 3D materials):

$$u = -\frac{1}{4(1-\nu)}y\psi + \frac{1}{2}\Phi + \psi_1 \quad (2.5)$$

$$v = -\frac{1}{4(1-\nu)}y\phi + \frac{1-2\nu}{4(1-\nu)}\Psi + \phi_1 \quad (2.6)$$

where $\phi+i\psi = f_0$ and $\phi_1+i\psi_1 = f_1$ are arbitrary analytic functions and $\Phi+i\Psi = F$ is the integral of $\phi + i\psi = f_0$. Using analyticity we can write

$$f_0(z) = \sum_{m=-\infty}^{\infty} C_m^{(0)} z^m \quad (2.7)$$

$$f_1(z) = \sum_{m=-\infty}^{\infty} C_m^{(1)} z^m \quad (2.8)$$

At this stage we will write a particular term, $C_{-1}^{(0)}$, as $\frac{b}{\pi i}$, anticipating the interpretation of b as the Burgers vector. Integrating the series we have

$$F = \frac{b}{\pi i} \log(z) + \sum_{m=-\infty, m \neq 1}^{\infty} \frac{C_m^{(0)}}{m+1} z^{m+1} + \text{const} \quad (2.9)$$

The various terms in the Laurent series of f_0 and f_1 give rise to terms in the displacement field which have corresponding r -dependence, namely r^m for integer

⁴Timoshenko presents the solution for plane stress. From a comparison of his eqns. (91) and (92) we see that the substitution $\frac{2}{1-\nu} \rightarrow \frac{2(1-\nu)}{1-2\nu}$, equivalent to $\nu \rightarrow \frac{\nu}{1-\nu}$ will convert to the plane strain case.

m . For each power m , there are four real parameters, two from the appropriate term in each Laurent series. Note in eqns. 2.5 and 2.6 that because the function f_0 appears both integrated as well as unintegrated but multiplied by y , the index of its coefficients contributing to a given power of r in the displacement field are off by one compared to those of f_1 . It is convenient to consider $m > 1$, $m = 1$, $m = 0$ and $m < 0$ separately. Terms with $n > 1$ cannot appear in an infinite crystal because the energy associated with them grows faster than the volume (the energy density goes like $r^{(m-1)^2}$). In a finite crystal they would be required in order to satisfy boundary conditions, but would be very small near the dislocation core. The $m = 1$ terms correspond to constant strain (and rotation) and are important for specifying an applied stress. $m = 0$ includes constant terms as well as logarithmic terms, such as the Volterra formula for the displacement field of a dislocation. Terms with $m < 0$ are called the *multipoles* of the elastic field. Because the displacement field of these terms dies away quickly they are generally not associated with boundary conditions at the physical boundary of the crystal but rather are determined by the structure of the dislocation—if the non-elastic core is removed then they may be considered as arising due to boundary conditions on the surface so created. The elastic energy associated with such terms is finite at infinity, but would diverge at the origin were it not cut off by the core.

We now present explicit expressions for the different powers of r in the displacement field, in terms of the Laurent or multipole series coefficients, not including $m > 1$. For $m = 1$, when we take the constant part of f_0 and the part of f_1 that is linear in z , we get as we must a linear function of position. There is no point in writing the expressions in terms of the C 's since it is much more straightforward to simply specify a symmetric constant strain tensor. For $m = 0$, taking the constant

term in f_1 clearly just gives a constant term in the displacement field which we choose so that in combination with the other $m = 0$ terms from f_0 we get the standard Volterra formula, eqns. 2.3 and 2.4; specifically, from f_0 we need the $1/z$ term, whose coefficient we have already written as $b/(\pi i)$. The two real parameters correspond to the components of the Burgers vector; by taking b to be real we make the Burgers vector lie in the x -direction. Then we have:

$$\begin{aligned} u &= -\frac{1}{4(1-\nu)}y\text{Im}\left(\frac{b}{\pi iz}\right) + \frac{1}{2}\text{Re}\left(\frac{b}{\pi i}\ln(z)\right) \\ &= \frac{b}{2\pi}\left(\theta + \frac{1}{2(1-\nu)}\frac{xy}{x^2+y^2}\right) \end{aligned} \quad (2.10)$$

$$\begin{aligned} v &= -\frac{y}{4(1-\nu)}\text{Re}\left(\frac{b}{\pi iz}\right) + \frac{1-2\nu}{4(1-\nu)}\text{Im}\left(\frac{b}{\pi i}\ln(z)\right) \\ &= -\frac{b}{2\pi}\left(\frac{1-2\nu}{4(1-\nu)}\ln(x^2+y^2) - \frac{y^2}{2(1-\nu)(x^2+y^2)}\right) \end{aligned} \quad (2.11)$$

which becomes the Volterra solution (eqns. 2.3 and 2.4) when the constant translation $\frac{-b}{8\pi(1-\nu)}$ is added to v . Now let us consider $m < 0$. Since we will no longer be concerned with $m > 0$ we will make a slight notation change, and from now on refer to terms as $1/r^n$, with $n > 0$. For the most part we will deal separately with the linear (constant strain) term because it couples to the applied stress, except in a few places where it will be necessary to use m rather than n . Using n , the indexing of the Laurent coefficients will contain minus signs and will remind us that we are talking about negative powers of r . So for the multipoles we take

$$f_0 = \frac{C_{-(n+1)}^{(0)}}{z^{n+1}} \Rightarrow F(z) = -\frac{C_{-(n+1)}^{(0)}}{nz^n} \quad \text{and} \quad f_1 = \frac{C_{-n}^{(1)}}{z^n} \quad (2.12)$$

Upon taking the real and imaginary parts and putting everything in terms of r and θ , we arrive at the following formulas:

$$\begin{aligned}
u = & -\frac{1}{4(1-\nu)} \frac{\sin(\theta)}{r^n} \left(-ReC_{-(n+1)}^{(0)} \sin((n+1)\theta) + ImC_{-(n+1)}^{(0)} \cos((n+1)\theta) \right) \\
& - \frac{1}{2nr^n} \left(ReC_{-(n+1)}^{(0)} \cos(n\theta) + ImC_{-(n+1)}^{(0)} \sin(n\theta) \right) \\
& + \frac{1}{r^n} \left(ReC_{-n}^{(1)}(-\sin(n\theta)) + ImC_{-n}^{(1)} \cos(n\theta) \right)
\end{aligned} \tag{2.13}$$

$$\begin{aligned}
v = & -\frac{1}{4(1-\nu)} \frac{\sin(\theta)}{r^n} \left(ReC_{-(n+1)}^{(0)} \cos((n+1)\theta) + ImC_{-(n+1)}^{(0)} \sin((n+1)\theta) \right) \\
& - \frac{1-2\nu}{4(1-\nu)} \frac{1}{nr^n} \left(ReC_{-(n+1)}^{(0)} - \sin(n\theta) + ImC_{-(n+1)}^{(0)} \cos(n\theta) \right) \\
& + \frac{1}{r^n} \left(ReC_{-n}^{(1)}(\cos(n\theta)) + ImC_{-n}^{(1)} \sin(n\theta) \right)
\end{aligned} \tag{2.14}$$

At this point let us change notation for the coefficients and set $d_0^{(n)} = ReC_{-(n+1)}^{(0)}$, $d_1^{(n)} = ImC_{-(n+1)}^{(0)}$, $d_2^{(n)} = ReC_{-n}^{(1)}$ and $d_3^{(n)} = ImC_{-n}^{(1)}$. We now write the general solution for negative multipoles as

$$\vec{u} = (u, v) = \sum_{n,j} d_j^{(n)} \vec{u}_j^{(n)} = \sum_n \vec{d}^{(n)} \cdot \vec{U}^{(n)} \tag{2.15}$$

where $\vec{u}_j^{(n)} = (u_j^{(n)}, v_j^{(n)})$ and

$$u_0^{(n)} = \frac{1}{r^n} \left(\frac{\sin(\theta) \sin((n+1)\theta)}{4(1-\nu)} - \frac{\cos(n\theta)}{2n} \right) \quad (2.16)$$

$$v_0^{(n)} = \frac{1}{r^n} \left(-\frac{\sin(\theta) \cos((n+1)\theta)}{4(1-\nu)} + \frac{(1-2\nu) \sin(n\theta)}{4n(1-\nu)} \right) \quad (2.17)$$

$$u_1^{(n)} = \frac{1}{r^n} \left(-\frac{\sin(\theta) \cos((n+1)\theta)}{4(1-\nu)} - \frac{\sin(n\theta)}{2n} \right) \quad (2.18)$$

$$v_1^{(n)} = \frac{1}{r^n} \left(-\frac{\sin(\theta) \sin((n+1)\theta)}{4(1-\nu)} - \frac{(1-2\nu) \cos(n\theta)}{4n(1-\nu)} \right) \quad (2.19)$$

$$u_2^{(n)} = \frac{-\sin(n\theta)}{r^n} \quad (2.20)$$

$$v_2^{(n)} = \frac{\cos(n\theta)}{r^n} \quad (2.21)$$

$$u_3^{(n)} = \frac{\cos(n\theta)}{r^n} \quad (2.22)$$

$$v_3^{(n)} = \frac{\sin(n\theta)}{r^n} \quad (2.23)$$

To illustrate the form of each terms, the four $n = 1$ fields are plotted in Fig. 2.2. The easiest to understand are the $j = 2$ and $j = 3$ fields. Here the angular dependence is simply that the direction of the displacement field vector rotates n times as the field point makes one full rotation about the origin. The difference between them is whether the displacement is tangential ($j = 2$) or radial ($j = 3$) at $\theta = 0$. The $n = 1, j = 3$ field is important because it represents a constant volumetric expansion (the integral around a circle is independent of radius).

We also use the derivatives of these expressions with respect to the field variables x, y ⁵, however we will not reproduce the expressions for the derivatives here.

⁵note that although we simplify expressions by writing them in terms of r and θ we are considering our coordinates to be Cartesian for the purposes of derivatives, etc.

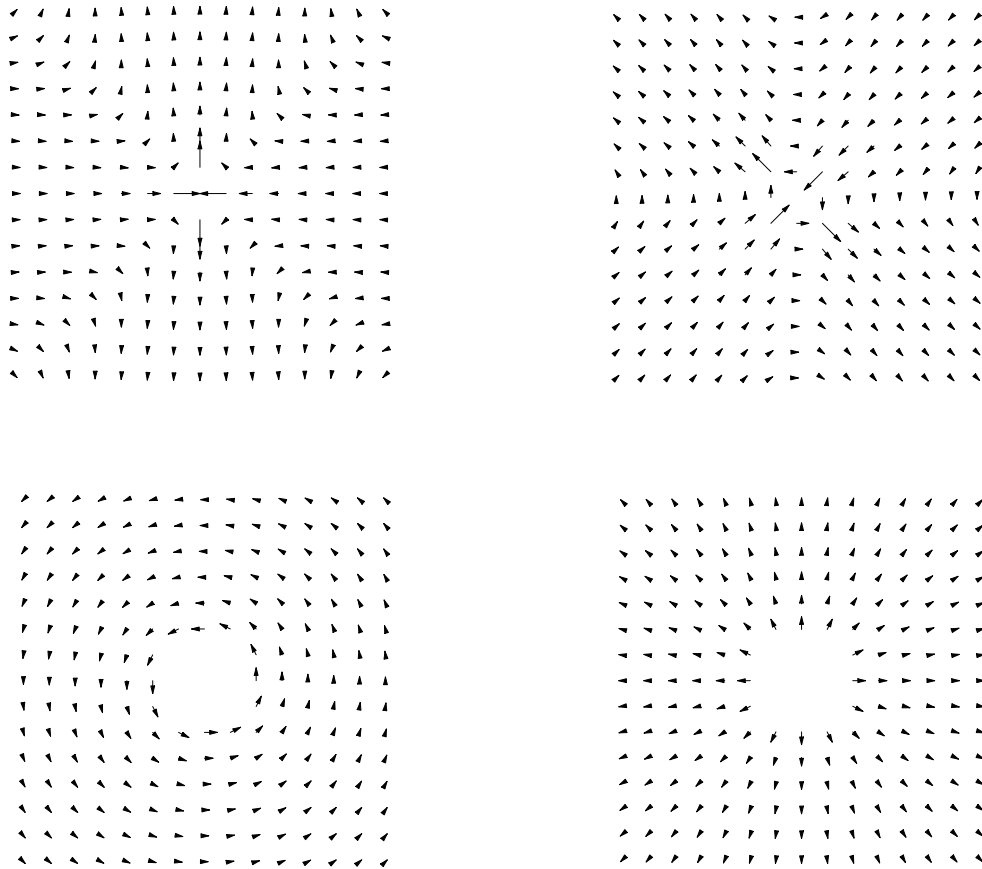


Figure 2.2: Plots of the $n = 1$ displacement fields; clockwise from top left: $j = 0, 1, 3, 2$.

2.2.2 Multipole coefficients and center as boundary degrees of freedom

We wish to define the degrees of freedom for the boundary atoms as the center of the dislocation (2 parameters) as well as the coefficients of the multipole terms up to some order N ($4N$ parameters). However as mentioned above, if we allow the center to move there are too many parameters in the solution. The general solution is complete with respect to any center, so simply allowing the center to move will not gain flexibility but rather destroy uniqueness (and thus cause numerical problems). The number of parameters must be conserved. We can see the relationship between the motion of the center and changes in multipole coefficients by taking the general solution referred to a center $\vec{c} = (c_1, c_2)$ and Taylor expanding about the origin. In eqn. (2.15) we introduced the notation $\vec{d}^{(n)} = (d_0^{(n)}, d_1^{(n)}, d_2^{(n)}, d_3^{(n)})$ and $\vec{U}^{(n)} = (\vec{u}_0^{(n)}, \vec{u}_1^{(n)}, \vec{u}_2^{(n)}, \vec{u}_3^{(n)})$; we will use this now for convenience. We consider only negative (and zero⁶) multipoles indexed with nonnegative n .

$$\vec{u}(\vec{x} - \vec{c}) = \sum_{n=0}^{\infty} \vec{d}^{(n)} \cdot \vec{U}^{(n)}(\vec{x} - \vec{c}) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \vec{d}^{(n)} \cdot \frac{(-1)^m}{m} (\partial_i c_i)^m \vec{U}^{(n)} \quad (2.24)$$

($\partial_i = \partial/\partial x_i$). Consider differentiating $\vec{U}^{(n)}$ in the direction \vec{c} . This term goes like $1/r^n$, so its derivative, and hence the first order correction term in the Taylor series, goes like $1/r^{n+1}$. Moreover, this correction is also a solution of the equations of elasticity⁷. But we know the four “basis functions” for solutions that go like

⁶The meaning of $\vec{u}_j^{(0)}$ and $d_j^{(0)}$ will be clarified below.

⁷The sum of the Taylor series corrections is the difference between two solutions, and so is a solution, and the division of a solution into terms with different powers of r is unique, so each term is itself a solution.

$1/r^{n+1}$, they constitute the term $\vec{U}^{(n+1)}$. Thus we can write the derivative along \vec{c} as a linear combination of components of $\vec{U}^{(n+1)}$:

$$(c_i \partial_i) \vec{U}^{(n)} = M(\vec{c}, n) \cdot \vec{U}^{(n+1)} \quad (2.25)$$

$M(n)$ is a matrix that is determined by a straightforward analysis. For $n > 0$ we have

$$M(\vec{c}, n) = \begin{pmatrix} c_x(n+1) & c_y(n+1) & \frac{c_y}{4(1-\nu)} & 0 \\ -c_y(n+1) & c_x(n+1) & 0 & \frac{c_y}{4(1-\nu)} \\ 0 & 0 & c_x n & c_y n \\ 0 & 0 & -c_y n & c_x n \end{pmatrix} \quad (2.26)$$

The $n = 0$ (Volterra) term is fit into this scheme by defining $\vec{U}^{(0)} = (\vec{u}^{(V)}, \vec{0}, \vec{0}, \vec{0})$ and $d^{(0)} = (1, 0, 0, 0)$. Identifying the derivatives of $\vec{U}^{(0)}$ among the $\vec{U}^{(1)}$ terms leads to $M(0)$.

$$M(\vec{c}, 0) = \begin{pmatrix} c_y \frac{b}{\pi} & -c_x \frac{b}{\pi} & 0 & \frac{-c_y b}{4\pi(1-\nu)} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.27)$$

Repeated derivatives give a chain of matrix products acting finally on the appropriately “down-shifted” \vec{U} . Now we rewrite the Taylor series:

$$\vec{u}(\vec{x} - \vec{c}) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} d^{(n)} \frac{(-1)^m}{m} M(\vec{c}, n) \dots M(\vec{c}, n+m-1) \vec{U}^{(n+m)} \quad (2.28)$$

We wish to identify the transformation of coefficients $d^{(n)}$ that corresponds shifting the center from \vec{c} back to the origin. We re-sum the series, defining $p =$

$n + m$. This gives

$$\vec{u}(\vec{x} - \vec{c}) = \sum_{p=0}^{\infty} \sum_{m=0}^p \frac{(-1)^m}{m} \vec{d}^{(p-m)} M(\vec{c}, p-m) \dots M(\vec{c}, p-1) \vec{U}^{(p)} \quad (2.29)$$

which implies

$$\begin{aligned} \vec{d}^{(p)} &\longrightarrow \vec{d}^{(p)'} = \sum_{m=0}^p \frac{(-1)^m}{m} \vec{d}^{(p-m)} \cdot M(\vec{c}, p-m) \dots M(\vec{c}, p-1) \\ &= \vec{d}^{(p)} + \sum_{m=1}^p \frac{(-1)^m}{m} \vec{d}^{(p-m)} \cdot M(\vec{c}, p-m) \dots M(\vec{c}, p-1) \end{aligned} \quad (2.30)$$

Now, back to the question of allowing the center to move: to compensate for the introduction of extra parameters we must take two parameters away. The fields that can cancel (to lowest order) a motion of the dislocation center are $\vec{u}_1^{(1)}$ for motion in the x -direction and the combination $\vec{u}_0^{(1)} - \vec{u}_3^{(1)}/4(1-\nu)$ for motion in the y -direction. These are set to zero—what this means in practice is that we have two parameters associated with $n = 1$: one is the coefficient $d_2^{(1)}$, multiplying $\vec{u}_2^{(1)}$, the other multiplies the “orthogonal”⁸ combination $(1/4(1-\nu))\vec{u}_0^{(1)} + \vec{u}_3^{(1)}$. With this prescription we can now *define* the “center of the dislocation” from the continuum point of view, as the value of the center-parameters when a minimization has taken place. This suggests a way to extract dislocation positions from general atomistic simulations: having identified by a coarse method a region in which the dislocation is known to be located, a copy of this region could be made, with a circular boundary. Then the outer layers could be constrained up to boundary degrees of freedom as in the present work, and the inner atoms fixed completely.

⁸The $n = 1$ multipoles are indeed orthogonal to each other the way trigonometric functions are (i.e., in terms of integrating their products from 0 to 2π) but in general the different terms are not orthogonal.

Minimization with respect to the boundary parameters would yield a value of the dislocation center. It must be emphasized that this definition depends on the choice we made in fixing the two $n = 1$ parameters associated with motion of the center. We fixed them to be zero, but we could easily have fixed them to some non-zero values. This is similar to fixing the gauge in electromagnetism. The minimized position of the center and the minimized values of the other parameters, are gauge-dependent quantities—but the Peierls barrier is not.

A final note about the multipoles: They can be classified according to whether they are even or odd under reflection in the y -axis⁹. Reflection in the y -axis means changing the sign of the x -component of either position or field, and leaving the y -component unchanged. To be even means that the actual field at the reflected point is the reflection of field of the original point; to be odd means that it is the negative of the reflection of the field. For even n , the odd fields are $j = 0$ and $j = 3$; for odd n , the odd fields are $j = 1$ and $j = 2$. In the infinite periodic system odd multipoles vanish, because the dislocation core structure preserves the reflection symmetry of the lattice; in the finite simulation however, whenever the dislocation is not at the center of the system we should expect to need odd multipoles due to the asymmetry between the position of the boundary with respect to the position of the dislocation. This is discussed further in section 2.5.1.

2.3 Energetics

We have now described the kinematics of our model, but this is just a skeleton—let us put flesh on it and make it move. We derive the dynamics of the model from a

⁹Actually, the x -axis in the simulation, since we have chosen the glide direction to be parallel to the y -axis.

Hamiltonian, in particular from a potential energy functional. This functional has the form of an interatomic potential in the central region and a continuum energy density in the far field region (all the way to infinity). The transition from one region to the other is managed through the use of a smoothing function (called the transition function).

2.3.1 Atomistic energy: “cut-Lennard-Jones” interatomic potential

Since our immediate concern is not to simulate a particular material, but rather to learn how to best compute certain material properties, we choose not to use realistic metal potentials, opting instead to use a simple pair potential, namely Lennard-Jones with a cutoff: “cut-Lennard-Jones” or CLJ, for short. The form of the cutoff was devised by Chen[18]. It differs from other truncated Lennard-Jones potentials in that the cutoff, 2.7σ , includes more than nearest neighbors. In 2D it includes up to third neighbors of the ground state lattice (in 3D it includes up to sixth!). The usual form of the Lennard-Jones potential is (in dimensionless form¹⁰).

$$V_{LJ}(r) = 4 \left(\frac{1}{r^{12}} - \frac{1}{r^6} \right) \quad (2.31)$$

where r is the distance between two atoms. An overall cutoff r_{c2} is chosen to lie between the third and fourth neighbor distances. To implement the cutoff smoothly—meaning continuous, with continuous first derivative, so that both energy and forces are continuous—we introduce a second cutoff r_{c1} within the main

¹⁰That is, all lengths are understood to be expressed in units of a length scale σ and all energies in units of a scale ϵ .

Table 2.1: Cut-Lennard-Jones parameters.

r_{c1}	2.41308778858241
r_{c2}	2.7
α	0.01257815434637538
β	-0.19971848731103492
γ	0.05134165513433732
δ	-0.0035213755236171

one. The form of the adjusted potential $V_{CLJ}(r)$ is

$$V_{CLJ}(r) = \begin{cases} V_{LJ}(r) + \alpha & r \leq r_{c1} \\ P_{CLJ}(r) & r_{c1} \leq r \leq r_{c2} \\ 0 & r \geq r_{c2} \end{cases} \quad (2.32)$$

where $P_{CLJ}(r)$ is a quartic polynomial:

$$P_{CLJ}(r) = \beta + \gamma r^2 + \delta r^4 \quad (2.33)$$

Having only even powers of r aids efficiency by avoiding the need for square roots.

The parameters $r_{c1}, r_{c2}, \alpha, \beta, \gamma, \delta$ are given¹¹ in Table 2.1.

For the purposes of embedding within a continuum model we need elastic constants, so that those of the continuum can be chosen to match. For a simple lattice (one atom per unit cell) these can be found analytically by differentiating

¹¹Some may question the appropriateness of quoting so many decimal places given that they are meaningless in terms of representing real materials. The parameters have been chosen to ensure that the derivatives of the potential, i.e., the forces, are continuous to machine precision. Not including all decimals may cause numerical problems.

the energy of a lattice with respect to a homogeneous strain. These are shown in Table 2.2. Because the lattice is hexagonal the constants for linear elasticity (called the quadratic constants because they are associated with quadratic terms in the free energy expansion) are isotropic¹², and therefore can be written in terms of the Lamé constants λ and μ . Furthermore for a central force (pair) potential, $\lambda = \mu$ (Cauchy relation). Since we include third order (cubic) nonlinear terms in the continuum analysis below, we also require the cubic elastic constants. There can be up to 10 independent components in a general two-dimensional material¹³, but the symmetry of the triangular lattice reduces that number to 4: which we take to be the following quantities: $\alpha \equiv C_{111}, \beta \equiv C_{112}, \gamma \equiv C_{133}, \delta \equiv C_{111} - C_{222}$. $C_{111}, C_{222}, C_{112}, C_{133}$ ¹⁴. The equality of C_{112} and C_{133} is presumably a Cauchy-like relation¹⁵.

For comparison, and to use a potential which is already in the literature, we have also used in our simulations the cut-Lennard-Jones devised by Holian et al. [36], whose cutoff only includes nearest neighbors. We quote the elastic constants for that potential as well in Table 2.2.

¹²Shown by requiring the elastic constant tensor to be invariant under a six-fold rotation.

¹³ $C_{111}, \dots, C_{IJK}, \dots, C_{333}$, where $I < J < K$.

¹⁴The other C_{IJK} are obtained from $C_{113} = C_{123} = C_{223} = C_{333} = 0$ and $C_{111} - C_{222} = C_{122} - C_{112} = C_{233} - C_{133} = \delta$.

¹⁵Note that the nonlinear constants are mostly negative; this is natural since at large positive strain (say xx) atoms are getting pulled far apart, and the force curve is beginning to flatten out, whereas at large negative strain, atoms are being pushed together and seeing increasing hard core repulsion. Thus, from this asymmetry, a third order elastic constant is allowed, and moreover should have a negative sign.

Table 2.2: Quadratic and cubic elastic constants for our cut-Lennard-Jones potential and for that of Holian et al.

	our CLJ	CLJ of Holian et al.
a	1.113201303556460427	$2^{1/6}$
λ	27.833155072804617	24.745133465974831
μ	27.833155072804617	24.745133465974831
$C_{111}(\alpha)$	-1842.81	-1883.43
$C_{222}(\alpha - \delta)$	-1497.17	-1540.99
$C_{112}(\beta)$	-161.182	-171.221
$C_{133}(\gamma)$	-161.182	-171.221

2.3.2 Continuum energy: linear analysis

The continuum energy functional must be the continuum limit of the interatomic potential energy. The goal is to obtain expressions for the total (i.e., integrated out to infinity) continuum energy in terms of the boundary parameters, which can then be used in the simulation. By including at least the lowest order part of the continuum energy we can accelerate the convergence of the energy barrier (or any other quantity) with system size. To make the integrals doable, we use the formulas from section 2.2.2 to transform the center to the origin (the transition function, to be introduced below, and which plays a role in the integration, is centered on the origin of the system), so the task is to compute the energy outside a given (undeformed) radius R as a function of the applied strain and multipole coefficients. Ignoring the ground state energy, the lowest order term is the linear elastic energy, quadratic in infinitesimal strain. This is isotropic for a hexagonal

lattice:

$$\mathcal{F}(x) = \mu \epsilon_{ij} \epsilon_{ij} + \frac{1}{2} \lambda (\epsilon_{ii})^2 \quad (2.34)$$

where $\epsilon_{ij} = \frac{1}{2}(\partial u_j / \partial X_i + \partial u_i / \partial X_j)$ is the infinitesimal strain tensor (the finite strain tensor, which includes the “geometrical nonlinearity” term, is defined in eqn. 2.35); X_i are undeformed coordinates. The Lamé constants λ and μ are chosen to match the interatomic potential being used.

Let us consider the expected contributions to the elastic energy from the different order terms in the multipole series, within the context of linear elasticity (below we consider nonlinear issues, such as the geometrical nonlinearity and the distinction between deformed and undeformed coordinates). At first sight we have a problem in that the lowest order terms, the logarithmic and linear (constant strain) ones, give energies which are infinite when integrated over all of space: the constant strain term gives a uniform energy density and hence a quadratic divergence, while the logarithmic term gives a $1/r$ energy density and a logarithmic divergence. However these are constant as far as the parameters of the model go: the coefficients of the linear terms are fixed, and the position of the dislocation only affects higher order terms—the size of the logarithmic divergence is controlled by the Burgers vector which is also fixed. In table 2.3 we show how the various combinations of multipole terms contribute to the energy. $E(R)$ represents the energy contained within a radius R , for diverging terms, or integral from radius R to infinity for the converging terms (the point is to show which terms are important, not to express precise results). In the bottom table, the pairs (m_1, m_2) , etc. represent combinations from r^{m_1} and r^{m_2} displacement fields¹⁶. $m_1 = m_2$

¹⁶We use the index m here because both positive and negative powers of r are

Table 2.3: Displacement, strain, energy density and total energy within a radius R for different multipoles and pairings of multipoles—linear analysis.

Field	$m = +1$	$m = 0(\log)$	$m = -1$	$m = -2$
u	r	$\log(r)$	$1/r$	$1/r^2$
σ, ϵ	r^0	$1/r$	$1/r^2$	$1/r^3$

Terms	(+1,+1)	(+1,0)	(+1,-1)	(0,0)	(0,-1)	(0,-2)	(-1,-1)	(-1,-2)
$\mathcal{F}(r)$	r^0	$1/r$	$1/r^2$	$1/r^2$	$1/r^3$	$1/r^4$	$1/r^4$	$1/r^5$
$E(R)$	R^2	R	$\log(R)$	$\log(R)$	$1/R$	$1/R^2$	$1/R^2$	$1/R^3$

represents a self-energy; $m_1 \neq m_2$ represents a cross term. As explained above, the (+1, +1), (0, 0) and (+1, 0) terms are constant, so we do not need to compute them. There is another apparent divergence, coming from the cross terms between the Volterra ($m = 0$) term and the applied strain ($m = +1$) terms. This turns out to give zero when the angular integration is done, although these terms are important for calculating the work done by external sources when the coefficients change. See section 2.3.5.

Thus the most important terms in the continuum energy are the (0, -1) cross terms between the Volterra term and the first multipole ($m = -1$ or $n = 1$). It is expected to be linear in the coefficients $d_j^{(1)}$ and the Burgers vector, and give a contribution to the total energy of order $1/R^2$. Having identified this candidate as the most important, let now consider the effects of non-linearities.

involved

2.3.3 Continuum energy: nonlinear analysis

What is wrong with a linear analysis of the elastic energy? The most obvious problem is that when one tries to calculate the energy contributions in a straightforward way, one gets zero for the terms that are of interest. This comes because the energy density terms have a trigonometric angular dependence that gives zero upon integrating around a circle. A nonlinear analysis would distinguish the deformed and undeformed geometries. Generally the nonlinear elastic energy density is formulated in undeformed coordinates, so the region of integration should be the undeformed body. In a system with a dislocation with the cut at -90° to the Burgers vector as ours is, the undeformed geometry has a gap of width one Burgers vector, corresponding to the material that was removed. Angular integration should start on one side of this gap, at $\Theta \sim -\pi/2 + b/(2r)$, and go around the body to the far side of the gap, at $\Theta \sim 3\pi/2 - b/(2r)$. This gives a non-zero integral, but also reduces the over contribution from that term by a power of r .

However, once we start down the path of nonlinearity, we must take care to be consistent, meaning that we must make sure to include all terms contributing to a given power of r . Thus we do a full nonlinear analysis of the continuum energy, but at each stage keep only terms up to a certain order in small quantities¹⁷. Many terms are generated during, for example, the transformation from the Eulerian displacement field to the Lagrangian displacement field, and it is very important to truncate those whose r -dependence is such that they will not appear in the energy at the order we seek, otherwise the number of terms can become too large even for a symbolic computation tool such as Mathematica. The question is then:

¹⁷This is not inconsistent with the fact that our degrees of freedom are the solutions of *linear* elasticity—they can still be considered as trial functions for a different PDE.

what are the small quantities? Since we are calculating the far-field energy, we may take b/r to be a small parameter. The upper limit on this quantity is b/Λ_1 , which is typically of order 0.1–0.15. The other small quantity that appears in the expressions is the applied strain ϵ , that is, the constant strain tensor associated with the applied stress. This also tends to be at most of order 0.1. Thus we may treat $1/r$ and ϵ on equal footing in the expansions, although in practice we set an independent limit on the power of ϵ that may appear in any term.

The nonlinear elastic energy density is a function of the finite strain tensor (see for example Ref. [72]), which is defined in terms of undeformed coordinates¹⁸:

$$\mathbf{D}_{ij} = \frac{1}{2} \left(\frac{\partial u_j}{\partial X_i} + \frac{\partial u_i}{\partial X_j} + \frac{\partial u_k}{\partial X_i} \frac{\partial u_k}{\partial X_j} \right) \quad (2.35)$$

Thus to calculate it, we must transform our formula for the displacement field of the boundary region, which are in Eulerian/deformed coordinates, into a formula. See appendix 2.C for the details of this procedure. By analyticity the energy density can be written as a power series in \mathbf{D} , of which the quadratic term is the lowest one. Since we have third order constants for our interatomic potentials we also include the third order terms in the continuum energy density—in fact, this is necessary to be consistent because terms involving third-order elastic constants appear in the final expressions at same order as those involving second order elastic constants. A priori we expect such terms simply because the lowest order part of the strain, coming from the Volterra field, should contribute a $1/R^3$ term to the energy density; however, it is not clear a priori that we should expect terms which depend on the boundary parameters (recall that we only need to worry about terms

¹⁸We use small letters x, y, r, θ for Eulerian/deformed coordinate, and capital letters X, Y, R, Θ for Lagrangian/undeformed coordinates.

in the final expression which depend on the boundary parameters). The energy density we use is

$$\mathcal{F}(x) = \mu \mathbf{D}_{ij} \mathbf{D}_{ij} + \frac{1}{2} \lambda (\mathbf{D}_{ii})^2 + \frac{1}{3!} C_{ijklmn} \mathbf{D}_{ij} \mathbf{D}_{kl} \mathbf{D}_{mn} \quad (2.36)$$

After carrying out the analysis using Mathematica, despite going through stages involving expressions with hundreds of terms, we arrive at the relatively simple collection of terms

$$\begin{aligned} E_{\text{cont}} = \int dR (1 - T(R)) & \left(\frac{b^2 (\lambda + \mu) d_0^{(1)}}{9\pi R^3} - \frac{b^2 (\alpha + \gamma) d_0^{(1)}}{108\pi R^3} \right. \\ & + \frac{b}{3R^2} ((\lambda(\epsilon_{xx} + \epsilon_{yy}) + \mu(\epsilon_{xx} + 3\epsilon_{yy})) d_0^{(1)} \\ & \left. + 4\epsilon_{xy} \mu d_1^{(1)} + 12\epsilon_{xy} \mu d_2^{(1)} - 6(\epsilon_{xx} - \epsilon_{yy}) \mu d_3^{(1)}) \right) \quad (2.37) \end{aligned}$$

There is something else we are leaving out of the continuum description: due to the non-locality of the interatomic potential, associated with a length scale of the lattice constant a , we expect terms involving gradients of \mathbf{D} to also appear in the elastic energy (a strictly local energy functional is an integral of the density only, not its derivatives). Such terms might be considered in the future.

2.3.4 Blending of atomistic and continuum energy functionals

The basis of our technique for embedding an atomistic simulation in a continuum is a function called the “transition function”. We have defined an atomistic energy functional and a continuum energy functional. We need to combine these to make a “blended functional”. Within an inner radius Λ_1 we wish to only count atomistic energy, while outside an outer radius Λ_2 we wish to only count continuum

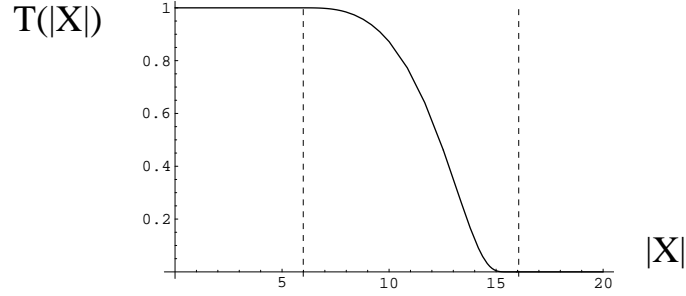


Figure 2.3: Transition function with limits $|\vec{X}| = 6$ and $|\vec{X}| = 16$.

energy. To blend them, we write down formally an atomistic energy density $\mathcal{G}(X)$ consisting of a sum of delta functions centered on the atoms. Next we introduce a function $T(X)$, called the transition function, which is unity within the inner radius Λ_1 , zero outside the outer radius Λ_2 and between them goes smoothly from unity to zero. The blended energy is then defined as follows

$$E = \int_{space} \mathcal{H}(\vec{X}) = \int_{space} (1 - T(\vec{X}))\mathcal{F}(\vec{X}) + T(\vec{X})\mathcal{G}(\vec{X})d^2X \quad (2.38)$$

This blending removes to a large extent the ambiguity inherent in defining a mathematical boundary to a region of discrete points. The transition function we use is

$$T(\vec{X}) = \begin{cases} 1 & \text{if } |\vec{X}| < 0 \\ \exp\left(\frac{1}{1-x^{-3}}\right) & \text{if } 0 < |\vec{X}| < 1 \\ 0 & \text{if } |\vec{X}| > 1 \end{cases} \quad (2.39)$$

where $x = (|\vec{X}| - \Lambda_1)/(\Lambda_2 - \Lambda_1)$. This is plotted in figure (2.3). There are two restrictions on the choice of the boundary radii. (1) Λ_1 must be at least the radius of region 2 in order that $T(X)$ be unity throughout region 1 and region 2; thus

when region 1 is minimized, the forces are exactly as if there was no transition function, and standard algorithms for minimization can be directly applied. (2) Λ_2 must be at least one cutoff distance less than the outermost boundary (the boundary of the atoms actually in the simulation); equivalently we could say that the atoms in the simulation must extend this distance beyond Λ_2 . This is necessary to ensure that all atoms with nonzero transition value have a full complement of neighbors.

We evaluate the transition function at *undeformed* positions \vec{X} , this being easier to implement than evaluating it at current positions¹⁹. The dislocation-cut is along a line perpendicular to the Burgers vector; thus the discontinuity along the cut is perpendicular to the radial direction and hence there will be only a small discontinuity in transition values in the dislocated crystal. The transition values must be assigned after the dislocation is introduced, because this process removes atoms and changes their numbers. Thus it is necessary to compute the undeformed position by evaluating the displacement fields in the current positions, yielding the original lattice positions.

In computing the continuum contribution to the energy we need expressions which have been integrated (analytically) with respect to Θ but left as a function of R since that integration depends on the Λ_1 and Λ_2 which are not known until run-time. The Θ integration reduces the entire energy expression to a sum of terms involving powers of $1/R^n$. At run-time the R -integration from Λ_1 to Λ_2 (where the energy density is multiplied by $(1 - T)$) is done numerically, and that from Λ_2 to

¹⁹One consequence of the latter would be to add terms in the atomistic and elastic forces involving the gradient of the transition function—these should cancel each other out, but would not cancel completely. Evaluating at undeformed positions means that the transition values on the atoms need only be evaluated once.

infinity analytically, being trivial.

2.3.5 Accounting for external stress

When considering the Peierls barrier for a dislocation in the presence of an applied shear stress we must be careful about how we define the barrier. For a system under constant pressure or stress, as opposed to one with fixed volume or size/shape, the appropriate thermodynamic energy is the Gibbs free energy. At zero temperature this is equal to the enthalpy, which is the internal energy—at zero temperature, potential energy—minus a work term, which is $-PV$ for a gas system with just a pressure, and $\sigma_{\alpha\beta}\epsilon_{\alpha\beta}V$ for say a solid system with skew-periodic boundary conditions characterized by a shear tensor $\epsilon_{\alpha\beta}$.

In the context of elasticity it is more intuitive, I find, to take a mechanistic point of view and think directly in terms of work being done on the system by external agents. This work is associated with a potential energy, and the work term that appears in the thermodynamic function is just the extra contribution to the potential energy from the external forces. It is a function of the external stress and the state of the boundary—therefore of the boundary parameters, the dislocation center and the multipole coefficients.

We have been considering an infinite system, but for the purpose of this discussion let us assume that the system is finite but large, with a circular boundary, of radius R . The applied stress that is felt within the system is due to actual traction forces applied at the boundary. In general for a finite system, the elastic solution may have extra positive multipoles, which are needed to satisfy arbitrary boundary conditions. If the applied tractions are chosen to equal the tractions due to the linear displacement ($m = 1$) term, then the amounts of positive multipoles that

appear are very small, being just what is necessary to cancel the stress remaining from the $m \leq 0$ multipoles²⁰

So for a large enough boundary radius we can consider just the work done by integrating the traction times the displacement change from each multipole term around the boundary. First assume the multipole coefficients are fixed, but the dislocation center moves. We will consider contributions from each term separately to see which actually contribute.

First the positive $m = 1$ term: this certainly contributes the most to the total energy of the system and to the displacements at the boundary, but we have centered this term on the system center independently of the dislocation position, so these displacements do not change when the dislocation moves and hence there is no contribution to the work done.

Next the $m = 0$ (Volterra) term: This has displacements which are order r^0 or $\log(r)$, thus the change in displacements is of order $1/R$ at the boundary. When multiplied by the tractions and integrated around the boundary we get a factor of R which cancels that in the denominator, so we are left with a constant. We will calculate this below.

Next, the $-n$ terms (negative multipoles). These have displacements of order $1/r^n$ and hence changes in displacement of order $1/r^{(n+1)}$. The contribution to the work is therefore $1/R^n$. Thus by taking the boundary radius to be arbitrarily large we see that we only need consider the work done by the tractions due to the

²⁰For example, the stress at $r = R$ from a negative multipole $\vec{u}^{(-n)}$ is of order $1/R^{(n+1)}$. To match the angular dependence, canceling positive multipoles $u^{(n)}$ and $u^{(n+2)}$ are needed, with coefficients of order $1/R^{2n}$ and $1/R^{2n+2}$ respectively. The total energy associated with both canceling terms is of order $1/R^{2n}$. Formulas for canceling the Volterra ($n = 0$) term are given on page 79 of Ref. [34] (note that the energy associated with the canceling term in this case is independent of R , even though the coefficient goes like $1/R^2$).

applied stress on the change in displacements associated with the Volterra term.

Let us see what this is.

Call the change in displacement at the boundary $\Delta\vec{u}$ and the applied traction \vec{t} . The work done is

$$W = \oint_{r=R} \vec{t} \cdot \Delta\vec{u} dl = \oint_{r=R} (\sigma \cdot \vec{n}) \cdot \Delta\vec{u} dl \quad (2.40)$$

$$= \sigma_{ij} \int_0^{2\pi} n_j \Delta u_i R d\theta = \sigma_{ij} \int_0^{2\pi} n_j \Delta c_\alpha \frac{\partial u_i}{\partial c_\alpha} \quad (2.41)$$

$$= \sigma_{ij} \Delta c_\alpha \int_0^{2\pi} \frac{1}{2} \left(n_j \frac{\partial u_i}{\partial c_\alpha} + n_i \frac{\partial u_j}{\partial c_\alpha} \right) \quad (2.42)$$

where we have used the fact that the applied tractions are those given by the stress from the linear term in the displacement—since this stress is constant it can be taken outside the integral—as well as the symmetry of the stress tensor. We then plug in the derivative of the Volterra displacement field, and write the normal as $\vec{n} = (\cos \theta, \sin \theta)$. Doing the integral gives the following expression:

$$W = \frac{b}{8(1-\nu)} (2\sigma_{xy}(3-4\nu)\Delta c_x + (\sigma_{yy}(1-4\nu) - \sigma_{xx}(5-4\nu))\Delta c_y) \quad (2.43)$$

Note that this expression assumes the Burgers vector to be in the x -direction. If we set $\nu = 1/4$ as appropriate to our central-force potential we get:

$$W = \frac{2b}{3} (\sigma_{xy}\Delta c_x - \sigma_{xx}\Delta c_y) \quad (2.44)$$

Consider the case of pure glide, i.e., $\Delta c_y = 0$. Then the work done is two-thirds of the Burgers vector times the shear stress times the distance moved. Since we know the glide force on a dislocation to be just the Burgers vector times the shear stress, the factor two-thirds might seem puzzling. The glide force is derived by

considering the work done *on the slip plane* by the applied stress as the dislocation moves. We have calculated work done on a circular boundary; the value is different for a rectangular boundary—for a square it is about $0.576b\sigma_{xy}\Delta c_x$. The difference between these the different boundary integrals can be accounted for by considering the change in elastic energy density (from the center moving) integrated over the area bounded by the two different curves—e.g. the area between the square and the circle (this comes from the divergence theorem in two dimensions); for details see appendix 2.D.

We must also consider the work done when multipole coefficients change. Since these terms are linear in the coefficients (obviously) the change in displacement at the boundary due to a change in the coefficient has the same R -dependence as the multipole itself. Repeating the arguments from before we see that the only contribution will come from terms which die at most as $1/R$, namely the $n = 1$ terms. In particular this includes the volume expansion term ($j = 3$), which couples directly to the pressure part of the stress. This is important because it gets large right at the saddle point. We again do the integral in 2.42 with c_α replaced by the appropriate parameters—recall that the $j = 0$ term and the $j = 3$ one are tied together and governed by a single parameter. The $j = 2$ term turns out to give zero work. We get:

$$W1 = p_{0/3}^{(1)}\pi \left(\frac{\sigma_{xx}}{16(1-\nu)^2} + \frac{\sigma_{yy}(2+3\nu)}{32(4-\nu)(1-\nu)^2} + (\sigma_{xx} + \sigma_{yy}) \right) \quad (2.45)$$

To implement correct accounting of the work done by external sources, we subtract the work expressions in 2.43 and 2.45 from the potential energy of the system, recalling that work done *on* the system corresponds to a *decrease* in the potential energy of the external agents. Corresponding terms are added to the

forces on the dislocation center and the $n = 1$ multipole coefficients.

2.4 Dynamics of embedded dislocation system

We follow Ohsawa and Kuramoto [50] in implementing a dynamics for the embedded dislocation system. We are not interested in real dynamics, but need a dynamical formalism in order to implement the MDmin²¹ technique of minimization. Since we have defined an energy functional of the coupled system, forces follow straightforwardly by differentiation. This defines one side of the equations of motion. The other side contains the accelerations coupled by a mass matrix. To see what the form of the mass matrix is let us derive the equations of motion from a Lagrangian. The degrees of freedom are core positions $\{\vec{r}_i\}$ and boundary parameters $\{p_\alpha\}$. The positions of boundary atoms will be denoted by $\{\vec{r}_{i'}\}$.

$$L = \sum_i \frac{1}{2}m|\vec{v}_i|^2 + \sum_{i'} \frac{1}{2}m|\vec{v}_{i'}|^2 - V(\{\vec{r}_i\}, \{p_\alpha\}) \quad (2.46)$$

where $\vec{v}_i(t)$ is the velocity of atom $i(t)$. There is a question about how many atoms to include in the i' sum, since in principle the boundary degrees of freedom represent all atoms in the infinite system. However for practical purposes we only sum over atoms actually represented in the simulation, which are all atoms out to one potential cutoff distance past the point where the transition function vanishes. We will discuss this point more below. Next, let us call the i' sum K' (since it is a kinetic energy), and use the chain rule to rewrite it in terms of the parameters $\{p_\alpha\}$.

²¹See for example Ref. [43]. We have found it better to keep the whole velocity when it has positive dot product with the force, rather than take just the parallel component.

$$K' = \sum_{i'} \frac{1}{2} m_{i'} \frac{d\vec{r}_{i'}}{dt} \cdot \frac{\vec{r}_{i'}}{dt} = \sum_{i'} \frac{1}{2} m_{i'} \sum_{\alpha} \frac{d\vec{r}_{i'}}{dp_{\alpha}} \frac{dp_{\alpha}}{dt} \sum_{\beta} \frac{d\vec{r}_{i'}}{dp_{\beta}} \frac{dp_{\beta}}{dt} = \frac{1}{2} \sum_{\alpha, \beta} M_{\alpha\beta} \frac{dp_{\alpha}}{dt} \frac{dp_{\beta}}{dt} \quad (2.47)$$

where the time-dependent mass matrix $M_{\alpha\beta}$, which is symmetric and positive definite, is given by

$$M_{\alpha\beta} = \sum_{i'} m_{i'} \frac{d\vec{r}_{i'}}{dp_{\alpha}} \frac{d\vec{r}_{i'}}{dp_{\beta}} \quad (2.48)$$

If we apply the Euler-Lagrange equations to the Lagrangian L , we get the usual $m_i \ddot{\vec{r}}_i = \vec{f}_i = -\partial V / \partial \vec{r}_i$. For parameter p_{α} , we get

$$\sum_{\beta} M_{\alpha\beta} \ddot{p}_{\beta} = -\frac{\partial V}{\partial p_{\alpha}} \quad (2.49)$$

To make separate force equations for the parameters we premultiply both sides of 2.49 by the inverse of the mass-matrix, yielding

$$\ddot{p}_{\alpha} = -\sum_{\beta} (M^{-1})_{\alpha\beta} \frac{\partial V}{\partial p_{\beta}} \quad (2.50)$$

Now we have a simple force equation for the $\{p_{\alpha}\}$ and we can use whatever dynamics algorithm (e.g. Verlet) we choose to evolve the coupled system. Returning to the issue of how many atoms should properly be included in the i' sum, consider the diagonal elements of the mass matrix. Roughly speaking these are the “effective masses” for the corresponding parameters:

$$m_{\text{eff},\alpha} = \sum_{i'} m_{i'} \left(\frac{\partial \vec{r}_{i'}}{\partial p_{\alpha}} \right)^2 \quad (2.51)$$

The question to consider is whether this sum converges as more and more atoms are included. For the parameters corresponding to the center of the dislocation

the derivative²² goes like $1/r$. Squaring and replacing the sum by an integral gives a logarithmic divergence in the effective mass. The coefficients of the $1/r$ multipoles also have logarithmically diverging effective masses. Higher order negative multipoles have convergent effective masses (e.g. the coefficient of a $1/r^2$ term has an effective mass that converges like $1/R^2$). For a diverging effective mass there must be some cutoff. Physically one can always find a length scale at which to cut off logarithmic divergences associated with dislocations (the energy is also logarithmically divergent), such as the distance to the nearest sample boundary or other dislocation. The value of the cutoff does not matter very much since log dependence is weak. Such a distance would be much larger than our simulation, but since we are not concerned with true dynamics—we are mostly interested in minimizing the energy—it is not necessary to have a physically realistic cutoff for the effective mass, and we simply include all simulated boundary atoms in the sum. This seems to work well²³.

²²from the Volterra term; the derivative with respect to center of the other multipoles is of lower order.

²³There is reason to understand the importance of the the effective mass cutoff better, however. We have experimented a little with making the linear displacement terms dynamic, with additional force terms to couple them to the applied stress—this is somewhat analogous to Parinello-Rahman dynamics. The effective mass in this case is very strongly divergent—like R^4 , because changing the linear displacement clearly involves moving many atoms significant distances. Even with our short effective mass cutoff, the effective masses of these terms turns out to be several orders of magnitude higher than those of the other terms (depending strongly of course on how big a system is simulated), which made their dynamics very slow. Simply replacing the diagonal components of the mass matrix with smaller values had a bad effect—convergence to the minimum was completely ruined, and the system went unstable. It would be worth understanding this better.

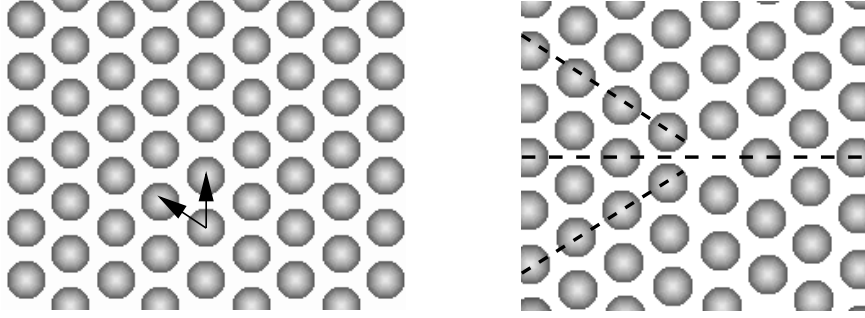


Figure 2.4: Undeformed and dislocated lattice with the lattice vectors, and lines indicating the choice of inserted half-planes, as well as the mirror plane.

2.5 Computing the energy barrier

We use the Nudged Elastic Band method [43] to compute the energy barrier. The method requires both the initial and final state to be specified, in order that a chain of states be constructed between the initial and final states and minimized in such a way that it relaxes to the minimum energy path (MEP). See Ref. [43] for the original prescription for “nudging”. Variations and improvements from the original version [31, 32] concern the way the tangent vector at each point on the chain is computed, as well as the formula for computing the spring force. Our choices for these are discussed below. First we consider the choice of the initial and final states. In this work we have chosen the orientation of the lattice such that the basis vectors are $\mathbf{a}_1 = a\hat{j}$; $\mathbf{a}_2 = a(-\frac{\sqrt{3}}{2}\hat{i} + \frac{1}{2}\hat{j})$; where \hat{i} and \hat{j} are unit vectors in the x - and y -directions respectively, and a is the lattice constant. This make the Burgers vector, and hence the glide plane, be in the \hat{j} direction. See Fig. 2.4.

2.5.1 Initial and final states

A major theme in this work is the idea of simulating an infinite system using a finite system with careful attention to boundary conditions. A dislocation moving in an infinite system is subject to the full translational symmetry of the lattice; thus the barrier between any two adjacent minima is the same—the effective potential in which the dislocation moves is something like the first diagram in Fig. 2.5, in the case of zero applied stress. We cannot, however, completely eliminate the effects of the finite boundary—if the dislocation were to glide to a position close to the edge of region I, the energy would be higher because atoms near the dislocation center would not be able to relax fully, some of them being constrained atoms. The effective potential is therefore more like the second of Fig. 2.5, where there is superimposed a background potential which is minimum at the center. Here the origin of the lattice has been chosen so that one of the minima for the dislocation position is at the center of the simulation region. Notice that there is a noticeable difference between the energy of this minimum and either adjacent one, one of which must be involved in an NEB calculation. To eliminate this effect we choose the origin of the lattice, and the initial position of the dislocation center, to be offset—half a lattice constant below the center. This location corresponds to one local minimum; another is located at a symmetric position above the center and the effective potential looks like the third diagram in Fig. 2.5. This way we minimize the distance from the center that the dislocation needs to explore in the barrier calculation.

Now, the triangular lattice has a reflection symmetry about any line parallel to the x -axis and through an atom. The dislocation core configuration should preserve this symmetry, and only terms in the elastic solution which are even

under the symmetry should appear in the far field of the infinite system. However when the dislocation is off-center, the symmetry is broken (there is asymmetry of the boundary with respect to the dislocation); this has an effect on the core configuration since there are constrained atoms—and therefore unbalanced forces—closer to it on one side than on the other. Relaxing the unbalanced forces due to the constraints is the whole point of adding higher order multipoles. Clearly when these forces are asymmetric, there should be asymmetric terms in the displacement field. The coefficients of these terms change sign as the dislocation glides to the corresponding position on the opposite side of the region’s center. Moreover they are observed to decrease in magnitude as the size of region I is made larger.

To actually create the dislocation we take an initial uniform lattice divided into the three regions. The Volterra displacement field—interpreted in the Lagrangian sense—is applied to all atoms, taking the branch cut to be 90 degrees to the Burgers vector. This leads to a layer of overlapped atoms which are then pruned. An iterative procedure is then used to put the atoms in their positions as determined according to an Eulerian interpretation. To find the initial state an iterative procedure is used which alternately relaxes the core atoms for about 100 time steps of the MDmin algorithm and makes a single MD step for the boundary degrees of freedom. This is continued until the sum of the squares of core atom forces and boundary forces is smaller than 10^{-10} . The resulting set of core atom positions and boundary parameters constitutes the initial state. To create the final state the following transformation is applied to the core atoms: For each core or boundary atomic position in the initial state, a new position is considered, determined by adding the Burgers vector to the given position. The closest atom to the new

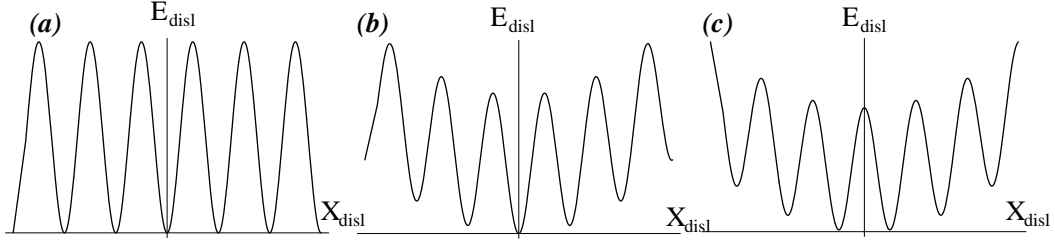


Figure 2.5: Effective potential for dislocation glide in (a) infinite crystal, (b) “centered” simulation and (c) “offset” simulation.

position is found²⁴, and if this atom is a core atom, its position is set to this new position. This effectively recreates the dislocation configuration, shifted upward by one Burgers vector, although no atom moves more than a small fraction of the Burgers vector. The boundary atoms are transformed by adding the Burgers vector to the dislocation position and changing the signs of the coefficients of the reflection-odd multipoles (if the signs are left unchanged they change anyway under minimization, but it takes longer). The resulting configuration is minimized in the same way as for the initial state.

2.5.2 Formulation of Nudged Elastic Band for extended system

We now discuss the formulation of the NEB method in the context of having boundary degrees of freedom as well as individual atomic positions. We would like to treat all degrees of freedom in an equivalent way. However, we cannot literally apply the method pretending that the new degrees of freedom are atomic

²⁴We found it useful to define “closest” to be with respect to a different distance measure, which weights the sum of the squares unequally: distance perpendicular to the shift direction (the Burgers vector) counts more than distance along it.

positions. They have different units, so we must be careful when (i) constructing vectors out of the parameters, (ii) computing lengths and angles and (iii) projecting out perpendicular and parallel parts of force vectors. To see this, consider a two-dimensional system consisting of a position variable x and an angle variable ϕ . Imagine a potential function which has two minima and consider a point on the MEP between them. The force (due to the potential, not the elastic band) on this point points along the MEP, i.e., is parallel to the tangent vector. Now if we change the unit of length, dividing it by a factor Λ , then the numerical values of lengths are multiplied by Λ . In the new units the potential landscape is stretched by Λ in the x direction but not in the ϕ direction. The MEP will similarly be stretched, and we would expect the transformed force at the (transformed) point of consideration to continue to point along the MEP. Now, since force has units of energy over length, and we do not change the unit of energy, the force associated with x , f_x is *divided* by Λ , while the force associated with ϕ , f_ϕ does not change. If $\Lambda > 1$ the ϕ component of the tangent vector increases while that of the force decreases, so the two vectors appear to rotate in opposite directions, destroying the parallelism.

What is wrong with this picture? Once we start applying transformations which are non-orthogonal, such as rescaling in different directions, we cannot treat vectors as Cartesian vectors; their transformation properties are different. In particular we must now distinguish between *contravariant* and *covariant* vectors (and tensors), as well as introduce a *metric tensor*. The fact that the position (and related) vectors and force vectors transform differently indicates that they are of opposite types, and we should not be comparing them (e.g. deciding if they are parallel) directly. Position vectors are “naturally” contravariant, and their “natural”

components are indicated with a superscript, as x^μ . Forces, which are derivatives with respect to positions, are covariant vectors, have subscript indices, as f_μ . The metric tensor is a covariant second rank tensor denoted $g_{\mu\nu}$, symmetric in its two indices. When μ or ν corresponds to a core atom coordinate, we take $g_{\mu\nu}$ to be the identity tensor, $\delta_{\mu\nu}$. We shall discuss the appropriate choice of the metric coefficients for the boundary parameters below. A physical interpretation of the metric is that it gives the length squared of contravariant vectors (summation on repeated indices is understood except where explicitly stated otherwise):

$$ds^2 = g_{\mu\nu} \Delta x^\mu \Delta x^\nu \quad (2.52)$$

If we consider transformations of coordinates of the form $x^\mu \rightarrow x'^\mu = \Lambda^\mu_\nu x^\nu$, then contravariant vectors transform in the same way and covariant vectors and tensors transform thus:

$$f_\mu \rightarrow (\Lambda^{-1})^\nu_\mu f_\nu; \quad g_{\mu\nu} \rightarrow (\Lambda^{-1})^\alpha_\mu (\Lambda^{-1})^\beta_\nu g_{\alpha\beta} \quad (2.53)$$

We can take dot products directly between contravariant and covariant vectors; this is “contracting” an upper index with a lower index. A dot product is invariant under Λ transformations. Thus, when using the natural components of a force and a displacement vector, if we had concluded they were perpendicular by considering the dot product, this would be a valid conclusion. However we cannot infer parallelism using the natural components—as evidenced from the two-dimensional example described above. Parallelism implies that one vector is a multiple of another. To check this we compare the components of the two vectors, but this only makes sense when comparing two vectors of the same type:

$$\vec{a} \parallel \vec{b} \Leftrightarrow a^\mu = \lambda b^\mu \quad (2.54)$$

for some λ . To compare a force and a displacement vector for parallelism we require a means to convert a contravariant vector to a covariant one and vice versa. This is done with the metric tensor; for example we define the covariant components of a displacement and the contravariant components of a force as follows:

$$x_\mu = g_{\mu\nu} x^\nu; \quad f^\mu = g^{\mu\nu} f_\nu \quad (2.55)$$

where $g^{\mu\nu}$ are the contravariant components of the metric—obtained by taking the matrix inverse of $g_{\mu\nu}$. In the NEB we concerned with constructing forces, so we are lowering indices more than often raising them. Before explaining the modifications needed for a “covariant” formulation of NEB, let us review how tangent vectors and spring forces are computed. In the original formulation of the NEB, the tangent vector was computed from the following expression:

$$\hat{\tau}_i = \frac{\mathbf{R}_{i+1} - \mathbf{R}_{i-1}}{|\mathbf{R}_{i+1} - \mathbf{R}_{i-1}|} \quad (2.56)$$

which is the difference vector between replicas, made into a unit vector. In Ref. [31] the following slightly improved method is suggested:

$$\hat{\tau}_i = \frac{\mathbf{R}_i - \mathbf{R}_{i-1}}{|\mathbf{R}_i - \mathbf{R}_{i-1}|} + \frac{\mathbf{R}_{i+1} - \mathbf{R}_i}{|\mathbf{R}_{i+1} - \mathbf{R}_i|} \quad (2.57)$$

where the difference vectors are first normalized. This expression must then be normalized to produce a unit vector. In our application we do not notice any significant difference between the two. We have not implemented the further improvement in Ref.[31] in which the choice of tangent vector involves the relative

energies of adjacent replicas. Another choice concerns the definition of the spring force, given the tangent vector. In the original formulation this was:

$$\mathbf{F}_i^s = [k_+(\mathbf{R}_{i+1} - \mathbf{R}_i) - k_-(\mathbf{R}_i - \mathbf{R}_{i-1})] \cdot \hat{\tau}_i \hat{\tau}_i \quad (2.58)$$

namely, a direct spring coupling between all degrees of freedom, but projected onto the tangent vector. However, regions of high curvature can necessitate adding back part of the perpendicular component of the spring force. As an alternative, Ref. [31] suggests the following expression:

$$\mathbf{F}_i^s = [k_+|\mathbf{R}_{i+1} - \mathbf{R}_i| - k_-|\mathbf{R}_i - \mathbf{R}_{i-1}|] \hat{\tau}_i \quad (2.59)$$

which maintains an equal spacing of replicas (when all springs have the same spring constant k) even in regions of high curvature. However we have noticed some difficulties with this method, namely that large curvature seems to develop near the top of the barrier, so we have kept with the original spring force-method.

We can now list the necessary modifications to the NEB algorithm:

1. Difference vectors between states are constructed in the usual way, including the boundary parameters. These are contravariant vectors.
2. To make unit vectors, divide by the square root of the length squared, computed via eqn. 2.52
3. Construct the tangent vector from the difference vector or unit difference vectors in the usual way, since it is contravariant. Make it a unit vector by computing the length via (2.52).
4. The spring force (original method) is constructed as the sum of difference vectors, multiplied by spring constants and projected onto the tangent vector.

Since difference vectors are contravariant and forces are covariant, we must multiply by the metric to lower the index, both to take the dot product with the tangent, and then to make the uncontracted tangent vector covariant:

$$\mathbf{F}_\mu^s = g_{\nu\rho} [k_+(\Delta_{i,i+1}^\nu) - k_-(\Delta_{i-1,i}^\nu)] \tau^\rho g_{\mu\sigma} \tau^\sigma \quad (2.60)$$

5. To remove the tangential component of the potential force, take the dot product with tangent vector in the usual way (since they are of opposite types) and again use the metric to make the uncontracted tangent vector covariant:

$$\mathbf{F}_{\parallel\mu}^U = \mathbf{F}_\mu^U - (\mathbf{F}_\nu^U \tau^\nu) g_{\mu\rho} \tau^\rho \quad (2.61)$$

Choice of metric

The only question remaining is how to choose the metric coefficients. A natural choice is such that the contribution to the total length squared of a difference vector between two configurations from the boundary parameters is equal to that which would be obtained by adding up the contributions from all of the constrained atoms. This turns out to give exactly the mass matrix:

$$ds_{\text{parameters}}^2 = \sum_{i'} |\Delta \vec{r}_{i'}|^2 = g_{\mu\nu} \Delta p^\mu \Delta p^\nu \quad (2.62)$$

where $g_{\mu\nu}$ is the (non-diagonal) metric given by

$$g_{\mu\nu} = \sum_{i'} \frac{\partial \vec{r}_{i'}}{\partial p^\mu} \cdot \frac{\partial \vec{r}_{i'}}{\partial p^\nu} \quad (2.63)$$

This automatically gives the metric components the correct units. However, there are two issues to consider. First, the mass matrix varies slightly from replica to replica, and from iteration to iteration. Presumably the metric should be fixed. At the very least it should be the same for each replica. Our choice has been to take average of the mass matrix from the initial and final states after each has been minimized. Second, it must be remembered that the boundary parameters are meant to describe not just boundary atoms in the simulation, but all the material out to infinity. However not all of the mass-matrix elements converge as the sum is extended over constrained atoms going out to infinity—the elements associated with $n = -1$ multipoles, and motion of the center, are logarithmically divergent (see the discussion in section 2.4). This divergence should be regularized somehow. Of course, by summing only over the atoms actually in the simulation we are indeed regularizing it. However we observe that the NEB method tends to become unstable when the boundary region is large compared to the core region, and so we regularize further by setting the metric equal to half of the (average of the) mass matrix.

Further NEB details.

We have also implemented the “Climbing Image” technique of Henkelman et al. [32]. After a small number of iterations the highest energy replica is chosen to be a “climbing replica”, that is, the component of the potential force along the tangent is not removed, but reversed, such that it seeks the saddle point (given that the tangent defined by it and its neighbors is approximately along the MEP). The climbing replica is not subject to the spring forces.

When not using the climbing image (CI) method, to estimate the barrier height

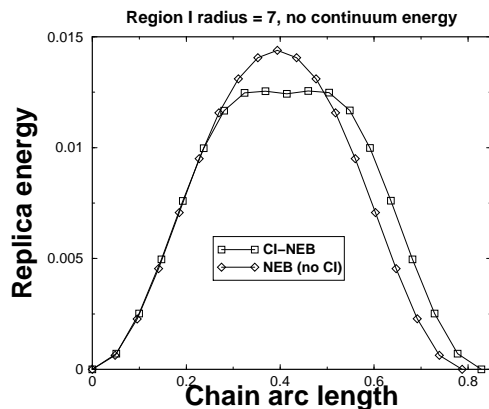


Figure 2.6: Effect of using climbing image (CI) technique in the Nudged Elastic Band method. The number of MDmin iterations was 1000 in both cases.

we fit the top four points of the energy profile to a cubic polynomial, whose maximum we solve for analytically. When using CI, it often turns out that the climbing replica converges more quickly to the saddle point than do its neighbors to their corresponding positions on the MEP, with the result that the energy profile appears to have a small minimum at the saddle point; see Fig. 2.6. This is because the adjacent replicas are not on MEP, and therefore not downhill. That the climbing replica is in fact on or very close to the saddle point can be checked by checking that the total force is close to zero²⁵ and that one eigenvalue of the Hessian matrix is negative.

²⁵We use the sum of the forces squared as a measure of convergence; we do not compute this quantity correctly (i.e., using the metric) since we only care that it decreases, and one positive definite matrix is as good as another for this purpose. When using the CI method, we notice that even though the rate of convergence of the climbing replica and of energy barrier itself is increased, the overall rate convergence of the chain seems to decrease. To check this definitively we should probably compute the correct invariant sum of squared forces, though it is also evident from the minimum in the energy profile.

2.5.3 Running the simulation

We use the MDmin method of minimization to relax the whole chain. At the end of the calculation we compute the energies of the replicas, in particular that of the saddle point. The difference between the saddle-point energy and that of the initial state gives the “forward barrier”; the difference between it and the energy of the final state gives the “backward barrier”—the barrier to move back from the final state to the initial state, which by periodicity (of the real system) is equal to the barrier to move backwards from the initial state. It is also equal to the barrier for moving forwards but under negative shear stress, since in the absence of shear both glide directions are equivalent, so changing the sign of the shear stress must be equivalent to switching forward and backward. Thus for a single NEB calculation we get two barriers, one for each sign of the shear stress (they are equal at zero shear).

We calculate barriers in batches, where a batch (corresponding to one execution of the main program) calculates barriers at several values of σ_{xy} , for particular values of σ_{xx}, σ_{yy} . It is convenient to calculate successive values of σ_{xy} in the same run because the initial and final states for one value are used as initial guesses for the next—if this is not done the dislocation, being started in the zero-shear configuration might move more than one lattice spacing while being relaxed to the given value of σ_{xy} .

Fig. 2.7 shows the barrier energy profile for a single barrier computation. Fig. 2.8 shows the values of the different boundary parameters as a function of replica index, including the end points (the numbering here calls the starting configuration 1). The simulation parameters are listed in table 2.4. The replica with is designated the “climbing image” tends to converge quite quickly to the saddle

Table 2.4: Simulation parameters for Figs. 2.7 and 2.8.

core radius	10
transition zone	$5R_c = 13.5$
free atoms	277
boundary atoms	181+1954
boundary parameters	8
applied stress	(0, 0, 0)
continuum energy	none
N_{replicas}	19
climbing image	yes
NEB steps	1500

point, in the last stages of the minimization, outpacing its neighbors. This leads to the apparent dip in energy in the minimum energy path—which means of course that the system has not converged to the MEP yet. If the full MEP is desired, more iterations should be run. The lagging behind of the non-climbing replicas is also evident in the profiles of parameters which have the same initial and final states, in the corners that form at the climbing replica. If full convergence of the whole MEP was obtained, these corners would be rounded out. The final convergence (squared force summed over atoms, boundary DOFs and replicas) was 0.0085; thus the average remaining force per degree of freedom was about 10^{-7} (the sum of squared forces on the climbing replica was a factor of 10 lower than that on its neighbors).

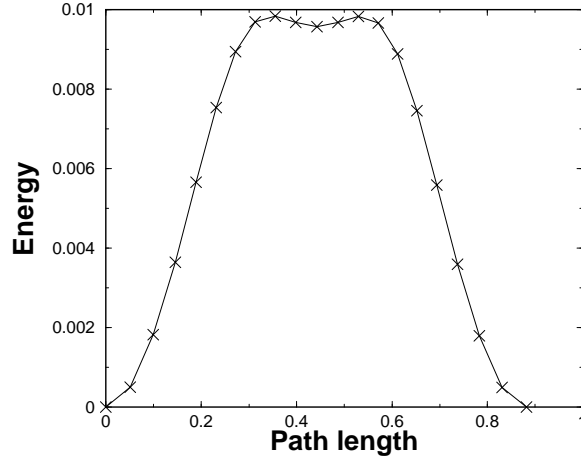


Figure 2.7: Typical barrier energy profile from simulation.

2.6 Representing the data: Functional Forms

In parameter-passing multiscale modeling, a key issue is the formulation of rules for continuum models which efficiently communicate information from smaller scales. Such rules are formulas which encode the results of large numbers of atomistic calculations. For the present work, we seek a formula—a functional form—which represents the stress dependence of the Peierls barrier. From experience in this context and others we have developed a general philosophy for producing functional forms [6]. The aim is to come up with functional forms requiring the fewest fitting parameters, and which can be fit with the smallest possible number of data points (these are of course not independent), given that there are typically so many independent variables. There are five ingredients of this philosophy: (1) Symmetry that is known to be present in the data, such as crystal symmetry (e.g. cubic), must be built into the functional form. (2) Singularities which are known to be in the data, such as cusps, must also be built into the form. A naive approach to fitting

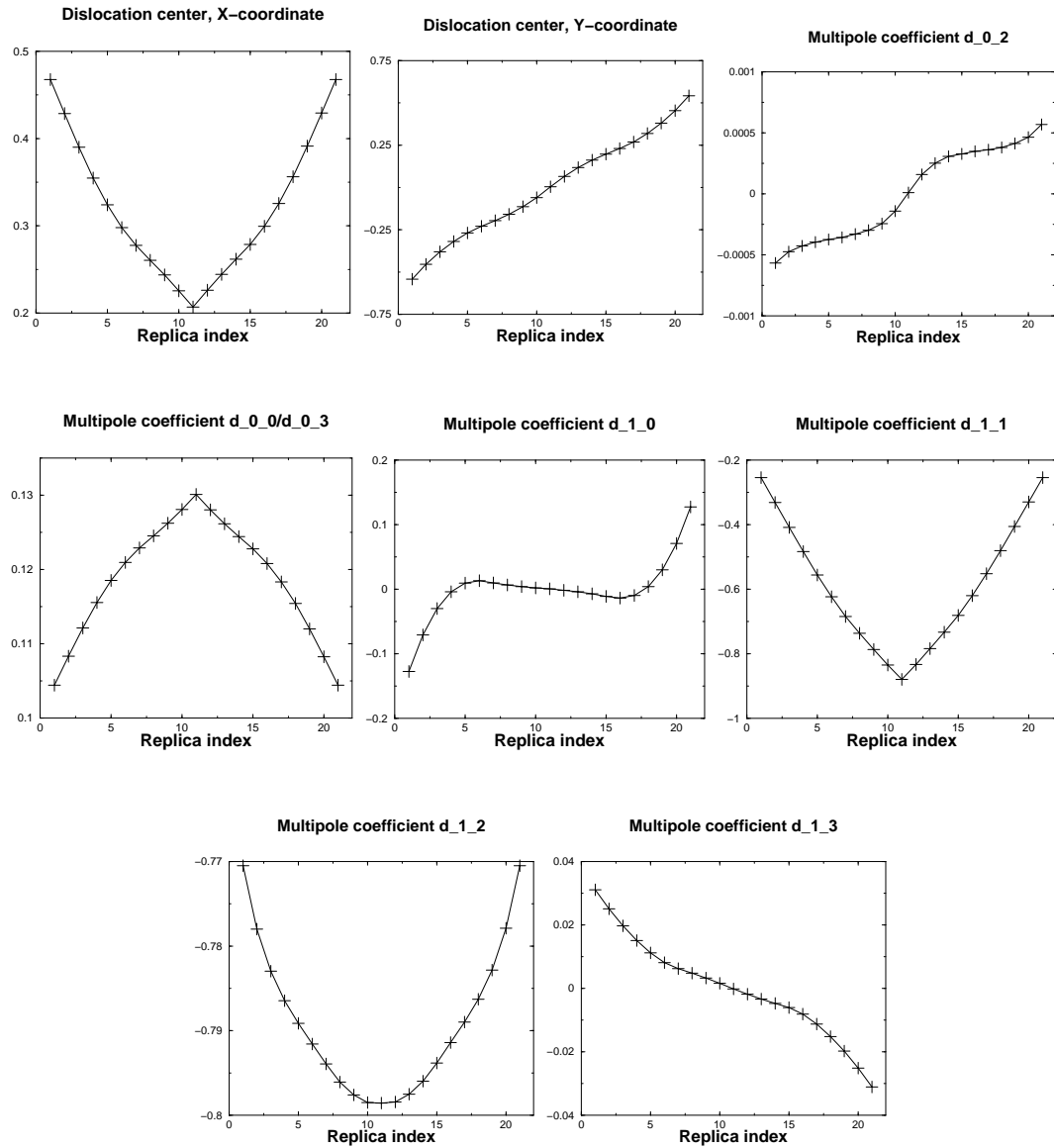


Figure 2.8: Parameter values along transition path.

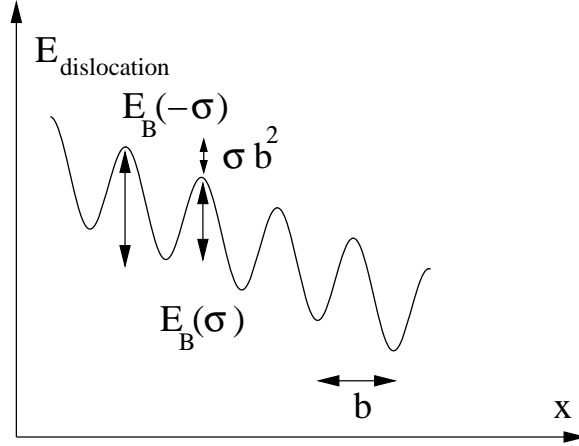


Figure 2.9: Schematic of dislocation potential energy under shear stress σ .

data would be to use some kind of analytic expansion, but such an expansion takes many terms to reproduce a singularity well. (3) To actually choose a functional form one looks for a very simple physical model of the defect. Often a model that is too simple to be of quantitative use can suggest appropriate functional forms which have the right symmetries and singularities. The last two ingredients of the philosophy are (4) subtleties in definitions—e.g. of the position of a defect, or other features of the continuum description that are missing from the atomistic description—and (5) new physics associated with singularities.

2.6.1 Symmetry

In the case of stress-dependent Peierls barriers, considering first only the σ_{xy} -dependence, the symmetry comes from the periodicity of the lattice. As we have seen, each NEB calculation gives us two barriers, for positive and negative shear, which correspond also to the barrier in the forward direction and the barrier in the backward direction; see Fig. 2.6.1. Since these barriers are equivalent by peri-

odicity, the difference between their energies has nothing to do with atomic configuration and everything to do with the work done by the external stress on the dislocation in moving it from one maximum to the next. This is just the force on the dislocation—the resolved shear stress times the Burgers vector—times the distance moved (b), thus the symmetry is (here $\sigma_{xy} > 0$ gives a force in the “forward” direction):

$$E_B(-\sigma_{xy}) - E_B(\sigma_{xy}) = \sigma_{xy}b^2 \quad (2.64)$$

$$\Rightarrow E_B(-\sigma_{xy}) + \frac{1}{2}(-\sigma_{xy})b^2 = E_B(\sigma_{xy}) + \frac{1}{2}\sigma_{xy}b^2 \quad (2.65)$$

$$(2.66)$$

which means that $E_B(\sigma_{xy}) + \frac{1}{2}\sigma_{xy}b^2$ is an even function of σ_{xy} , or $E_B(\sigma_{xy}) = -k\sigma_{xy} + h(\sigma_{xy})$, where $k = \frac{1}{2}b^2$ and $h(x)$ is even function of x . Based on symmetry alone one might consider expanding h in even powers of its argument.

2.6.2 Singularity

At high enough shear stress the barrier vanishes as the unstable maximum in energy merges with the stable minimum. We call the value of σ_{xy} at which this happens the critical shear σ_c . At this point continuous sliding down the potential slope can happen and the dynamics is no longer governed by eqn. 2.2. The vanishing of the barrier at σ_c is the singularity. The merging of stable and unstable equilibrium points like this is known as a *saddle-node bifurcation* in dynamical systems theory (see for example Refs. [63, 28]). The normal form for this kind of bifurcation, $\dot{X} = -\epsilon + X^2$, when put in terms of potential energy (from which the right hand side is derived as a force) shows that to lowest order, the energy barrier (difference

between a maximum and a adjacent minimum of energy) in the vicinity of σ_c is a $3/2$ power law in $\sigma_c - \sigma_{xy}$:

$$E_B(\sigma_{xy}) \sim (\sigma_c - \sigma_{xy})^{3/2} \quad (\sigma_{xy} \lesssim \sigma_c) \quad (2.67)$$

By adding other terms to the normal form, or considering an exact solution of a one-dimensional model (see below) we can show that in general we expect a series of half-integer powers of $(\sigma_c - \sigma_{xy})$, starting with $3/2$. A functional form suggested by consideration of the singularity alone would be:

$$E_B(\sigma_{xy}) = \alpha_{3/2}(\sigma_c - \sigma_{xy})^{3/2} + \alpha_{5/2}(\sigma_c - \sigma_{xy})^{5/2} + \dots \quad (2.68)$$

where $\sigma_c, \alpha_{3/2}, \alpha_{5/2}, \dots$ are fitting parameters.

2.6.3 Simple model

To get a functional form incorporating both the symmetry and the singularity we consider the following one-dimensional potential, non-dimensional position x and shear stress s :

$$E(\sigma_{xy}, x) = \frac{E_0}{2} \left(1 - \cos \left(\frac{2\pi x}{b} \right) \right) - \sigma_{xy} b x \quad (2.69)$$

where b corresponds to the Burgers vector, σ to the shear stress and E_0 to the zero stress barrier. The exact energy barrier between adjacent extrema for this system is

$$E_B(\sigma_{xy}) = E_0 \sqrt{1 - \left(\frac{\sigma_{xy} b^2}{\pi E_0} \right)^2} + \frac{\sigma_{xy} b^2}{\pi E_0} \arcsin \left(\frac{\sigma_{xy} b^2}{\pi E_0} \right) - \frac{1}{2} b^2 \sigma_{xy} \quad (2.70)$$

which is shown in Fig. 2.10. In this figure E has been scaled by E_0 and σ_{xy} by σ_c , which is $\pi E_0/b^2$. Here there is a simple relation between E_0 and σ_c , but we do not expect this in general. We do, however, wish σ_c to be a fitting parameter, so we rewrite eqn. 2.70 in terms of σ_c . Also, we replace b^2 with $2k$ —recall that k is *not* a fitting parameter!

$$E_B(\sigma_{xy}) = \frac{2\sigma_c k}{\pi} \sqrt{1 - \left(\frac{\sigma_{xy}}{\sigma_c}\right)^2} + \frac{2\sigma_{xy} k}{\pi} \arcsin\left(\frac{\sigma_{xy}}{\sigma_c}\right) - k\sigma_{xy} \quad (2.71)$$

An expansion about $\sigma_{xy} = \sigma_c$ gives the expected half-integer series. To generalize this functional form to accommodate real data we wish to identify a part of it that can be expanded in a power series while preserving the symmetry and correct singularity of the basic form. This is not trivial: the linear term is fixed, and the requirement that E_B vanish at σ_c constrains the arcsine term; actually, if you expand the arcsine term about σ_c you get constant and linear terms (which cancel the linear term in the overall function) as well as a series of half-integer powers, starting with the square root. The square root term also expands to a series of half-integer powers starting with square root—its first term cancels that of the arcsine term. Thus the arcsine term has three pieces which cancel parts of the other terms. We cannot multiply it by an analytic function without destroying this, but we can multiply the square root term by an analytic function $h(\sigma_{xy})$. What limitations are there on h ? To satisfy symmetry requirements it must be even. Furthermore being analytic in σ_{xy} implies being analytic in $\sigma_c - \sigma_{xy}$. Consider multiplying the expansion of h by the expansion of the square root term. The latter has a square root term which cancels with the corresponding term from the arcsine term—we must not interfere with this. Since this is the first term we can preserve it by making the constant term in h be unity, in other words $h(\sigma_c) = 1$. A simple choice

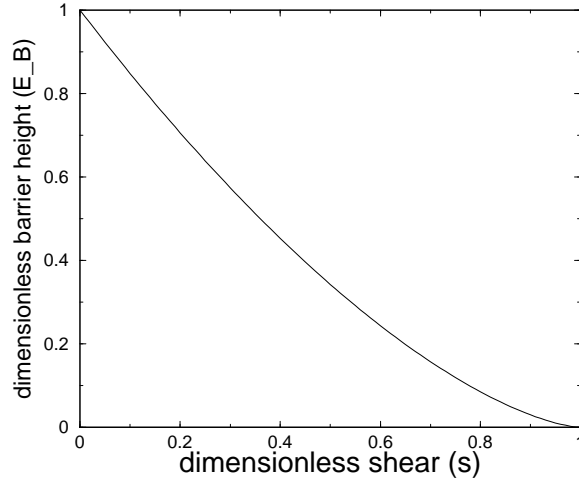


Figure 2.10: Plot of barrier versus stress for sinusoidal potential.

is $h(\sigma_{xy}) = 1 + \sum A_n(1 - (\sigma_{xy}/\sigma_c)^2)^n$. Ideally only one or two terms in the h series are needed for accurate fits; we generally take two terms, giving three parameters for the σ_{xy} -dependence. Our resulting functional form for the σ_{xy} -dependence of the Peierls barrier is therefore:

$$E_B(\sigma_{xy}) = \frac{2k}{\pi} \left(h(\sigma_{xy})\sigma_c \sqrt{1 - \left(\frac{\sigma_{xy}}{\sigma_c}\right)^2} + \sigma_{xy} \arcsin(\sigma_{xy}/\sigma_c) \right) - k\sigma_{xy} \quad (2.72)$$

where the fitting parameters are σ_c and the A_n in $h(\sigma_{xy})$.

2.6.4 Subtleties, extra physics

The subtlety associated with the definition of the dislocation center has been discussed in section 2.2.2. The extra physics here is the fact that as the barrier vanishes, single hops over the barrier are not the whole story—one must consider also double and multiple jumps, and eventually continuous sliding, at which point the dislocation motion becomes dominated by dynamic effects.

2.6.5 Dependence on σ_{xx} and σ_{yy}

The dependence on σ_{xx} and σ_{yy} is less interesting than that on σ_{xy} , having no apparent singularities. To include it in the full functional form we allow the parameters in eqn. 2.72 to depend on σ_{xx} and σ_{yy} and expand that dependence in a low order polynomial—a quadratic. Thus each parameter eqn. 2.72 becomes a function involving six parameters, thus

$$\begin{aligned}\sigma_c(\sigma_{xx}, \sigma_{yy}) &= \sigma_c^{00} + \sigma_c^{10} \sigma_{xx} + \sigma_c^{01} \sigma_{yy} + \sigma_c^{11} \sigma_{xx} \sigma_{yy} + \sigma_c^{20} \sigma_{xx}^2 + \sigma_c^{02} \sigma_{yy}^2 \\ A_n(\sigma_{xx}, \sigma_{yy}) &= A_n^{00} + A_n^{10} \sigma_{xx} + A_n^{01} \sigma_{yy} + A_n^{11} \sigma_{xx} \sigma_{yy} + A_n^{20} \sigma_{xx}^2 + A_n^{02} \sigma_{yy}^2\end{aligned}\tag{2.73}$$

We use three parameters (σ_c , A_1 and A_2) in the basic functional form, giving 18 parameters for fitting the full stress dependence.

2.6.6 Fitting procedure

The data from different σ_{xx}, σ_{yy} batches are fit first for their σ_{xy} -dependence. The fitting algorithm is nonlinear least squares (Levenberg-Marquardt). The values of $\sigma_c, A[n], \dots$ obtained from these initial fits are then fit to quadratic polynomials for their σ_{xx}, σ_{yy} -dependence. The resulting parameters are then used as initial guesses for a single fit of the whole data-set to the full fitting function. By fitting differences between positive and negative barriers to a linear dependence on σ_{xy} we can check that the difference is correctly given by the constant $k = \frac{1}{2}b^2$. In this way each σ_{xx}, σ_{yy} pair yields a value of k . For the CLJ potential, the mean value is 0.6317 ± 0.003 which is greater than the ideal value 0.6196 by 0.012 or 2%. It is not clear if this difference is significant. We used the ideal value to fit the data.

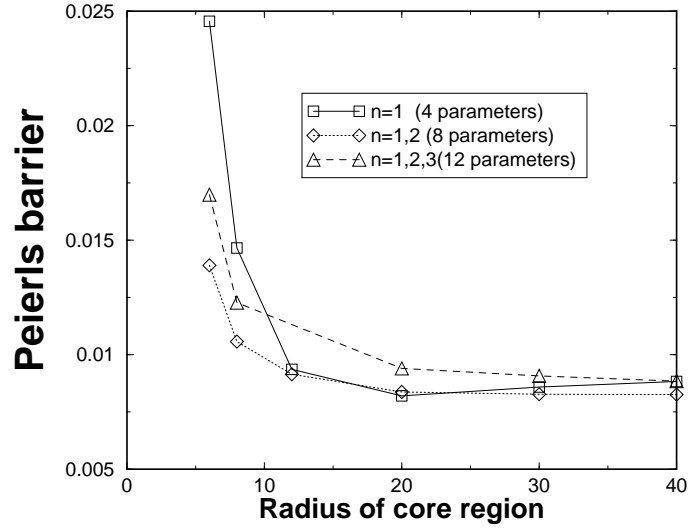


Figure 2.11: Dependence of zero-stress barrier height on core region size, for different numbers of boundary parameters (CLJ potential), without continuum energy.

2.7 Results and discussion

The zero-stress Peierls energy for the CLJ potential is 0.0082 ± 0.0002 ; this value comes from the largest system sizes we have simulated, using eight boundary parameters. This is rather small, as the depth of the pair potential well is unity (in natural units, otherwise ϵ). The Peierls stress is the order of 0.2% of the shear modulus, which makes 2D Lennard-Jonesium intermediate between covalently bonded materials (typically 1% of μ) and ductile fcc metals (typically 0.01% of μ).

2.7.1 Size dependence;

Fig. 2.11 shows the size dependence of the zero-stress barrier for the CLJ potential, for different numbers of parameters: 4, 8 and 12, corresponding to keeping up to $n = 1, 2, 3$ multipoles respectively (it makes sense to add all parameters for

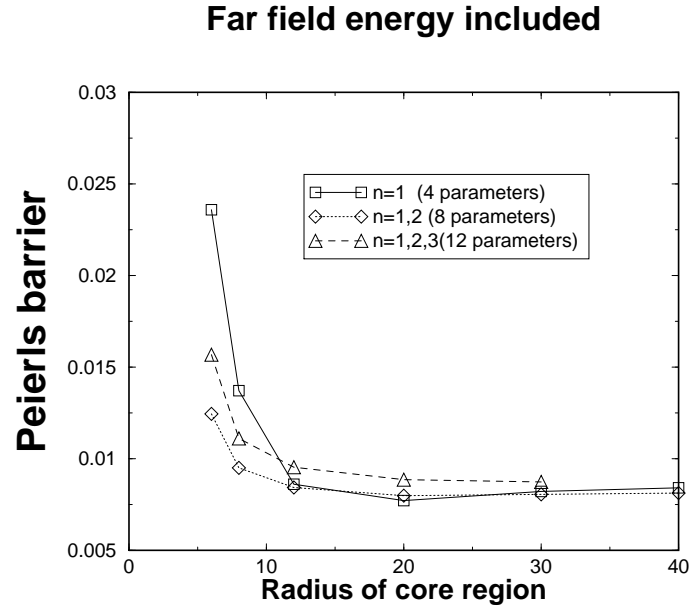


Figure 2.12: Zero-stress barrier height vs. core region size, far field energy included.

a given r -dependence at a time). In these data no continuum contribution to the energy was included. Fig. 2.12 shows the corresponding data for the case where the continuum energy was included. There is not a dramatic difference; to make a comparison it is better to examine the data with and without continuum energy together, as shown in Fig. 2.13, for the case of 8 boundary parameters. The continuum energy terms seem to make a noticeable improvement in the convergence, but not enough to get a good result with a radius smaller than about 10. One surprising aspect of the continuum-energy-included data is that the size dependence ceases to be monotonic. This may be due to difficulties in convergence at the largest system sizes, both the initial and final states, and the transition chain itself.

Notice that while eight parameters seems to give a noticeable improvement

over 4 parameters, the increase of flexibility to 12 parameters makes no further improvement. This is possibly due to the following: the continuum energy terms that we have included depends only on $n = 1$ coefficients. The additional flexibility in the boundary gained by including higher order multipoles allows lowering of energy in the atomistic region, without any cost in the continuum region. Therefore it likely that by including higher multipoles without including appropriate terms in the continuum energy, one *over-relaxes* the energy in the atomistic region, and does not get a closer approximation to the far field energy. A simple calculation involving the $n = 1, j = 3$ multipole, which is the simple volume expansion term, indicates that when the appropriate far field energy is included, the error in the Peierls barrier should go like $1/R_1^3$, if R_1 is the size of region 1, because the energy associated with not relaxing it goes like $1/R_1^2$. Assuming that the same holds for all $n = 1$ multipoles, we should expect the energy barrier computed using 4 parameters to converge to the infinite system limit as $1/R_1^3$. In fact, if the barrier is plotted against $1/R_1^3$, a reasonably straight line results (not shown). Although the magnitude of the error certainly decreases at small R_1 with eight parameters, it still appears to decrease like $1/R_1^3$. The gain in accuracy when using eight parameters is therefore seen to be fortuitous, since we should expect converge at a faster rate (we have not computed the expected power law for $n = 2$ multipoles, but it should presumably be of higher order than for just the $n = 1$ multipoles). Work is in progress to iron-out these details and ensure that there is consistency between the flexibility of the boundary and the terms included in the far field energy.

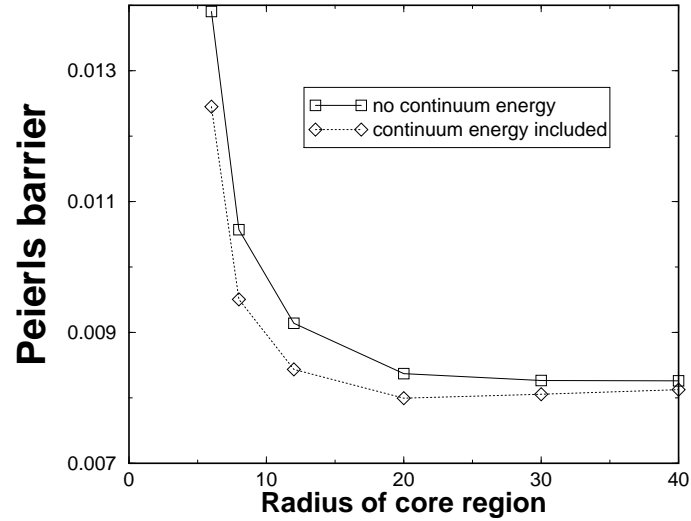


Figure 2.13: Zero stress barrier with and without continuum energy terms, with eight boundary parameters.

2.7.2 Stress dependence

Our main results for are a system with region I radius of 10, and a transition region-width ($\Lambda_2 - \Lambda_1$) of six cutoff distances, with no continuum terms included. It should be noted that the system orientation in the simulations was such that the Burgers vector, and hence also the glide direction, were in the y direction. The formulas given for multipole displacements and stress-dependent terms in the energy assumed the Burgers vector was in the x -direction. In the software, the rotation of the elasticity formulas is accomplished by referring them to arbitrary basis vectors and setting the basis vector accordingly. To implement the stress-dependent terms correctly the components of the applied stress must be appropriately transformed. are: σ_{xx} and σ_{yy} are interchanged while σ_{xy} changes sign. With regard to signs, in presenting the stress-dependent data, we have chosen the following

sign convention: the diagonal components are positive for a stress which tends to compress the system, and the shear stress is positive when it tends to make the dislocation glide in the forward sense. Otherwise, all of the stresses we quote would have extraneous minus signs.

In Figs. 2.14, 2.15, 2.16 and 2.17 we show contour plots of the fitted barrier height function for the CLJ potential in three different planes cutting stress space. These are $\sigma_{xx} - \sigma_{xy}, \sigma_{yy} = 0$, $\sigma_{yy} - \sigma_{xy}, \sigma_{xx} = 0$ and $\sigma_{xx} - \sigma_{yy}, \sigma_{xy} = 0$ respectively. The fit parameters are listed in table 2.5. The ranges of σ_{xx} and σ_{yy} in the plots correspond to the range explored in the computations. The limit of σ_{xy} for a given $\sigma_{xx} - \sigma_{xx}$ pair was whatever it took to make the barrier go to zero, which is the Peierls stress. White represents zero barrier, thus the line bordering the white region in the $\sigma_{xx} - \sigma_{xy}$ and $\sigma_{yy} - \sigma_{xy}$ give the Peierls stress as a function of σ_{xx} and σ_{yy} , respectively. The first feature to note is that the Peierls stress, and thus the Peierls barrier in general, increases with σ_{xx} (i.e., as the material is compressed perpendicular to the slip plane) and decreases with σ_{yy} (i.e., as the material is compressed parallel to the slip plane). At the larger values of σ_{xx} the Peierls stress begins to saturate. It would be interesting to push the value of σ_{xx} up further to see if the Peierls stress remains saturated or begins to rise again.

The errors in the fit can be quantified by making a histogram of the fractional errors. Because we consider barriers which are close to zero, many of the fractional errors are large, and in fact the mean fractional error for the CLJ data is 0.318. However the median fractional error is only 0.028. A histogram of the fractional errors is shown in Fig. 2.18.

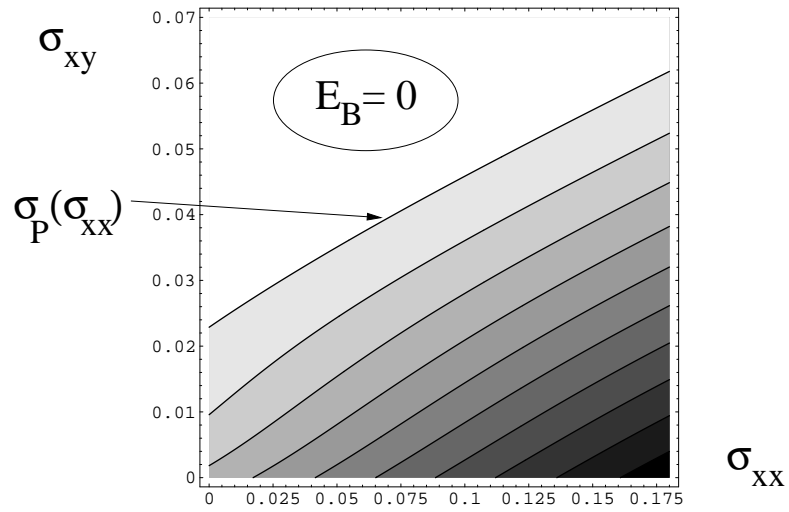


Figure 2.14: Contour plot of Peierls barrier ($\sigma_{xx} - \sigma_{xy}$).

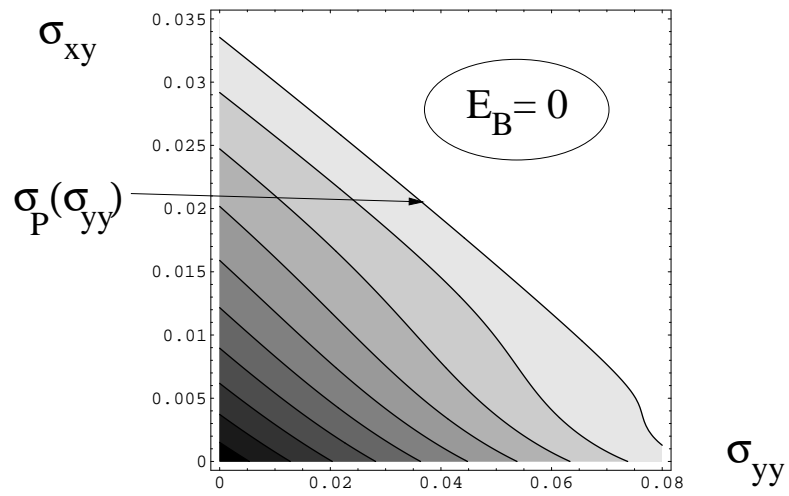


Figure 2.15: Contour plot of Peierls barrier ($\sigma_{yy} - \sigma_{xy}$).

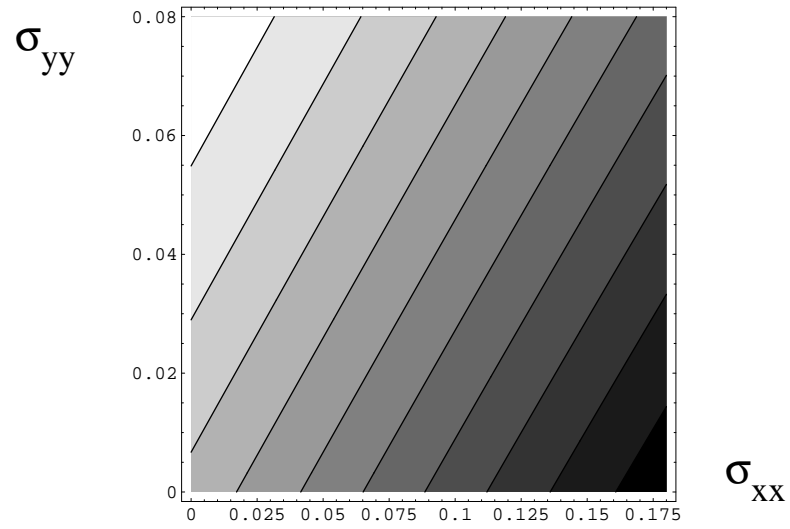


Figure 2.16: Contour plot of Peierls barrier ($\sigma_{xx} - \sigma_{yy}$).

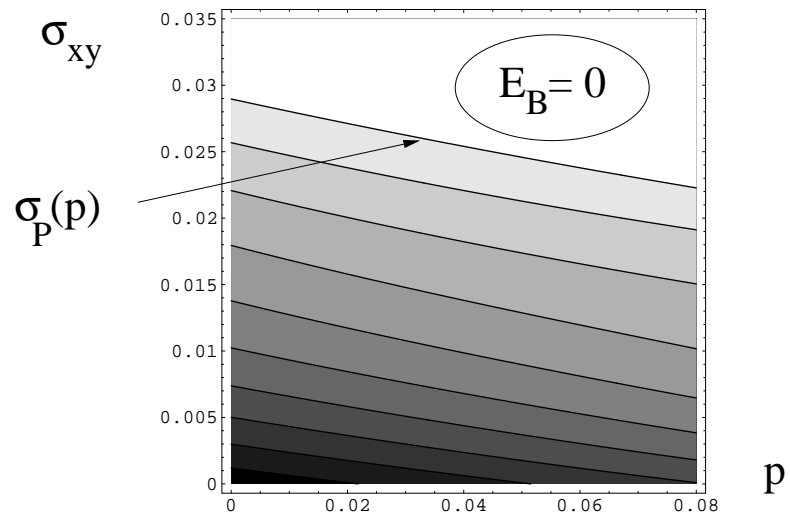


Figure 2.17: Contour plot of Peierls barrier (pressure- σ_{xy}).

Table 2.5: Fit parameters for CLJ potential.

A_1^{00}	0.451238	A_2^{00}	-0.645441	σ_c^{00}	0.0321618
A_1^{01}	5.09441	A_2^{01}	-8.1819	σ_c^{01}	-0.386674
A_1^{02}	34.1911	A_2^{02}	-42.4789	σ_c^{02}	-0.236238
A_1^{10}	-5.93857	A_2^{10}	8.79044	σ_c^{10}	0.292252
A_1^{11}	-27.0486	A_2^{11}	36.7874	σ_c^{11}	0.609572
A_1^{20}	19.6471	A_2^{20}	-23.8286	σ_c^{20}	-0.301861

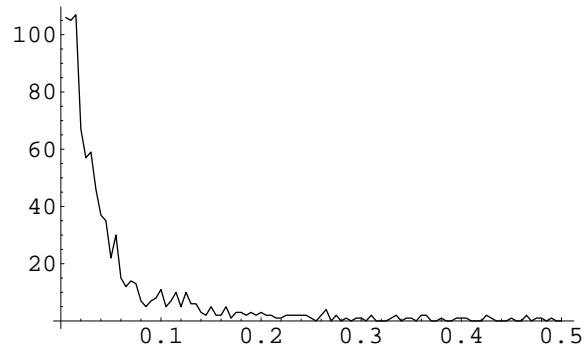


Figure 2.18: Histogram of fractional errors in multidimensional nonlinear fit to barrier data.

2.A 2D versus 3D elasticity

In the Volterra formula for the displacements of a dislocation there appears the Poisson ratio ν ; this also appears in the other terms of the general elasticity solution. We need to choose this in accordance with the interatomic potential being used. There is a formula for ν in terms of for example the Lamé constants, but this formula also depends on the dimension of space, D :

$$\nu_D(\lambda, \mu) = \frac{\lambda}{2\mu + (D-1)\lambda} \quad (2.74)$$

Taking $\lambda = \mu$ for a central potential we have $\nu_2 = 1/3$ and $\nu_3 = 1/4$. However even though we have a two-dimensional material, it turns out that the ν in the Volterra solution should be the 3D one, $\nu = 1/4$. A re-derivation of the Volterra formula following that in Hirth and Lothe [34] for a purely two dimensional material shows that when written in terms of the Lamé constants, the combination of λ and μ that appears in the solution is exactly the 3-dimensional formula for ν . For a central potential $\lambda = \mu$ so we use $\nu = 1/4$.

Alternatively, in order to make use of solutions derived for three-dimensional materials directly, consider the relationship between a purely two-dimensional material and a three dimensional material of unit thickness²⁶ undergoing two-dimensional deformation. For the latter there are two possibilities: *plane stress* ($\sigma_{zz} = 0, \epsilon_{zz} \neq 0$) and *plane strain* ($\sigma_{zz} \neq 0, \epsilon_{zz} = 0$)²⁷. Plane stress is generally associated with thin plates, because material is free to strain in the third dimension; while plane strain is associated with structures which are long in the third direc-

²⁶So we can avoid factors of the thickness which are only necessary to make the units correct.

²⁷There are other possibilities—for example setting the zz component of the appropriate field to something other than zero—but these are the standard cases.

tion, because material far from the ends is constrained by surrounding material to have no displacement in that direction. For an isotropic 3D material with given Young's modulus E_3 and Poisson ratio ν_3 , the resulting two-dimensional behavior is also isotropic. In plane stress and the effective E and ν are the just E_3 and ν_3 ; in plane strain they are modified:

$$E_2^{\text{eff}} = E_3/(1 - \nu_3^2); \quad \nu_2^{\text{eff}} = \nu_3/(1 - \nu_3) \quad (2.75)$$

Going the other way we can choose to map a two dimensional material onto a three-dimensional one, either in plane stress or in plane strain by choosing the relationship between the 2D and 3D elastic constants appropriately. The Volterra solution presented in Hirth and Lothe was derived for plane strain (we also use the plane strain general solution to get the multipoles), so to use it we should map our two dimensional material to a 3D one in plane strain. Inverting 2.75 we get

$$E_3^{\text{eff}} = E_2 \frac{1 + 2\nu_2}{(1 + \nu_2)^2}; \quad \nu_3^{\text{eff}} = \nu_2/(1 + \nu_2) \quad (2.76)$$

It turns out that this mapping is equivalent to the relations between the 2D and 3D E and ν for fixed Lamé constants. Since neither the Volterra formula nor the multipole formulas depend on the Young's modulus²⁸ we only need the expression for ν_3^{eff} —or, we can just remember to use the 3D formula when calculating the Poisson ratio from the Lamé constants.

One final note: we saw that E and ν are unchanged in the mapping between 3D plane stress and 2D. The results of the last paragraph are equivalent to the

²⁸This is the benefit of the E, ν expression of the elastic constants for an isotropic material—it separates them into a dimensional number which sets a quantitative scale for the relation between stress and strain, and a dimensionless number, which characterizes the elastic response in a purely qualitative way.

statement that the Lamé constants are unchanged in the mapping between 3D plane strain and 2D, which is also clear if one compares the elastic free energy, which is naturally expressed in terms of the Lamé constants, of a 3D material of unit thickness which is in plane strain with a 2D material. If we were working with plane strain solutions expressed in terms of the Lamé constants we would not have to worry about using the “3D” version of anything, but if we were working with plane stress solutions involving the Lamé constants we would have to worry.

2.B Lagrangian versus Eulerian coordinates

In modeling the far field of the dislocation using the general solution of two-dimensional linear elasticity, we need to make a choice of interpretation, namely whether the formula is to be interpreted as referring to undeformed positions (Lagrangian) or deformed/current positions (Eulerian). We choose the latter for one important reason, and in this we follow Sinclair et al.[60]—with a Lagrangian interpretation the positions become sensitive to the choice of the branch cut of the logarithm (corresponding to the imaginary cut through the material to make the dislocation), and also the positions obtained do not have the crystallographic symmetry associated with the inserted half-plane. Gehlen et al. [26] used Lagrangian formulation, treating atoms near the cut in a careful way in order to maintain symmetry. However their approach is not feasible for our context where the center of the dislocation is variable. Using Eulerian coordinates makes all these problems disappear, at the expense of needing to use an iterative means to compute the actual current position of an atom, given the displacement field. This will be described now.

We describe the displacement of a material particle, $\vec{u} = \vec{x} - \vec{X}$, using a function \vec{F} . In a Lagrangian description, \vec{F} depends on the undeformed position X as well as some parameters $\{p_\alpha\}$. In an Eulerian description, \vec{F} depends on the current position X , but the functional form remains the same. Thus for a given functional form, such as the Volterra formula for the displacement field of a dislocation, choosing the position variable for \vec{F} to be X or x specifies an Lagrangian or an Eulerian description respectively. Interpreting the formula this way makes a difference to the actual positions (of the atoms) as a function of the parameters $\{p_\alpha\}$. Actually displacing the atoms is simple for the Lagrangian description: given the parameter values and the undeformed position of an atom, one inserts these into the formula for \vec{F} and obtains the displacement, which is then added to X to get the current position x . In the Eulerian description, x appears on both sides of the equation:

$$\vec{x} = \vec{X} + \vec{F}(\vec{x}, \{p_\alpha\}) \quad (2.77)$$

and thus we need to solve for x . In the simulations we do this iteratively. As a first approximation we simply apply the formula in a Lagrangian sense to get the first guess for x , x_0 . Thereafter, we apply the following procedure:

1. Evaluate \vec{F} at the current position to get the supposed displacement
2. Subtract this from the current position to get a supposed undeformed position.
3. Find the closest lattice point to get the actual undeformed position and hence actual displacement.

4. The difference between the supposed displacement and the actual displacement is taken as the correction to the current position. Increment the position by the amount. Accumulate the sum of squared differences as the error.
5. While the error is larger than tolerance, repeat.

Note that we assume the undeformed positions are the sites of a Bravais lattice. If this was not true we could store the undeformed positions explicitly. If there are many terms in the displacement field, they are summed at each iteration—there is no way to do them one at a time. In general the procedure converges very quickly to machine precision (less than 10 iterations). Next, apart from being able to put the atoms in the correct positions according to the Eulerian description, we need the gradient of the position with respect to the parameters (to calculate effective forces on parameters). In a Lagrangian description this would simply be the derivative of \vec{F} with respect to $\{p_\alpha\}$. In the Eulerian description we need to differentiate eqn. 2.77 with respect to $\{p_\alpha\}$ implicitly (the index i refers to a Cartesian coordinate):

$$\frac{\partial x_i}{\partial p_\alpha} = 0 + \frac{\partial F_i}{\partial x_j} \frac{\partial x_j}{\partial p_\alpha} + \frac{\partial F_i}{\partial p_\alpha} \Rightarrow \left(\delta_{ij} - \frac{\partial F_i}{\partial x_j} \right) \frac{\partial x_j}{\partial p_\alpha} = \frac{\partial F_i}{\partial p_\alpha} \quad (2.78)$$

$$\Rightarrow \frac{\partial \vec{x}}{\partial p_\alpha} = (\mathbf{1} - \vec{\nabla} \vec{F})^{-1} \frac{\partial \vec{F}}{\partial p_\alpha} \quad (2.79)$$

2.C Transforming Eulerian to Lagrangian coordinates in continuum energy analysis

As discussed in appendix 2.B we use displacement formulas expressed in Eulerian/deformed coordinates to define the positions of the boundary atoms. These also represent the displacement field of the imagined material all the way to infinity. We use a nonlinear continuum elasticity analysis to calculate the energy contribution from this material. The first step in the analysis is to convert the Eulerian displacement field, which is the sum of the linear strain field, the Volterra field and the multipole fields, to a Lagrangian displacement formula. The link between Eulerian and Lagrangian descriptions is that the actual displacement of a particular material point is the same in either description. This gives us the following rule

$$\vec{u}_L(\vec{X}) = \vec{u}_E(\vec{x}) = \vec{u}_E(\vec{X} + \vec{u}_L(\vec{X})) \quad (2.80)$$

For simple displacement fields this can be solved exactly for $\vec{u}_L(\vec{X})$. Otherwise we must use an iterative procedure. We could insert eqn. 2.80 into itself to get

$$\vec{u}_L(\vec{X}) = \vec{u}_E(\vec{X} + \vec{u}_E(\vec{X} + \vec{u}_L(\vec{X}))) \quad (2.81)$$

However a more practical and efficient procedure consists of starting with the expression $\vec{u}_L(\vec{X})$, iterating the following three steps: (1) apply eqn. 2.80 to every occurrence of $\vec{u}_L(\vec{X})$. (2) Replace every occurrence of $\vec{u}_E(\vec{X} + \vec{u}_L(\vec{X}))$ by its Taylor-expansion truncated at an order large appropriate to whatever order of $1/R$ is being kept. This depends on the lowest order that appears in the Eulerian formula. The formula will now be a mixture of \vec{u}_{LS} , \vec{u}_E and derivatives of the latter.(3) Again

using knowledge of what the lowest order in the actual expression will be, remove terms which will be of too high an order in $1/R$. The maximum order allowed in the displacement field is chosen in accordance with the maximum order desired in the energy density. For $1/R^4$ terms in the density, we need up to $1/R^2$ in the displacement field. Steps (1), (2) and (3) are repeated until there are no more \vec{u}_L s left. At this point the actual \vec{u}_E field and its derivatives can be substituted in. It is necessary then to pass the resulting expression through a function which removes terms of too high order, which can now be done completely, since the actual expressions are available. The truncation procedure is used in several other parts of the analysis, where high order terms are generated from multiplying strains together, etc.

The second aspect of the transformation from Eulerian to Lagrangian coordinates involves the domain of integration. As mentioned in the main text, we need only to do the Θ -integration at fixed R . But we need to know for a given value of R , what values of Θ correspond to the two sides of the gap from the missing material. An exact, explicit formula for the boundary in undeformed coordinates is not easy to obtain for the full Eulerian field involving linear strain field and multipoles. The boundary is defined by $x = 0$, and $\theta(\text{deformed}) = -\pi/2$ or $3\pi/2$, which define coincident straight lines in the deformed configuration, but will be two curves in undeformed coordinates; furthermore if there is applied shear these curves will be skewed away from the negative y -axis.

We can begin to solve for the boundary by writing the relation between deformed and undeformed coordinates in the form $(X, Y) + (u(x, y), v(x, y)) = (x, y)$, where $X = R \cos(\Theta)$; $Y = R \sin(\Theta)$. We call the value of Θ on the positive x side of the cut Θ_- , since it is a negative number; the value for the other boundary is Θ_+ .

Rather than solving for say, deformed positions in terms of undeformed positions, in this case we know one coordinate of each: we know $x = 0$, with y unknown, and we know R with Θ_{\pm} unknown. Thus we have the following two equations:

$$\begin{aligned} R \cos(\Theta_{\pm}) + u_{\pm}(0, y) &= 0 \\ R \sin(\Theta_{\pm}) + v_{\pm}(0, y) &= y \end{aligned} \tag{2.82}$$

with u_{\pm}, v_{\pm} are given by

$$\begin{aligned} u_{\pm}(0, y) &= \pm \frac{b}{2} + \epsilon_{xy}y - \frac{d_1^{(1)}}{6y} - \frac{d_2^{(1)}}{y} \\ v_{\pm}(0, y) &= \epsilon_{yy}y - \frac{b}{2\pi} \left(-\frac{1}{3} + \frac{\log(y^2)}{6} \right) + \frac{d_0^{(1)}}{2y} + \frac{d_3^{(1)}}{y} \end{aligned} \tag{2.83}$$

where only $n = -1$ multipoles have been included. The solution when the d s and ϵ_{ij} are zero is $\Theta_{\pm} = \cos^{-1}(-b/(2R))$. From this we can see how the integration with respect to Θ brings terms down by a power of R . We have tried to solve eqn. 2.82 using a power series expansion (of $\cos \Theta_{\pm}$ and $1/y$ in terms of b/R) but the logarithm makes this problematical. Assuming that using the exact solution will make a difference only to terms in the energy expression of higher order than those we are interested in, we have used the zero-order solution for Θ_{\pm} in doing the Θ -integration. A possible problem with this is that term in the integrand which are of low order in $1/R$, but independent of the boundary parameters may acquire such a dependence from the higher order terms in Θ_{\pm} , and still be low enough order in $1/R$ to count. Thus it would be worth trying harder to solve eqn. 2.82.

2.D Relating changes in elastic energy to work done on boundaries

This appendix continues the discussion in section 2.3.5 regarding the differences between the work done on different shaped boundary curves. This can be accounted for by considering energy changes within the area enclosed by the two curves in question. What follows is a linear elasticity analysis, and the curves we consider are rectangles and circles centered on the dislocation. Taking just the linear and logarithmic (Volterra) terms in the displacement field, we find the change in energy density as the dislocation center moves: since the energy density is quadratic in the strain, there are three terms. One is the energy of the linear displacement field, which is independent of the dislocation center. One is the dislocation self-energy (the divergence does not matter for this part of the argument), which depends on the center but for the symmetric boundary integrals we are considering, the changes ahead of and behind the dislocation have opposite signs and cancel out (since this energy is essentially “carried along” by the dislocation. Finally there is a cross term which is linear in the applied stress and in the burgers vector. This term accounts for the changes in the integrals. It turns out to be proportional to $\cos(4\theta)/r^2$, and the integral over the area between a square and a surrounding circle gives the difference in the work done on these respective curves, and vice versa.

This is a little confusing given the fact that there is no difference for integration paths of the same shape but different sizes. A corollary of this is that the integral of the energy density change between a circle and any surrounding square is the negative of the integral between a square and any surrounding circle; see

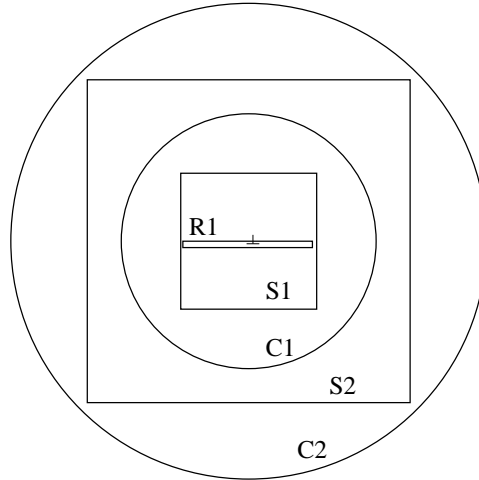


Figure 2.19: When the dislocation glides a distance b , the work done on the boundary of squares $S1$ and $S2$ is $0.58\sigma_{xy}b^2$, that on circles $C1$ and $C2$ $2\sigma_{xy}b^2/3$. On rectangle $R1$ it is very nearly $\sigma_{xy}b^2$.

figure 2.19. Note that the connection between line integral and area integral—the divergence theorem—does not apply when the area includes the logarithmic singularity²⁹, that is, the dislocation itself, so we cannot directly integrate the energy change within the circular boundary to find the missing one-third of the work done on the dislocation when the center moves. Interestingly it does show up in the simulation—without including the work terms, we find a difference between the forward and backward barriers equal to one third of the expected value $\sigma_{xy}b^2$. The other two-thirds comes from the external work done. With the circular boundary, when the dislocation glides by one Burgers vector, the external sources do work $2\sigma_{xy}b^2/3$, the elastic energy within the boundary is decreased by an amount $\sigma_{xy}b^2/3$, so that the work done on the slip plane is the expected amount

²⁹The divergence theorem requires the integrands to be continuous and differentiable at every point in the domain.

$\sigma_{xy}b^2$. It should be possible to integrate the energy density change over the circle minus a thin strip which includes the slip plane (or do a semi-circle minus this strip, and double the result) but it was not obvious how to do this in Mathematica. Experiments with rectangular regions make it clear how as the aspect ratio of the rectangle increases, say by making the rectangle thinner and thinner, the work calculated by the line integral approaches the limit $\sigma_{xy}b^2$, with the difference as the width is changed being exactly accounted for by the appropriate area integral. The work done on the slip plane does not end up as stored energy anywhere; it is dissipated as heat.

Chapter 3

Fracture of Notched Single Crystal Silicon

3.1 Introduction

Recently there has been experimental[22, 66, 67] and theoretical[76] interest in fracture in sharply notched single crystal silicon samples. Such samples have technological importance because silicon is a commonly used material in the fabrication of MEMS devices; the etching process used tends to create atomically sharp corners due to highly anisotropic etching rates [67]. Failure in such devices is often a result of fracture which initiated at sharp corners[65]. In the case of a notch, there exists a parameter K analogous to the stress intensity factor of traditional fracture mechanics, which parameterizes the elastic fields in the vicinity of the notch. Suwito et al.[66, 67] have carried out a series of experiments which have (i) established the validity of the stress intensity factor as a fracture criterion in notched specimens and (ii) measured the critical stress intensities for several notch geometries. On the theoretical side Zhang[76] has carried out an analysis which

models the separation of cleavage planes by a simple cohesive law, and thereby derived a formula for the critical stress intensity as a function of notch opening angle. The material properties which enter this formula are the elastic constants and the parameters of the cohesive law, the peak stress $\hat{\sigma}$ and the work of separation Γ_0 . This recent interest has prompted us to investigate the phenomenon of fracture in notched silicon using atomistic simulations: In this chapter we present direct measurements of the critical stress intensity for different geometries (i.e., notch opening angles) and compare them to the experimental results of Suwito et al. We apply a load by specifying a pure K -field of a given strength (stress intensity factor) on the boundary of the system; we do not have to infer a value of K from some arbitrary far-field load and system geometry (this is the power of atomistic simulations, although in other respects, such as the accuracy of the interatomic potential, that power is limited). In doing this we are effectively using the result of Suwito et al. that the notch stress intensity factor is indeed the quantity which determines fracture initiation, so we can ignore higher order terms in the local stress field.

3.1.1 Elastic fields near a notch

The essential geometry of a notch is shown in Fig. 3.1. The notch opening angle is denoted γ and the half-angle within the material, which is the polar angle describing the top flank, is β (thus $\beta = \pi - \gamma/2$). As discussed in detail by Suwito et al. [66, 67], it is fairly straightforward to solve the equations of anisotropic linear elasticity for a notched specimen. The formalism used is known as the Stroh formalism [73], which is useful for dealing with materials with arbitrary anisotropy in arbitrary orientations, as long as none of the fields depend on the z coordinate

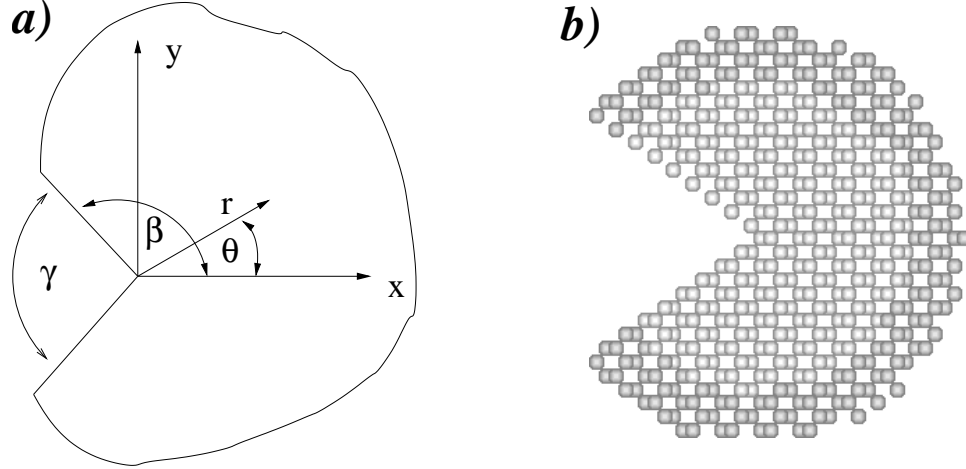


Figure 3.1: (a) Notch schematic and notation; (b) silicon crystal with a notch; darker layer is fixed boundary atoms.

(this will be the out-of-plane coordinate; note that this does not restrict the deformation itself to be in-plane). Here we only consider mode I (symmetric) loading. The displacement and stress fields for a notch can be written as

$$u_i = Kr^\lambda g_i(\theta) \quad (3.1)$$

$$\sigma_{ij} = Kr^{\lambda-1} f_{ij}(\theta) \quad (3.2)$$

where λ plays a role like an eigenvalue; its value is determined by applying the traction-free boundary conditions to the notch flanks. There is an infinity of possible values for λ of which we are interested in those in the range $0 < \lambda < 1$, which give rise to a singular stress field, often known as the K -field, at the notch tip. This is entirely analogous to the singular field near a crack tip, which is simply the limiting case where the notch opening angle goes to zero (or the angle β goes to π), and λ becomes one half. Further details of the Stroh formalism are given in

appendix 3.B. The complete elastic solution involves the whole infinity of values for λ , corresponding to different multipoles of the elastic field. Negative values of λ correspond to more singular fields which are associated with properties of the core region stemming from the non-linear atomistic nature of this region; they do not couple to the far-field loading. $\lambda > 1$ corresponds to fields which are less singular, and do not influence conditions near the notch-tip, since the displacements and stress vanish there. They are, however, essential to represent the full elastic field throughout the body, and ensure that boundary loads and displacements (whatever they may be) are correctly taken into account. This is the basis for asserting that only the K -field is important. This field is unique among the multipoles in that it both couples to the far-field loading and is singular at the notch tip. Thus the stress intensity factor must characterize conditions at the crack tip, and therefore a critical value, K_c , is associated with the initiation of fracture. The validity of this approach hinges on the validity of linear elasticity to well within the region in which the K -field dominates.

From eqn. 3.2 we see that the units of K and therefore K_c are $stress/length^{\lambda-1}$ which depends continuously on the notch angle γ through λ . Hence the shape of a plot of K_c against notch angle depends on the units used to make the plot. In metric/SI units K_c changes by an order of magnitude between 70° and 125° whereas if an atomic scale unit of length is used the plot is nearly flat (Fig. 3.15). The most interesting feature of this is that it seems to provide a direct link from macroscopic measurements to a microscopic length scale. From a continuum point of view, one incorporates atomistic effects into fracture via a *cohesive zone*, a region ahead of the crack tip where material cleaves according to a specified force-separation law. One of the parameters of such laws is the length scale—the distance two surfaces

must separate before the attractive force goes to zero—which for a brittle material is an atomic length scale. It is this scale that one would identify from the plot of K_c versus angle. Note that one can only identify a *scale*, and not an actual length parameter, in particular because the different geometries that are involved in the plot involve different fracture surfaces, with presumably different force-separation parameters.

Fig. 3.2 shows the normal and shear stresses on radial planes (perpendicular to the plane of the sample) emanating from the notch tip, for unit K and r (i.e., they are derived from the tensor f_{ij} appearing in eqn. 3.2). The figures show the functions for the $\gamma = 70^\circ$ case; the other geometries have the same qualitative behavior. Both stresses vanish at the maximum angles, corresponding to the notch flanks; this is in accordance with traction-free boundary conditions. What is most important to note is that the normal stress, which presumably is most relevant for cleavage on a radial plane, has its maximum at $\theta = 0$. The shear stress, which is relevant for possible slip behavior (dislocations) which could compete with cleavage as a means of relieving stress, is zero at $\theta = 0$, and has a maximum at intermediate angles. If there is an easy crystal slip plane in the vicinity of the maximum, slip could conceivably compete with cleavage.

3.2 Simulation

3.2.1 Geometry

We simulated a cylindrical piece of silicon with a notch, making a ‘PacMan’ shape as in Fig. 3.1(b), consisting of an inner *core* region and an outer *boundary region*. By focusing on just the initiation of fracture we avoid the need for large systems

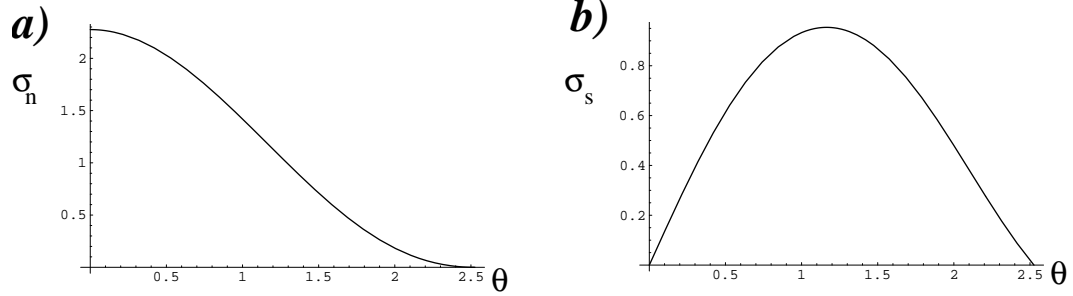


Figure 3.2: Normal (a) and shear (b) stresses on radial planes as functions of plane angle, for $\gamma = 70^\circ$.

since we are not interested in the path the crack takes after the fracture (if we were, we would have a problem when the crack reached the edge of the core region and hit the boundary which is only a few lattice spacings away). We consider three notch geometries, which we call the 70° (actually 70.5288°), 90° and 125° (actually 125.264°) geometries respectively, referring to the notch opening angles. The 70° sample has $\{111\}$ surfaces on the notch flanks and the plane of the sample is a $\{110\}$ surface. The 90° sample has $\{110\}$ surfaces on the flanks and the plane is a $\{100\}$ surface (in this case the crystal axes coincide with the coordinate axes). The 125° sample has a $\{111\}$ on the bottom flank and a $\{100\}$ surface on the top flank, while the plane of the sample is a $\{110\}$ plane. In addition, we studied the zero degree notch geometry, corresponding to a standard crack. The crack plane is a $\{111\}$ surface and is the xz plane in the simulation, and the direction of growth is the $\langle 211 \rangle$ direction, which is the x direction in the simulation. The radius of the inner, core region in almost all the cases presented is 5 lattice spacings or about 27\AA . The exceptions were the crack geometry for the EDIP potential (core radius was 7.5 lattice spacings—the larger size makes the ductile behavior of the potential

more obvious) and the 90° geometry with the MEAM potential (core radius was four lattice spacings because this potential is computationally more demanding). The coordinate system in each case is oriented so that the plane of the sample is the xy plane and the notch is bisected by the xz plane.

3.2.2 Potentials

We have used three different silicon potentials. The first is a modified form of the Stillinger-Weber [62] potential (mSW), in which the coefficient of the three body term has been multiplied by a factor of 2. This has been noted by Hauch et al. to make the SW potential brittle; they were unable to obtain brittle fracture with the unmodified SW potential. However it worsens the likeness to real silicon in other respects such as melting point and elastic constants[37, 38, 29]. The second potential is a more recent silicon potential known as “environment-dependent interatomic potential” (EDIP)[10, 44], which is similar in form to SW but has an environmental dependence that makes it a many-body potential. Bernstein and coworkers[12, 1, 13] have used EDIP to simulate fracture in silicon. They reported a fracture toughness about a factor of 4 too large when compared with experiment, and that fracture proceeds in a very ductile manner, accompanied by significant plastic deformation and disorder. On the other hand, using tight-binding molecular dynamics near the crack tip they successfully simulated brittle fracture in silicon. In view of failure failure of many empirical potentials to simulate brittle fracture, Pérez and Gumbsch[52] used density functional theory to simulate the fracture process, measuring lattice trapping barriers for different directions of crack growth on different fracture planes. A reason for the failure of empirical potentials that has been proposed in Ref. [13] is that their short-ranged nature necessarily

requires large stresses to separate bonds. This however is not the case in our third potential, which is the modified embedded atom method (MEAM) of Baskes[8]. This is a many-body potential similar to the embedded atom method but with angular terms in the electron density; it has been fit to many elements including metals and semi-conductors. A significant feature of this potential is its use of “three-body screening” in addition to the usual pair cut-off distance. This means that atoms in the bulk see only their nearest neighbors, while surface atoms, on the other hand, can see any atoms above the surface (for example on the other side of a crack) within the pair cut-off distance. The pair cut-off has been set to 6 Å to allow the crack surfaces to see each other[9] even after they have separated. The MEAM potential has been used successfully to simulate dynamic fracture in silicon[9], and we have found it to be the most reliable potential in our studies of notch fracture. In table 3.1 we list the low-index (relaxed, unreconstructed) surface energies for the three potentials.

3.2.3 Boundary Conditions

The boundary conditions are as follows: in the z -direction (out of the page) there are periodic boundary conditions. The thickness of the sample in this direction is always one or two repeat distances of the lattice in that direction. For the 70° and 125° geometries the repeat distance is $\sqrt{2}a$ where a is the cubic lattice constant; for the 90° geometry it is $2a$. In the plane, the boundary conditions are that an layer of atoms on the outside of the system has the positions given by the analytic formula (3.1) for displacements from anisotropic linear elasticity, with a specified stress intensity factor K . The thickness of the layer is twice the cutoff distance of

Table 3.1: Surface energies for silicon according to mSW, EDIP and MEAM potentials.

potential	surface	atomic units	SI units
mSW	111	0.1718	1.3593
mSW	110	0.2105	1.6649
mSW	100	0.2976	2.3545
EDIP	111	0.06538	1.0475
EDIP	110	0.08194	1.3128
EDIP	100	0.1320	2.1150
MEAM	111	0.07668	1.2285
MEAM	110	0.09030	1.4469
MEAM	100	0.08126	1.3019

the potential, in order that the core atoms feel properly surrounded by material¹. We interpret the displacement formulas in terms of Eulerian coordinates, using an iterative procedure to compute the current positions. The numbers of core atoms were 890, 894, 1260 and 892, for the 0° (crack), 70°, 90° and 125° systems respectively (except for the EDIP/crack case where the core radius was 7.5 lattice constants; there the number of core atoms was 2002). The number of boundary atoms depends on the potential (through the cutoff distance); it is typically about 500 atoms. For the most part no special consideration was given to the lattice origin, which meant that by default it coincided with the notch tip². In a few cases it was necessary to shift the position of the origin in order to make sure that the notch flanks were made cleanly, in particular so that the {111} flanks in the 70° and 125° geometries were complete close packed {111} surfaces, rather than having dangling atoms.

3.2.4 Critical stress intensities

The simulation consists of alternating the following two steps: (1) We increment the value of K by a small amount, changing the positions of the boundary atoms accordingly. (2) We relax the interior atoms as follows. First we run about 50 steps of Langevin molecular dynamics with a temperature of 500-600 K; the purpose of this is to break any symmetry (the 70° and 90° samples are symmetric about the xz plane). It is still a zero temperature simulation; these finite temperature steps are simply a way to introduce some noise. Next we run 500 time steps of the dynamical minimization technique known as “MDmin” (a Verlet time step is

¹It has to be twice because of the three- and many-body terms in the potential.

²When applying singular elastic deformations, a check ensures that an atom sitting at the location of the singularity is simply not displaced.

carried out, but after each velocity update, atoms whose velocities have negative dot-products with their forces have their velocities set to zero). Finally 500 time steps of conjugate gradients minimization are carried out. We observe that the combination of both types of minimization is more effective (converges to a zero force state more quickly) than either alone. The procedure generally results in the atoms having forces of around $10^{-5}\text{eV}/\text{\AA}$.

The initial value for K could be zero; however it turns out to be possible to start from a fairly large value of K by applying the analytic displacements to the whole system at first. When the critical K value, K_c , is not yet known the increment size is chosen reasonably large to quickly find the K_c . When this has been found, the simulation is restarted from a value below the critical value with smaller increments and a more accurate value for K_c found. The increment is a measure of the uncertainty in K_c .

3.3 Results

3.3.1 Observed fracture behavior

We observe brittle cleavage of the simulated crystals at definite values of K for all geometries using the mSW and MEAM potentials, but only for the 70° geometry when using the EDIP potential. Figures 3.3-3.14 show snapshots of the simulation process for the different geometries and potentials. In most cases three snapshots are shown: one of the configuration immediately before crack initiation, one of the configuration immediately after initiation, and one of a “late-stage” configuration, to illustrate the fracture plane more vividly; generally this was chosen to be the configuration corresponding to the highest applied load, which depended on how

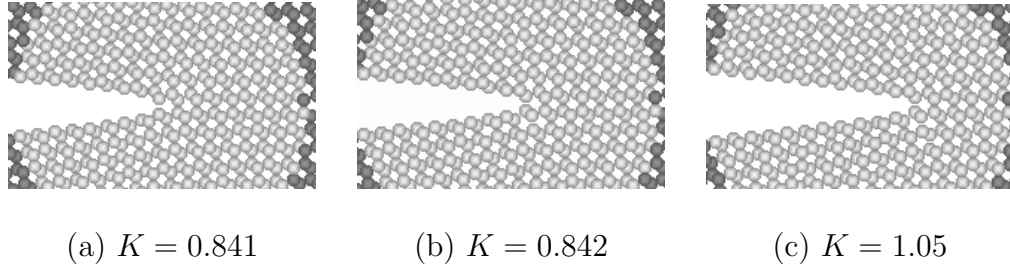


Figure 3.3: mSW-crack.

long the simulation was run past the initiation point. For the EDIP potential, which gives unphysical ductile behavior (except in one case, the 70° geometry), more snapshots are shown, in order to illustrate the plastic behavior more completely, since a variety of stages is involved.

The behavior in crack geometries is shown in Figs. 3.3-3.5. The initial applied load must be such that no crack healing takes place upon relaxation, so that the location of the crack corresponds to the center of the system (in reference to which the boundary displacements are calculated). In this case we are not investigating crack initiation (since the notch is already a crack) but crack growth; the critical K_c is defined as that at which the crack advances, or when the next bond across the crack plane breaks. This is somewhat hard to see in the figures; one must count atoms along the crack surface and compare from one figure to another to see that growth has occurred.

The mSW and MEAM potentials produce similar, brittle, fracture behavior. The EDIP potential produces quite different behavior; the crack propagates in a ductile manner. Frame (a) shows the configuration before any plastic deformation has taken place. Frame (b) shows what appears to be the nucleation of a dislocation onto the $\{110\}$ slip plane which is at an angle of 54.6° to the positive x -axis.

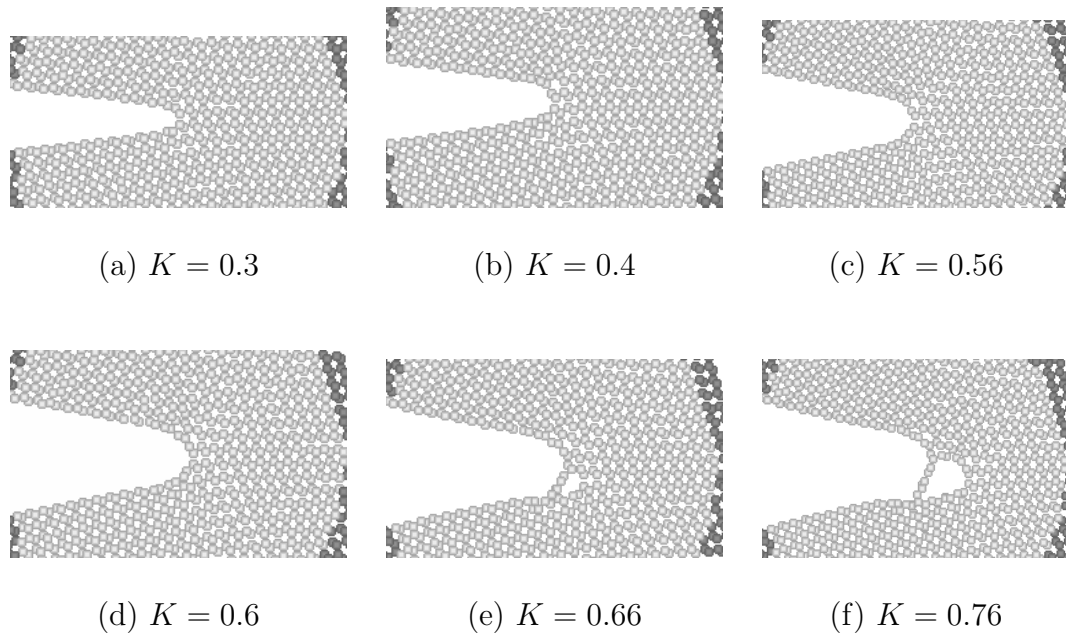


Figure 3.4: EDIP, crack.

By frame (c) the crack tip has blunted noticeably, and in frame (d) a growth of the blunted crack by about a lattice constant has taken place—we take the stress intensity at this stage to be the critical value. Frames (e) and (f) show a void nucleating and growing behind the crack tip, which would under further loading join with the crack—such coalescence of voids the essence of ductile crack growth.

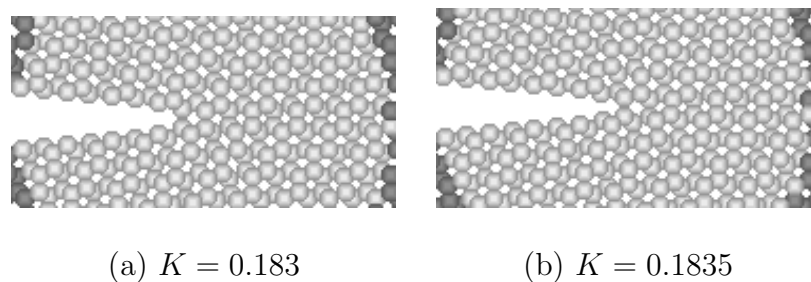


Figure 3.5: MEAM, crack.

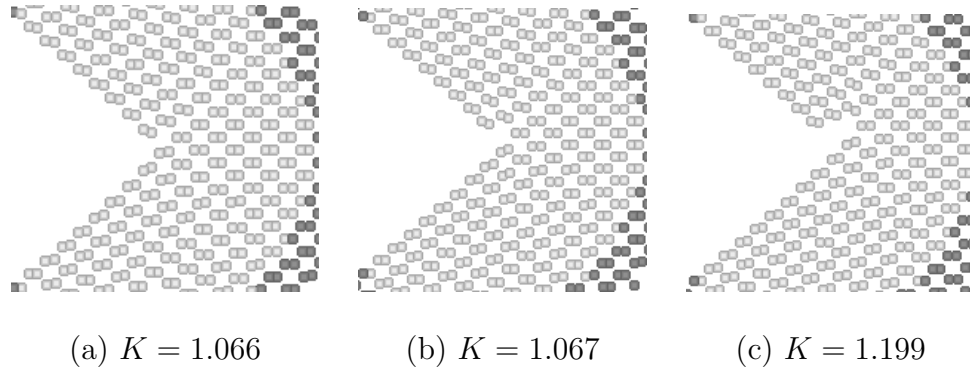


Figure 3.6: mSW-70°.

In the 70° system (Figs. 3.6-3.8) fracture occurs along a $\{111\}$ plane. There are two choices for this, symmetrically placed with respect to the xz plane. Here all three potentials produced brittle behavior; this was the only geometry in which the EDIP potential did so. Possible reasons for this exception are discussed in section 3.4. However, when the origin was not shifted as mentioned in section 3.2.4, so that the notch flanks had dangling atoms, the EDIP-behavior was quite different: the notch blunted to a width of several atomic spacings.

The behavior for 90° models is shown in Figs. 3.9-3.11. We get three different behaviors for three different potentials—providing a cautionary demonstration of the limitations of empirical potentials. The easy cleavage planes available here are the $\{110\}$ planes which are extensions of the notch flanks. The mSW model starts to cleave along the lower of these (the extension of the upper flank) but the crack advances only one atomic before cleavage switches to an adjacent parallel plane. The net result is a kind of “unzipping” along the hard $\{100\}$ plane. This is presumably because the peak in the normal stress across this plane, compared to the normal stress at the 45° angle, outweighs the increased cost of cleavage (but note that the surface energy ratio $\gamma_{100}/\gamma_{111}$ is in fact lower for MEAM, which cleaves

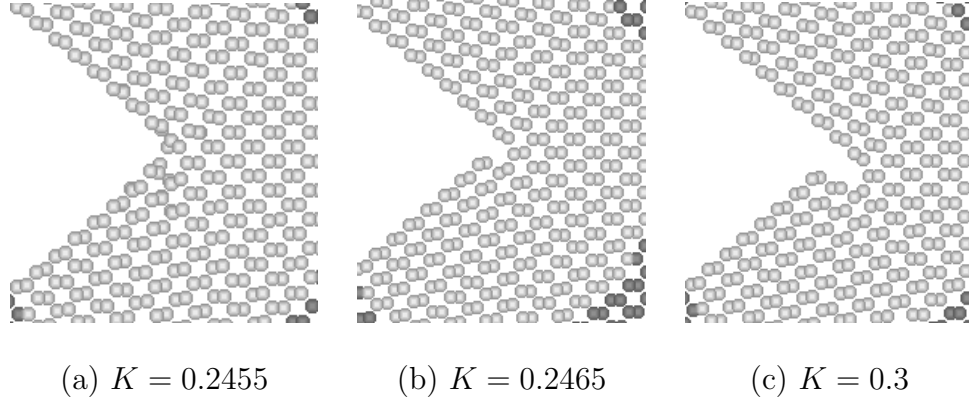


Figure 3.7: EDIP-70°.

on the $\{110\}$ plane—see table 3.1 for the energies of the different surfaces according to the different potentials). The EDIP potential deforms plastically in this case, as depicted in the six frames of Fig. 3.10. It is harder to identify specific processes here than in the 70° case, including where crack growth starts, though it seems to have definitely started by the frame (c) ($K_c = 0.6$). The MEAM potential behaves in the manner most consistent with experiment, namely cleaving on $\{110\}$ planes, and switching from one to the other—this is illustrated dramatically in the third frame of Fig. 3.11. Experimentally switching between planes, when it happens, occurs over longer length scales ($25\mu\text{m}$ for the 70° case[66]), although the behavior at atomic length scales has not been examined. Too much should not be read into the switching we observe, because once cleavage has occurred over such distances the proximity of the boundary probably has a large effect on the effective driving force on the crack.

For the 125° geometry (Figs. 3.12-3.14), there are again two $\{111\}$ planes to choose from but they are not symmetrically placed. Fracture occurs for the mSW and MEAM potentials on the one closest to the xz plane, i.e., closest to the plane

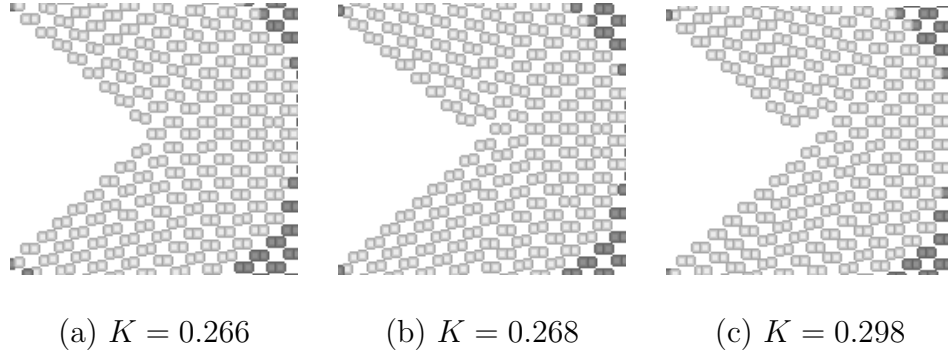


Figure 3.8: MEAM-70°.

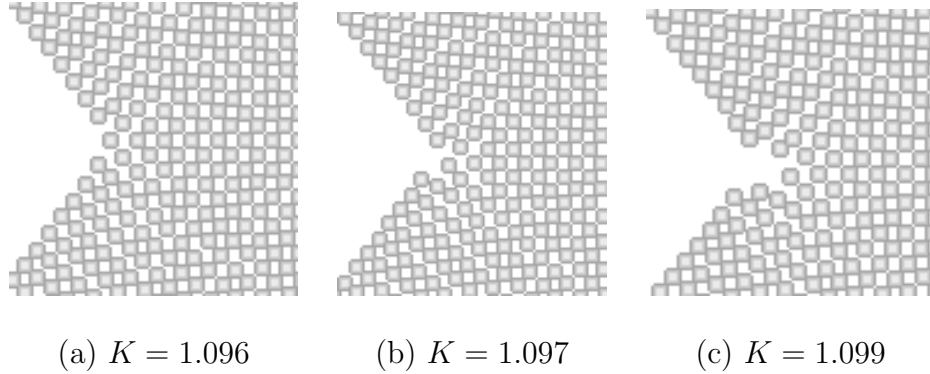


Figure 3.9: mSW-90°.

of maximum normal stress, which is the $(11\bar{1})$ plane. The direction of growth is $[21\bar{1}]$, and growth proceeds much more readily than in the other notch geometries, presumably because it is along the high stress plane. In the EDIP system, plastic deformation is favored over cleavage. This appears to proceed as follows: First slip occurs on the $(11\bar{1})$ plane in the $[21\bar{1}]$ direction, as a single edge dislocation is nucleated (frame (a)-frame (b)). Next, slip occurs on the other $\{111\}$ plane, the $(\bar{1}\bar{1}1)$ plane, in the $[21\bar{1}]$ direction, with two dislocations being nucleated (frame (b)-frame (c)-frame (d)), on adjacent $(\bar{1}\bar{1}1)$ planes. In the last two frames a void appears and grows.

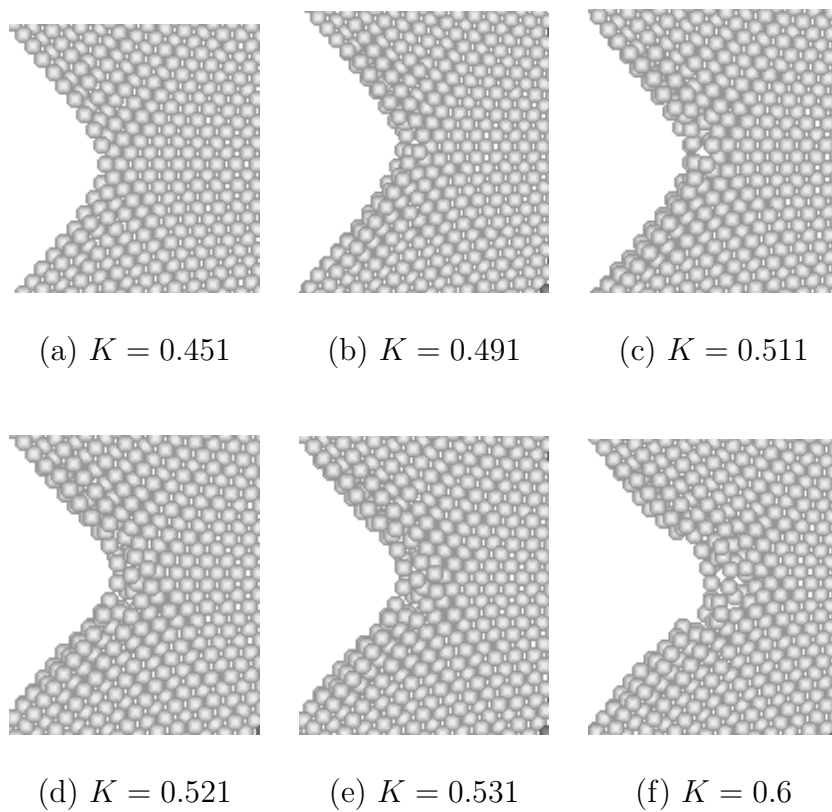


Figure 3.10: EDIP-90°.

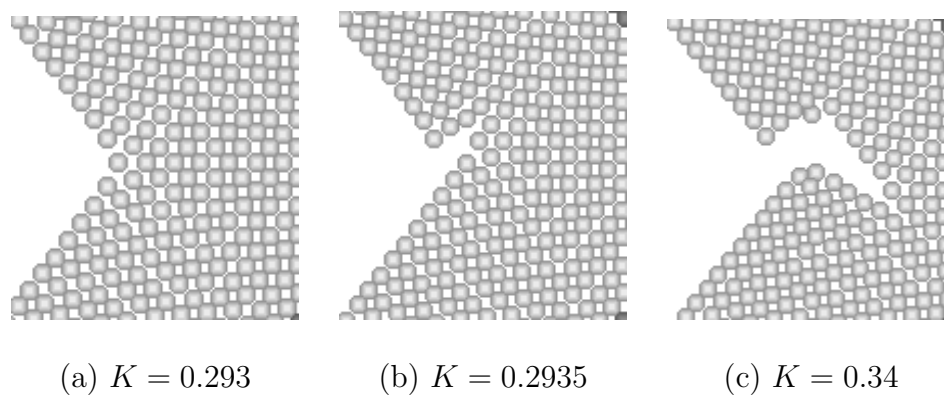


Figure 3.11: MEAM-90°.

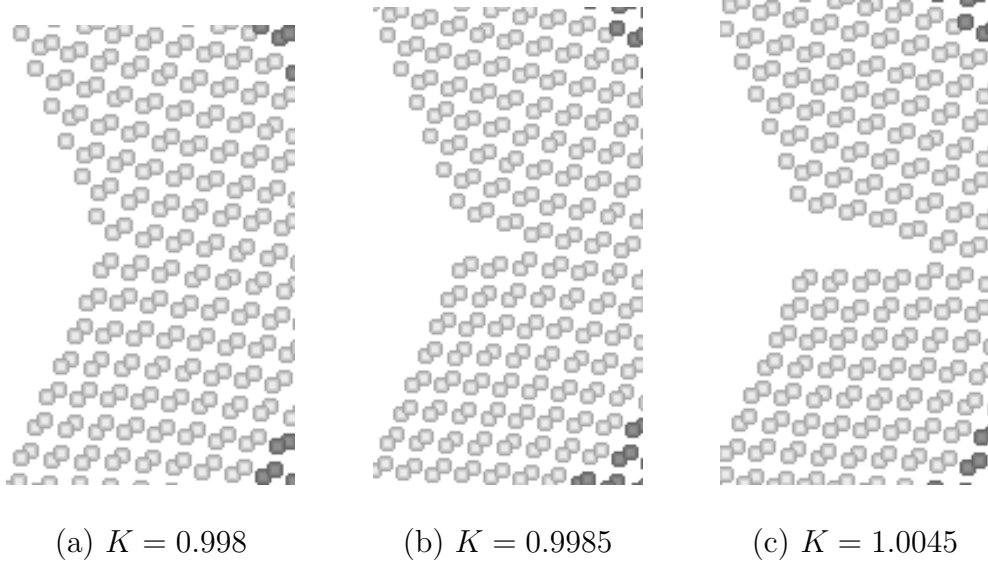


Figure 3.12: mSW-125°.

3.3.2 Critical stress intensities

The values of K_c , for the different potentials as well as from experiment, are listed in table 3.2 and plotted in Fig. 3.15. The increment size for K is listed as an estimate of the error in K_c . The values for ductile fracture from the EDIP potential are marked with an asterisk as a reminder that the definition of K_c in these cases is problematic. The experimental value for the crack geometry is from Ref. [17]. Notice that the critical stress intensities for difference angles are almost the same in atomic units, and differ by more than a factor of ten in standard units³. To check for finite size effects, we repeated the measurement on the 70° geometry, but with larger radius of 8 Å, using the MEAM potential. In this case K_c was determined to be 0.262, or about 1.7% lower than the value from the smaller system. This

³Note that the exact conversion factor depends on the eigenvalue λ which depends on the potential (see appendix 3.A), but for a given angle the dependence on potential is quite small.

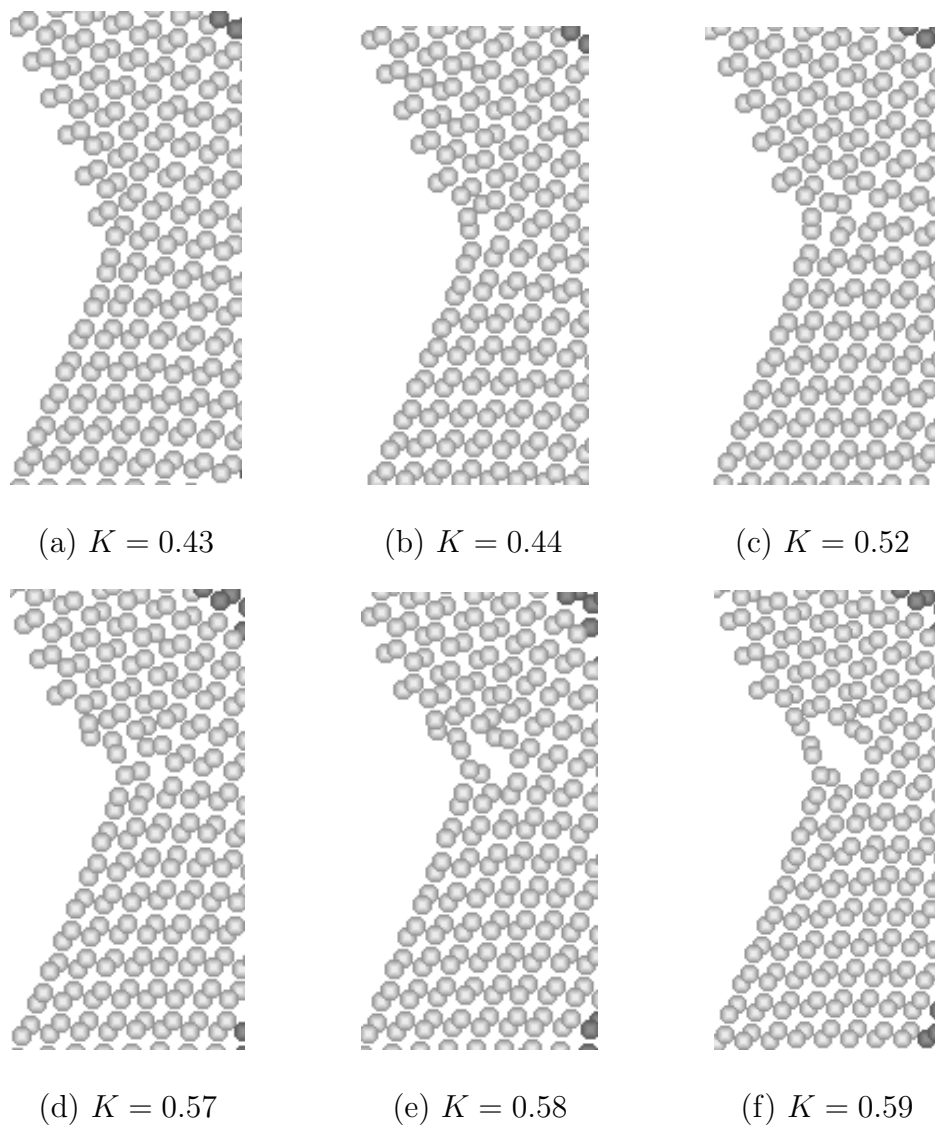


Figure 3.13: EDIP-125°.

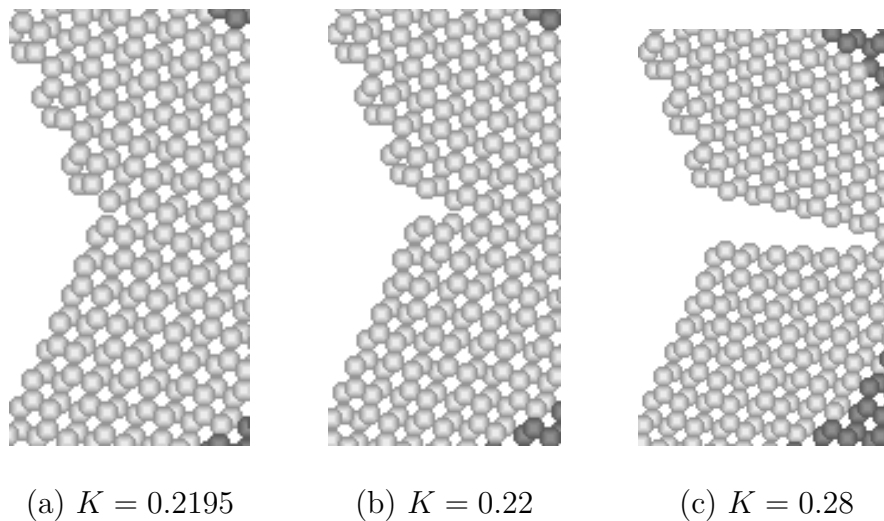


Figure 3.14: MEAM-125°.

indicates that finite size effects are small, but not negligible. To compensate for them without using larger systems a flexible boundary method as described in chapter 3 might be used, involving “multipoles” appropriate for the notch.

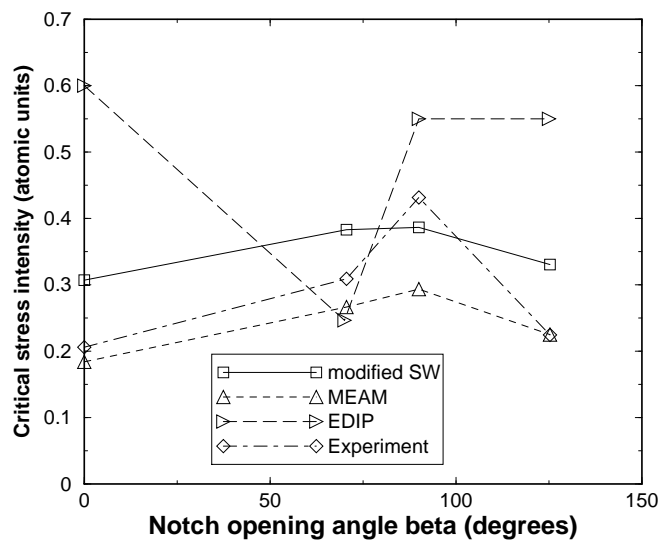
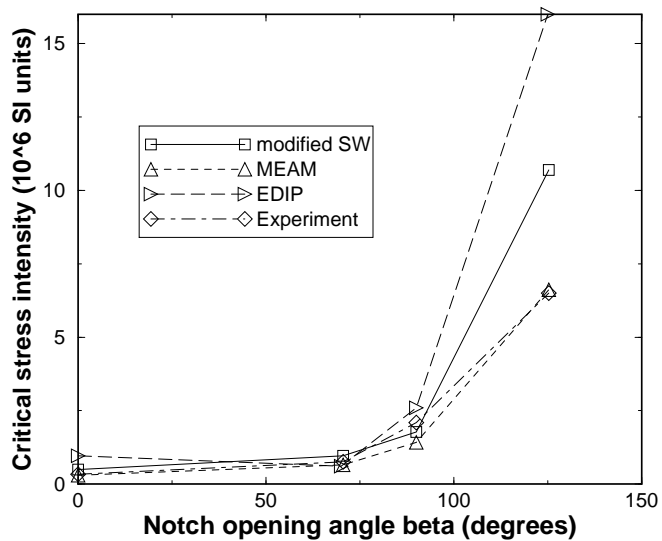
3.4 Discussion

3.4.1 Critical stress intensities

Comparisons are easier to make when looking at the data plotted using atomic scale units. Then the data for the two brittle potentials is a gentle, almost horizontal, curve. The experimental data mostly lies between that for the MEAM potential and that for the mSW potential, but significantly closer to the former. The exception is the 90° case where the experimental value jumps to higher than the mSW value. Since the curves from the two potentials are very similar in shape—the main difference seems to be an overall shift or factor—and the jump in the experimental

Table 3.2: Critical stress intensity values for different geometries and potentials, including experimental data from Refs. [66, 67].

Potential	Geometry	Atomic Units	Error	Griffith	SI Units
mSW	0	0.842	0.001	0.53543	4.9×10^5
mSW	70	1.067	0.001	-	9.6×10^5
mSW	90	1.097	0.001	-	1.78×10^6
mSW	125	0.9985	0.0005	-	1.07×10^7
EDIP	0	0.6*	0.02	0.14634	9.6×10^5
EDIP	70	0.2465	0.001	-	6.1×10^5
EDIP	90	0.5-0.6*	0.0005	-	$2.4 - 2.8 \times 10^6$
EDIP	125	0.5-0.6*	0.001	-	$1.5 - 1.8 \times 10^7$
MEAM	0	0.184	0.0005	0.16406	3×10^5
MEAM	70	0.2665	0.0005	-	6.57×10^5
MEAM	90	0.2935	0.0005	-	1.42×10^6
MEAM	125	0.22	0.0005	-	6.47×10^6
Expt	0	0.2060	-	0.1776	3.3×10^5
Expt	70	0.31	10%	-	7.6×10^5
Expt	90	0.43	10%	-	2.1×10^6
Expt	125	0.22	10%	-	6.5×10^6

(a) Units based on eV and \AA 

(b) SI units

Figure 3.15: Computed critical stress intensities for the three potentials and experiment.

value at 90° is a departure from this shape, it would not be meaningful to assert that the mSW potential does a better job in predicting the 90° - K_c . For the other angles the MEAM values are more or less within experimental error of experiment: the error (standard deviation across all the tested samples) is close to 10% in all cases (the error is not available for the crack case), and the percentage differences of the MEAM values with respect to the experimental values are -10%, -14%, -32% and -0.5% for the 0° , 70° , 90° and 125° geometries respectively. The 0.5% is clearly fortuitous. Note that the experimental error bar is not enough to account for the anomalously high value for the 90° case; there must be some feature of the physics or energetics of fracture initiation in this geometry that is missing from the others.

An interesting question is why the EDIP potential behaves unlike the other potentials and experiment except at one particular geometry, the 70° one. Possibly there is some feature of this geometry that suppresses the nucleation of dislocations. Dislocation Burgers vectors in silicon are always in $\frac{1}{2}\langle 110 \rangle$ directions, since these are the shortest perfect lattice vectors in the diamond lattice[34]. The periodic boundary conditions constrain possible dislocation lines to be out of the plane. Moreover, since we are considering only mode I and II loading, we expect slip to be within the plane, so we are considering edge dislocations. In the 70° geometry the $\frac{1}{2}\langle 110 \rangle$ direction that is available within the plane is at an angle of 90° to the x -axis, while the cleavage plane is at an angle of 35.26° . Looking at Fig 3.2, we can see that the shear stress and normal stresses on these planes respectively are both near their maximum values, although the ratio of shear stress to normal stress is 0.43 (both plots are normalized with respect to K and r). Without knowing the values of stress needed to initiate slip or cleavage on these respective planes, this ratio is not enough to explain anything. What we can do is compare the same ratio

Table 3.3: Angles of slip planes and crack planes and ratio of shear to normal stress for different geometries.

geometry	slip plane	crack plane	shear/normal
70	90°	35.26°	0.43
90	45°	45°	0.52
125	27.34°	−7.90°	0.34
0	54.6°	0°	0.37

for the other geometries and check if it is higher in other cases, thereby tending to make slip more favorable than cleavage, for a given potential (namely, EDIP). The numbers—angles for slip and cleavage and the appropriate stress ratio—are shown in table 3.3. Unfortunately, there is no conclusive trend. The ratio for 90° is indeed higher than that for 70° but the others are lower, and EDIP notches suffer plastic deformation in all of the other cases.

For the crack cases we can make a comparison of our results with the so-called *Griffith criterion* for crack propagation. This comes from setting the energy release rate equal to twice the surface energy. An expression for the mode I energy release rate in terms of the stress intensity factor is given in [58]; setting it equal to twice the surface energy leads to the following expression for the critical stress intensity factor.

$$\mathbf{K}_{\text{Griffith}} = \left(\frac{2\gamma}{\pi b_{22} \text{Im}((\mu_1 + \mu_2)/(\mu_1 \mu_2))} \right)^{\frac{1}{2}} \quad (3.3)$$

where μ_1 and μ_2 are the roots of a characteristic polynomial which depends on the elastic constants (see appendix 3.E) and b_{22} is an element of the compliance tensor

for plane strain . The ratio K_c/K_{Griffith} is associated with lattice trapping, when fracture is brittle. This ratio is 1.57 for the mSW potential and 1.12 for MEAM. These values are respectively somewhat larger and somewhat smaller than the ratio 1.25 determined by Pérez and Gumbsch using total energy pseudopotential calculations[52] (our K_c corresponds to their K_+). In the EDIP case, where fracture proceeds only accompanied by significant plastic deformation, K_c is four times the Griffith value.

In our simulations, for a given potential, only one fracture behavior is observed, in contrast to what was observed in the experiments of Suwito et al.[66]. Specifically, in the case of the 70° geometry, they observed three different “modes” (not to be confused with loading modes), including propagation on the (110) plane, yet we have observed cleavage only on $\{111\}$ planes in this geometry. It is possible that finite temperature, and the relative heights of different lattice trapping barriers, play an important role here. More likely it is related to experimental microcracks or defects near the crack tip. In any case, it would be of great benefit to systematically calculate the barriers for different processes that can occur at a notch (or crack) tip, as a function of applied load.

A further point to note, and a warning, is this: In comparing simulations involving such very small length scales (27\AA) to experiment it is appropriate to consider the question of whether the experimental notches are indeed as sharp as we have made our simulated notches. Suwito et al.[66] could only put an upper limit of $0.8\mu\text{m}$ on the radius of curvature of their notches, although notch radii of the order of 10nm have been reported in etched silicon⁴. The addition of just a few atoms right at the notch tip would presumably have a significant effect on

⁴This fact is mentioned without any citation in Ref. [67].

the energetics of cleavage initiation. The success of our simulations provides an important indication that these notches are indeed atomistically sharp.

We have not made any investigation of this, and this question should be borne in mind given the absence of experimental data characterizing the notch tip at the atomic scale.

3.5 Summary

We have determined by atomistic simulation the critical stress intensities to initiate fracture in notched single crystal silicon samples. The samples had angles of 0° (a crack), 70.5233° , 90° and 125.264° —chosen so that the flanks of the notches were low index crystal planes. These geometries correspond to those studied experimentally in measurements of critical stress intensities for fracture initiation. Of the three potentials used, modified Stillinger-Weber (mSW), environment-dependent interatomic potential (EDIP) and modified embedded atom method (MEAM), MEAM produced the most realistic behavior. The mSW potential produced brittle fracture, but its resemblance to silicon in other respects is quite weak. Except in the case of the 70° notch, the EDIP potential gives ductile fracture with a critical stress intensity factor, which is much higher than that determined using the other potentials, or by experiment.

3.6 Acknowledgments

We thank Zhiliang Zhang for inspiration and helpful discussions, and Noam Bernstein for helpful discussions. We also thank Mike Baskes for help in coding the MEAM potential. This work was financed by NSF-KDI grant No. 9873214 and

NSF-ITR grant No. ACI-0085969.

3.A Units and Conversions

Three different sets of units are used in this paper. To each atomic potential (Stillinger-Weber, EDIP, MEAM) is associated a set of atomic units (EDIP and MEAM use the same units); also we often wish to use SI units to compare to experiment. In the context of this paper there is the further subtlety that the units of the chief quantity under consideration, namely the stress intensity factor K , are not simple powers of base units but involve a non-trivial exponent λ which is a function of geometry and potential. In fact the SI units units for K are $Pa m^{1-\lambda}$ which for brevity we simply refer to as ‘SI units’ throughout the chapter.

The units for an atomic potential are determined by specifying the unit of energy and that of length (for dynamics the unit of time is determined from these and the particle mass). The SW potential as originally written down did not have units built into it. By taking the energy unit to be $\epsilon = 2.1672eV = 3.4723 \times 10^{-19}J$ and the length unit to be $\sigma = 2.0951 \text{ \AA}$, the authors modeled molten silicon[62]. However other authors[7] have taken the energy unit to be $\epsilon = 2.315eV$. The difference is not really important since we have modified the potential itself to make it more brittle so the resemblance to real silicon is reduced noticeably. We use the second scaling which seems to be more common. The EDIP potential[10, 44] on the other hand has the electron-Volt ($\epsilon = 1eV = 1.60217733 \times 10^{-19}J$) and Angstrom ($\sigma = 10^{-10}m$) built in as its units. Since $\sigma \sim Kr^{\lambda-1}$, the units of K are $[stress]/[length]^{\lambda-1} = [energy]/[length]^{2+\lambda}$, so to convert a value for K in atomic units to SI units, one uses the conversion factor $\epsilon/\sigma^{2+\lambda}$. Table 3.4 gives the factors

for the mSW and EDIP potentials and the geometries studied in this paper.

Table 3.4: Unit conversion factors for K .

Potential	geometry	λ	factor
mSW	0	0.5	584000
mSW	70	0.51954	902000
mSW	90	0.54597	1626000
mSW	125	0.63047	10690000
EDIP	0	0.5	1602000
EDIP	70	0.51922	2490000
EDIP	90	0.54708	4730000
EDIP	125	0.62844	30840000
MEAM	0	0.5	1602000
MEAM	70	0.51875	2467000
MEAM	90	0.54794	4832000
MEAM	125	0.62639	29420000

3.B Stroh formalism for notches

Here we summarize the application of the Stroh formalism to the notch problem.

More details are available in Refs. [66, 67] and [47]. We can write the solution

for the displacement field \mathbf{u} and the stress function ϕ as

$$\mathbf{u} = \sum_{\alpha=1}^6 \mathbf{a}_{\alpha} f_{\alpha}(z_{\alpha}) \quad (3.4)$$

$$\phi = \sum_{\alpha=1}^6 \mathbf{b}_\alpha f_\alpha(z_\alpha) \quad (3.5)$$

The independent variable here is the complex variable $z_\alpha = x_1 + p_\alpha x_2$. The stress function ϕ determines the stresses through $\sigma_{i1} = -\phi_{i,2}$ and $\sigma_{i2} = \phi_{i,1}$. The p_α , \mathbf{a}_α and \mathbf{b}_α come from solving the following eigenvalue problem⁵.

$$(\mathbf{Q} + p(\mathbf{R} + \mathbf{R}^T) + p^2\mathbf{T})\mathbf{a} = 0 \quad (3.6)$$

where

$$\mathbf{Q} = \begin{bmatrix} C_{11} & C_{16} & C_{15} \\ C_{16} & C_{66} & C_{56} \\ C_{15} & C_{56} & C_{55} \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} C_{16} & C_{12} & C_{14} \\ C_{66} & C_{26} & C_{46} \\ C_{56} & C_{25} & C_{45} \end{bmatrix} \quad (3.7)$$

$$\mathbf{T} = \begin{bmatrix} C_{66} & C_{26} & C_{46} \\ C_{26} & C_{22} & C_{24} \\ C_{46} & C_{24} & C_{44} \end{bmatrix}$$

The above is general within the context of two-dimensional anisotropic elasticity. To specify the notch problem we choose a form of the arbitrary function f to which we can apply the boundary conditions of the problem, which are that notch flanks are traction-free. The following choice does the trick:

$$f_\alpha(z_\alpha) = \frac{1}{\lambda} \frac{z_\alpha^\lambda}{\xi_\alpha(-\beta)^\lambda} \mathbf{b}^T_\alpha \mathbf{q} = \frac{1}{\lambda} r^\lambda \left[\frac{\xi_\alpha(\theta)}{\xi_\alpha(-\beta)} \right]^\lambda \mathbf{b}^T_\alpha \mathbf{q} \quad (3.8)$$

where $\xi(\theta) = \cos(\theta) + p_\alpha \sin(\theta)$ and \mathbf{q} is to be determined. The traction with respect to a radial plane at angle θ is given by

⁵This is not quite an eigenvalue problem in the usual sense, but it behaves very much like one.

$$\mathbf{t} = r^{\lambda-1} \sum_{\alpha=1}^6 \left[\frac{\xi_{\alpha}(\theta)}{\xi_{\alpha}(-\beta)} \right]^{\lambda} \mathbf{b}_{\alpha} \mathbf{b}_{\alpha}^{\mathbf{T}} \mathbf{q} = \frac{\lambda}{r} \phi \quad (3.9)$$

With the above form the traction condition is already satisfied on the bottom flank $\theta = -\beta$. Applying the condition on the top flank leads to a matrix equation

$$\mathbf{K}(\lambda) \mathbf{q}(\lambda) = 0 \quad (3.10)$$

The appropriate value of λ is determined by setting the determinant of the matrix equal to zero and solving the resulting equation numerically. Two values can be found corresponding to modes I and II. For a given λ , the vector \mathbf{q} is determined up to a normalization which is related to how one defines the stress intensity factor \mathbf{K} .

For the purpose of the simulations described in this paper, we calculated the Stroh parameters as follows. For each potential, the elastic constants were determined by standard methods (straining the supercell, relaxing, measuring the relaxed energy per unit undeformed volume and fitting to a parabola). This gives C_{11} , C_{12} and C_{44} , which are the three independent constants for a cubic crystal. In the formulas for the displacements and stresses given above, the coordinate system is aligned with the notch (in that the negative x -axis bisects the notch itself) and not with the crystal axes. So we must transform the elastic constants accordingly. Once we have the transformed constants we can construct the Stroh matrices \mathbf{Q} , \mathbf{R} and \mathbf{T} , and compute the Stroh eigenvalues and eigenvectors as above.

3.C Subtleties associated with taking powers of complex numbers

Some subtleties emerge when using a Mathematica script to evaluate the Stroh formulas for the notch, particularly in the context of taking powers of complex numbers. Like the C++ function *pow* for complex numbers $pow(x, y) = x^y$, Mathematica's power function takes the modulus ρ and argument θ of the base x and then uses the definition

$$x^y = \exp(y \log(x)) = \exp(y(\log(\rho) + i\theta))$$

The subtlety is the definition of the argument θ since any multiple of 2π can be added to this for a given x . The presence of extra multiples of $2\pi i$ does affect the value of the power. In C++ the argument of x is given by the function `atan2 (imag (x), real (x))`. This function always returns a value between $-\pi$ and π . Now in the formulas for the notch displacement and stress fields there are factors of the form

$$\left(\frac{\xi_\alpha(\theta)}{\xi_\alpha(-\beta)} \right)^\lambda = \exp \left(\lambda \log \left(\frac{\xi_\alpha(\theta)}{\xi_\alpha(-\beta)} \right) \right) = \exp(\lambda L)$$

where λ is real and ξ_α are complex. For now assume $\beta < \pi$ (the case $\beta = \pi$, which is a crack, is special in its own right and will be considered below). The value of the first Stroh eigenvalue for the crystal orientation in the 70 degree sample, using the EDIP potential, is $p_1 = 1.16436i$, which is approximately equal to i ; this is the case for all of the Stroh eigenvalues, except the complex conjugate ones (α even) which are approximately $-i$ (they do not all have vanishing real part). Thus $\xi_\alpha(\theta) \sim \exp(i\theta)$ is like a phase factor for the angle with respect to the x -axis—

taking the imaginary part of the log of ξ_α gives an angle equal to θ at 0 and π and approximately equal in between. By rewriting the power of the quotient as the exponential of the exponent λ multiplied by the difference of the logs, we get an expression which is formally identical, but leads to different values when evaluated on the computer

$$\exp(\lambda(\log(\xi_\alpha(\theta)) - \log(\xi_\alpha(-\beta))))$$

i.e., we take $L = \log(\xi_\alpha(\theta)) - \log(\xi_\alpha(-\beta))$. From this expression we can see that L is approximately equal to i times the angle measured from the bottom notch flank. Let us call the case where we take the log of the quotient “case Q” and that where we take the difference of the logs “case D”. In case D, the imaginary part is between 0 and 2π , whereas in case Q the imaginary part is necessarily between $-\pi$ and π . They agree in the lower half-plane, where the physical angle θ is negative, but differ by 2π in the positive- θ region. The difference between the two cases can be thought of as a difference in where we take the branch cut of the final quantity L (rather than the individual logs). Since they agree for negative θ but not for positive θ we can take a look at what happens as we pass $\theta = 0$. In case Q the value of L jumps by $2\pi i$ at this point—the branch cut is at $\theta = 0$; in case D the imaginary part of L (the real part is well behaved) is continuous all the way from the lower flank (where it is zero) up to the upper flank. In this case the branch cut is actually outside the material.

The question is then to decide which way is correct. Since we expect continuous displacements and stresses, we should clearly choose case D. So in the Mathematica script, everywhere we have this log of a quotient in the formulas, we replace it with the difference of the logs. This works also for the complex conjugate terms where

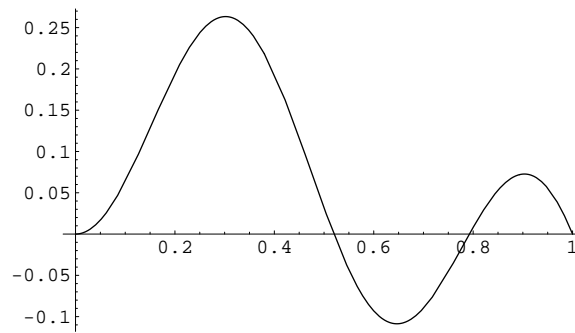


Figure 3.16: $\text{Det}(\mathbf{K})$ versus λ using difference of logs.

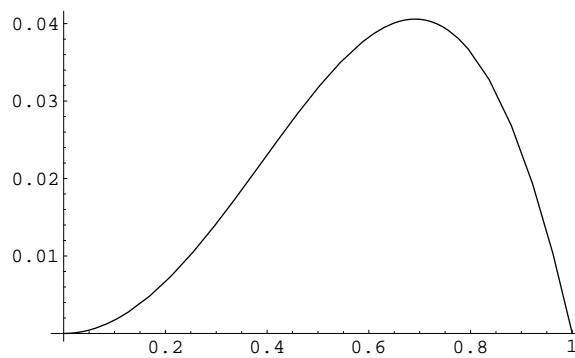


Figure 3.17: $\text{Det}(\mathbf{K})$ versus λ using log of the quotient.

$p_\alpha \sim -i$. There is one more subtlety. For the case of zero notch angle ($\beta = \pi$), when applying boundary conditions to determine the appropriate value of λ , if you blindly follow the procedure that works for finite notch angle you get zero for the matrix K , because then $\xi_\alpha(\beta) = \xi_\alpha(-\beta) = -1$. What's happening is that the notch flanks have come all the way around to meet each other, and more particularly, to meet the branch cut. Here we must recognize that we need to put in 2π by hand. (This gives us factors like $\sin(2\pi\lambda)$ in the determinant, which are zero when $\lambda = \frac{1}{2}$, as it should be for a crack).

3.D Crystal orientations

We list here for reference the rotations applied to the crystal to achieve the appropriate orientations in each case.

70 degree geometry Only a rotation of $\pi/4$ about the x -axis is needed, which makes the in-plane vertical direction and out-of-plane direction both $\{110\}$ surfaces. The in-plane horizontal direction remains $\langle 100 \rangle$.

90 degree geometry No rotation is necessary, since the notch flanks are $\{110\}$ surfaces and the plane of symmetry a $\langle 100 \rangle$ surface.

125 degree geometry The same rotation about the x -axis as in the 70 degree case is employed, and in addition to this (and following it) a separate rotation about the z -axis by an amount $\frac{1}{2} \arccos(1/\sqrt{3})$

3.E Subtlety in defining mode I/mode II in crack case (no reflection symmetry)

We have seen how there are two singular modes in the vicinity of a notch which have singular stresses. For finite notch angle the singularities are different, and thus we can make a clear distinction between the two modes of in-plane deformation, mode I and mode II. In the case of zero notch angle, i.e., a crack, the singularities are equal, namely one-half, and the analysis by which mode I and mode II are distinguished for notches does not apply: the matrix K is zero and hence cannot give a relation between the components of the vector \vec{q} . So how do we distinguish modes I and II? Traditionally these are known as the symmetric and antisymmetric

mode respectively, meaning that if one applies a reflection through the crack plane (or the plane which bisects the notch) the mode I displacement field becomes itself, and the mode II field becomes minus itself (you can also think of it as the values of the field at two points related by a reflection through the crack plane are themselves related by the same reflection, or by the reflection plus a sign change, respectively). It is possible to choose solutions which are even or odd under this reflection because the governing equations are invariant under this symmetry. In an isotropic material, and for certain planes in a cubic material, this is true. It is not true for the $\langle 111 \rangle$ plane, which is the crack plane in the case of the crack geometry. This can be seen by noticing that the elastic constant $C_{45} \equiv C_{yzxz}$, which changes sign under a reflection in the $x - z$ plane (i.e., $y \rightarrow -y$, and there is an odd number of y in this component), is not zero, thus the material is not invariant under this reflection⁶. The elastic constant is small (about 5% of C_{11}), so this is an approximate symmetry, and indeed the q_1 and q_2 parts of the displacement field are almost antisymmetric and symmetric respectively.

To define linear combinations which we will call mode I and mode II respectively, we follow Sih et al. [58], who present solutions for cracks in anisotropic bodies. Their mode I solution has the feature that the shear stress in the $\theta = 0$ plane is zero (i.e., the crack plane, ahead of the crack), and their mode II solution has vanishing normal stress in the same plane. These are of course both properties of the symmetric and antisymmetric solutions when these exist. These linear combinations mix a small amount of the q_1 field into the q_2 field for mode I, and little bit of the q_2 field into the q_1 field for mode II. The definition of the stress intensity

⁶This can be seen by staring at a cube long enough to see that this reflection doesn't take the cube to itself. If you follow it with a rotation of 60 degrees about the 111 direction then you get back the cube.

factors in Sih et al. differs from that used for the notches by a factor of $\sqrt{2}$, so a factor of $1/\sqrt{2}$ appears in their expressions for stresses and displacements. We have taken this into account when quoting their expression for the energy release rate to obtain the Griffith criterion. The numbers μ_1 and μ_2 which appear in the energy for the energy release rate are two roots of the following equation not related by complex conjugation:

$$b_{11}\mu^4 - 2b_{16}\mu^3 + (2b_{12} + b_{66})\mu^2 - 2b_{26}\mu + b_{22} = 0 \quad (3.11)$$

where $b_{ij} = a_{ij} - a_{i2}a_{i3}/a_{33}$ for $i, j = 1, 2, 6$ and a_{ij} are the elastic *compliances* (elements of the inverse of the elastic constant matrix). The subtraction is appropriate for plane strain problems, which is what our fixed-thickness periodic boundary condition corresponds to.

A slight anomaly appears in the comparison of the crack solutions of Sih et al. to those obtained from the notch formulas in the limit of a crack: the stress and displacements are not identical, even when the modes are defined appropriately as described above, and with the correct normalization. An example is shown in Fig. 3.18. The reason for the discrepancy is still under consideration.

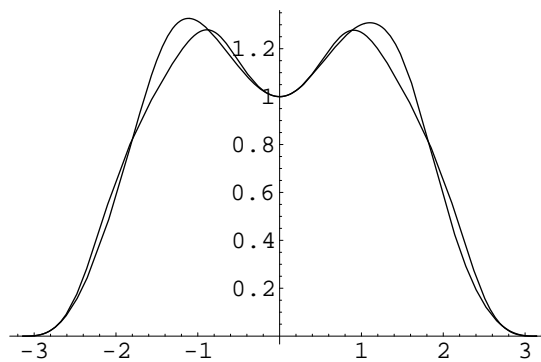


Figure 3.18: Angular dependence of σ_{yy} for the crack, using Sih and Stroh formalisms. The Stroh curve has the higher peaks.

Chapter 4

Digital Material, A Modern Molecular Dynamics Code

4.1 Introduction: The complexities of today's simulations, as driven by multiscale materi- als modeling

Simulations¹ of materials continue to become increasingly complex, driven by the need for greater modeling fidelity and the opportunities provided by advances in available computational resources. In recent years, this complexity seems to have advanced at an even faster rate, as powerful-but-unwieldy parallel computing platforms have become widely available for high-performance computation, and as

¹This chapter is based on the work of several people, and in fact will be published with the following author list: Nicholas P. Bailey, Thierry Cretegnny, Andrew J. Dolgert, Christopher R Myers, Jakob Schiøtz and James P. Sethna. Most of it has been written by N. B., apart from sections 4.1, written by C. M. 4.2.1 and 4.2.4, which were written by T. Cretegnny, subsequently edited by N. B.

researchers have reached across disciplinary boundaries to address the multiscale nature of material behavior.

Material structures and phenomena are inherently multiscale, so the desire for greater realism in materials modeling has driven a growing interest in multiscale materials modeling techniques. In some cases, these techniques explicitly link together disparate numerical models (at different length and/or time scales) to form hybrid meta-models. In other cases, simulation results from one scale are implicitly incorporated into computational models at other scales (e.g., in the form of constitutive descriptions). Investigation of material behavior across scales can involve treatment of more complex simulation geometries, boundary conditions, constitutive models, and numerical algorithms. Furthermore, optimal models and/or numerical methods are in many cases not yet known, and need to be discovered through numerical experimentation. All of these trends conspire to suggest a need for more sophisticated software frameworks to support the generation of complex material models, the construction of compound and hybrid numerical methods, the structuring of code for high performance on modern supercomputers, and the flexible control and interrogation of simulations and data.

Complexity in the multiscale investigation of materials can arise from many sources, and has many implications for the software development process. Whereas much of atomistic modeling to date has involved relatively simple simulation geometries, the desire to provide input to processes active at larger scales (e.g., plasticity and fracture) increasingly requires construction of atomistic models with more complicated geometries, involving, say, sets of interacting dislocations or a grain boundary of a specified misorientation. Furthermore, extracting useful information from small-scale atomistic simulations for use in larger scale theories

or models requires careful treatment of boundary effects. This has led to the development of hybrid numerical methods for finding the structure of atomistic defects (e.g., dislocation cores); these hybrid methods can involve both atomistic and continuum degrees of freedom which are simultaneously acted upon.

Our efforts in developing software frameworks for materials modeling fall under the general rubric of Digital Material (DM), which connotes both a general approach to software development for materials simulation and specific software systems for particular types of simulation. Our focus in this paper is on the atomistic modeling system that we have developed; we describe elsewhere [48] a related framework for the modeling of texture in polycrystalline systems. Prior to describing the specific details of the DM atomistic modeling system, we present some of the high-level design and implementation goals that characterize the DM effort broadly.

4.1.1 Digital Material design goals

We aim to build a system that is flexible, expressive, and extensible, while not sacrificing computational performance. We believe it is important to support composition of many computational modules, both to enable the construction of hybrid models and methods, and to facilitate the development of simulation software. Fortunately, there are a number of recent software engineering developments which we can exploit to build such a system.

Design patterns [24] represent an important set of object-oriented design techniques to have emerged in the last decade. These patterns address the collaborations among computational objects, in such a way as to support software change and reuse. An important element of these design patterns is that they aim to sup-

port additive rather than invasive change. That is, if a new piece of functionality is desired, it is preferable to be able to add (plug in) a new module rather than change (rip up) an existing one. Developing the correct decomposition of desired functionality to facilitate change of this sort is one of the central tasks in building such a system. As such, describing such a decomposition lies at the heart of this paper. In this introduction, we introduce some of the more general patterns which guide the overall structure of the system.

Materials simulations typically involve one or more material “samples” (instantiations of the relevant modeling degrees-of-freedom, e.g., atoms, grains, displacement fields, etc.) which are acted on by some model of a physical processes (e.g., applying loads, following a time evolution). We have therefore chosen to separate our description of material samples from the “movers” that act to modify those samples. This allows us to identify a material state independently of the models used to modify that state, and to switch different sorts of movers in and out as we develop complex models and algorithms. (In a similar fashion, computational probes which interrogate the state of a material sample are also separated out as “observers” of the underlying state.) Furthermore, we have chosen to subdivide the description of a material sample into one or more sets of geometric entities with associated sets of attributes. In particular, our `ListOfAtoms`, is *not* an array of objects of a class `Atom`, but rather is an array of positions, plus an array of velocities, etc. This is useful for several reasons. First, for reasons of performance, it is necessary to act on aggregates of data in tight numerical kernels (the “inner loops”) without the cost of higher-level overhead and control. In atomistic modeling, the positions of the atoms (which constitute the geometry of the sample) are accessed to compute neighbor lists and forces, and we wish to be able to access that

geometric information independently of other attributes (e.g., masses or velocities) for optimal performance. Second, there are other operations (e.g., visualization, or computation of local atomic coordination number) where only geometric information is necessary, and we would like to be able to extract that information without striding over all other atomic attribute data. Finally, inherent in many approaches to multiscale modeling is the need to treat different material objects in different contexts at different scales: a dislocation line, for example, is a collective and emergent feature in an atomistic simulation, while being explicitly represented as part of the computational model in a dislocation dynamics simulation. The power of multiscale modeling often lies in the ability to factor a complex description (e.g., a constitutive model at one scale) into a geometric piece and a different set of less complex descriptions: the quasicontinuum method, for example, replaces complex continuum constitutive descriptions of solids with an alternate description, involving the mutual self-organization of collections of atoms (a geometric structure) interacting via interatomic potentials (a less complex constitutive description).

Another important software engineering development which has had a significant impact on our research is the growing use of high-level interpreted scripting languages to control and steer compiled numerical simulation frameworks. Prominent examples demonstrating the value of this approach include SPaSM,[11] a system for molecular dynamics simulations of solids, and the Molecular Modeling Toolkit (MMTK)[33], for biomolecular simulations. Like these other projects, we use the Python programming language to develop high-level interfaces to our simulation kernels, and to glue together applications composed from disparate pieces (for numerical algorithms, data storage, visualization, graphical interfaces, etc.). A lightweight, programmable interface layer like Python supports our need for flexible

prototyping, control and interrogation, without impacting low-level computational performance.

4.2 Components

In this section, which makes up most of the chapter, we systematically describe the components we have identified as being logically separate pieces of a molecular dynamics code, starting with the main data structure, `ListOfAtoms`, and continuing with `Potential`, `Mover/Transformer`, `NeighborLocator`, `BoundaryConditions`, `Constraint`, `AtomsInitializer` and `ListOfAtomsObserver`. In section 4.3 we describe aspects of the code which are considered “infrastructure”: serialization, parallelization and graphics.

4.2.1 ListOfAtoms

Responsibilities

The primary responsibility of a `ListOfAtoms` is to store and provide access to the current state and properties of the atoms in the simulation. The base class stores simply the positions.

The secondary responsibilities of a `ListOfAtoms` are notifications:

1. it ensures the validity of its current state by passing on changes to the `Constraints` and `BoundaryConditions` (which may for example project the atoms back into the supercell),
2. it warns the `NeighborLocator` of changes in state (so it may need to check e.g. whether its neighbor list needs to be rebuilt),

3. it acts as a Subject which, when prompted by the user, will notify a stored list of Observers (for visualization, or any kind of analytical measurement, ...).

Examples

Velocities are not always necessary in an atomistic simulation (energy minimization), but are often important so our main derived class, `DynamicListOfAtoms`, adds velocities to the state. A more sophisticated example comes from our implementation of the quasicontinuum method, which mixes MD with finite elements. Here we have derived slave and master atoms classes, so that each slave atom can know the element it belongs to and each master atom has its own weighting factor, e.g. for the energy calculation.

Implementation, Efficiency, Flexibility

Cache performance led us to favor arrays of native types for storage. The cache on current processors loads an entire line of contiguous data whenever a value is fetched, so storing all the position information in separate array (rather than in an atom class, mingled with velocities and other attributes) reduces the number of cache misses.

Pipelining leads us to store all atoms of a given type together (and atoms subject to the same constraint together). The concurrent processing of multiple sequential instructions, is easiest for the compiler to optimize when simple tasks are repeated in regular patterns (facilitating loop unrolling, ...). Control statements (like “if (`atom.type()==..`)”) typically cause pipelines to stall. By putting the atoms of a given type together, these control statements are implemented once per

type outside the loop over atoms.

We implement `ListOfAtoms` as a tree structure, with each atom type (subject to each kind of constraint) on its own leaf. By making each branch and leaf of the `ListOfAtoms` a `ListOfAtoms`, other classes can work on subtypes of the atoms without modification (graphics, correlation functions, ...). Constraints can be applied to sublists of `ListOfAtoms` without the constraint class being aware of the surrounding atoms. We also chose to store the data for the atoms in a tree structure (the `DMArray` class), which mirrors the tree structure of `ListOfAtoms`.

Alternative Choices

Our tree-structure array class has ended up being quite complex. Some of the complexity is needed because of the need to support parallel processing. The entries in temporary force arrays in the `AtomsMovers`, for example, need to be registered with the base class so that their entries are automatically transferred to other processors when the atoms cross processor boundaries. Some of the complexity, however, could have been avoided by storing the data for all the sublists contiguously in memory. This has the disadvantage that each time atoms migrate the entire list of atoms must be shuffled up or down to make room. On the other hand, the current implementation demands extra overhead for computing the address of the neighboring atoms in force loops.

The `DMArray` class also makes heavy use of templates. In an application where all attributes of atoms were of type `double` (or `double[DIMENSION]`) this could have been avoided, making the class easier to read (but reducing the flexibility). There are also many other freely available templated C++ array classes (such as `BLITZ++`[74]), but they do not support the kinds of hierarchies in storage that

we needed here.

Finally, there are other models for notification that we could have used. Instead of having `ListOfAtoms` call `BoundaryConditions` and `Constraints` and notify `NeighborLocator` these responsibilities could have been left to a “MotherBoard” simulation class.

4.2.2 Potential

This is the modularization that is most likely to be already implemented in existing codes: most people want to have the freedom to change potentials. Different potentials represent different materials, so the number of `Potential` classes one implements is only limited by the number of materials one wishes to simulate.

Responsibilities

The primary responsibilities of the `Potential` class are two. Given a `ListOfAtoms` object and whatever arrays are necessary:

1. it calculates the current forces on the atoms, putting them into the passed array.
2. it calculates the potential energy of the current configuration, returning it as a double.

Other methods that a `Potential` may have include one to calculate both energy and forces together (there are cases where both are needed and where it is much faster to calculate them together), to calculate the atomic energy (for a given atom), the atomic stress or the Hessian matrix. Not every function needs to be implemented; they are implemented in the base class as functions which do

nothing except throw an exception. Thus client code may test the potential to see if it has the function. However at the minimum, a new potential should include `CalculateForces` and `CalculateEnergy`.

Examples

We currently have two versions of Lennard-Jones, one our own, and one by Holian et al.[36]. They differ in how they are cut-off; these are pair potentials of course. We have the Stillinger-Weber[62] potential for silicon, which includes three-body terms as well as the EDIP[10, 44] Si potential which is a many body potential. We also have the Effective Medium Theory for FCC transition metals, with our implementation specifically including Al, Cu, Ag, Au, Ni, Pd and Pt, and recently added Baskes's Modified Embedded Atom Method (MEAM) potential, with parameters for 26 elements (it needs to be modified still to allow multiple types to be simulated).

Implementation, Efficiency, Flexibility

When say, the `CalculateForces` function is called, the potential is passed the `ListOfAtoms` and an array for the forces. It gets a pointer to the `NeighborLocator` of the atoms. If it is a pair potential, it loops over all atoms, and for each one calls `HalfNeighbors` on the `NeighborLocator`, which returns the neighbors j of atom i with $j > i$ to avoid double counting. The `NeighborLocator` also returns the relative displacements (vectors and squared lengths) of the neighbors of atom i , which the `Potential` object uses to compute the corresponding pair contributions to the forces or the energy. In the case of the forces these are added to the force array for atom i , and their negatives to the force of each neighbor. For non-pair potentials, the

looping is done differently but the interaction with the NeighborLocator is similar (in some cases one calls Neighbors rather than HalfNeighbors, e.g. for the three body terms of Stillinger-Weber).

For extra efficiency, the interface to the NeighborLocator has been designed so that rather than computing all the force or energy contributions involving atom i before going onto the next atom in this loop, one can continue to fill the arrays (of displacements and squares of displacements) with neighbors of successive atoms i in the loop, until some pre-determined size limit on the array has been reached (as indicated by the return value of the NeighborLocator function which is boolean). Then if the potential involves only floating point operations these can be done faster when the data is packed into fewer, larger arrays as described. Our EMT (Effective Medium Theory) potential works this way. On the other hand this will not help if the potential involves cutoffs within the main cutoff, as in for example our CutLennardJonesPotential. Here the potential takes the usual form within the inner cutoff but has a different form between the inner and outer cutoffs. Because of this “if” statements are necessary, and so the operations are not all floating-point.

Another efficiency point is that when possible the parameters of each potential class are declared as constant variables, thus the compiler is allowed to make optimizations that it might not otherwise make. This is not possible in potentials such as EMT and MEAM where different choices of parameters are allowed.

Alternative Choices

4.2.3 Mover and Transformer

The algorithms associated with time evolution of the ListOfAtoms are encapsulated as *AtomsMovers*. Every class of this family has a function called Move which uses

the potential to evolve the `ListOfAtoms` according to the appropriate algorithm some number of time steps of some length. The `Move` function can be considered a transformation of the `ListOfAtoms`, but of a particular type—one associated with time steps and potentials, what we might refer to as “dynamics”. For all other transformations on the `ListOfAtoms` we have another family of components called `Transformers` which have a similar interface, with the function being called `Transform` in this case.

Responsibilities

The responsibilities of a `Mover` are less well defined than with components such as `NeighborLocator` and `Potential`. Particularly in the case of the `NeighborLocator`, the information returned should be independent of which implementation of a `NeighborLocator` one has used. For a potential, of course the forces and energy will vary from one potential to another but the meaning of `CalculateForces` and `CalculateEnergy` is the same for all (for instance the forces are always the negative gradient of the energy). For movers, there are not such specific requirements. Several movers implement time stepping algorithms, but it is not required that these give identical results for a given `ListOfAtoms`. Some do not even perform an operation corresponding to physical time evolution but rather perform energy minimization. Nevertheless it is understood that all `Movers` are linked to a `Potential` object, and store a value of time step, and number of steps to perform. The time steps performed in a single call to `Move` are what are sometimes called the *minor* time steps. Each call to move, made from some outer loop, constitutes a major time step.

The responsibilities of a `Transformer` are none at all, other than to provide a

function called `Transform`, to which is passed a pointer to a `ListOfAtoms`.

Examples

Our Movers include a slightly non-standard Verlet time-stepping algorithm, as well as the Gear Predictor-Corrector algorithm. For thermalized time-stepping there is a `LangevinAtomsMover` (which implements the Langevin equation with a given temperature and friction), a `HooverAtomsMover`[35], a `VerletThermalizeAtomsMover` which randomizes velocities before performing the time steps. We also implement the *QuickMin* algorithm for minimizing a molecular dynamics system as a Mover since it is closely related to the Verlet algorithm. To implement Conjugate-Gradients (CG) minimization we provide an interface class (Mediator Design Pattern) which allows a general purpose CG class to operate on a given `ListOfAtoms` using a given Potential.

Since Transformer is so general, we have many examples. Some, such as `VoidMaker`, `NotchMaker` and `OverlapPruner` cut away parts of a `ListOfAtoms` defined by some geometrical criterion. These are frequently used in conjunction with an `Initializer`, where one first creates a simple shape filled with atoms and then cuts away pieces to achieve the actual desired geometry. A subclass of Transformer is `ElasticFieldTransformer`, which covers Transformers whose transformation is associated with a displacement field as in continuum elasticity. In these the `Transform` function is passed to a separate function called `ElasticField`. This is useful in cases where one wants to be able to evaluate the `ElasticField` function without actually transforming any atoms. Also, the `EulerCoordinateTransformer` class keeps a list of `ElasticFieldTransformers` and iteratively determines using their `ElasticField` functions, the resultant displacement given by the elastic field formula interpreted as an

expression in Eulerian coordinates. Examples of ElasticFieldTransformers include (Anisotropic)DislocationMaker, which implements the standard displacement formulas for straight edge and screw dislocations in (an)isotropic elasticity theory; NotchFieldDisplacer, which implements the displacement field for a notched or cracked sample, and (An)isotropicMultipoleField which supplies the entire set of terms in the general solution for quasi-two-dimensional elastic theory in a circular geometry, apart from the dislocation (log) terms, and the terms which grow with distance from the origin.

Implementation, Efficiency, Flexibility

For movers, the implementation is for the most part as straightforward as writing out the formulas for the algorithm in terms of functions on ListOfAtoms and potential, taking care only to do use the array versions of operations on the ListOfAtoms—one should never write a `for` loop in which the positions are updated one by one. Apart from the loop overhead, each call to `SetCartesianPosition()` or `IncrementCartesianPosition()` (no `s`) entails a call to the `NeighborLocator` to update its internal variables, which will possibly involve initiating communication between processors in a parallel simulation. Clearly this is bad.

Thus the loop for `VerletAtomsMover` is not much more than a call to `CalculateForces (Potential)` followed by a call to `IncrementVelocities` and a call to `IncrementPositions` (both `ListOfAtoms`). At the beginning and end of the loops there are increments by half a time step to correctly implement the Verlet algorithm and have the velocities and positions correct and consistent when the function exits. Extra details concern the handling of constraints (see section 4.2.6). For transformers of course there is little one can say in general about implementation, but

the point about using array operations to change atomic positions is just as valid. So in the case of applying an elastic displacement for example, the increments should be computed as a separate array which is then added to the positions using `IncrementCartesianPositions()`.

4.2.4 NeighborLocator

Responsibilities

It is very natural to assign the task of identifying which atoms are located within some cutoff distance of a given atom to an individual component: a `NeighborLocator`.

It is possible to set clear responsibilities to a `NeighborLocator` and to decouple it almost totally from the other components. In principle it doesn't need to know anything about the atoms themselves, their possible constraints, or the details of their interaction: we have a well defined geometrical problem, and all the input that is needed is

1. a collection of points in space (the atoms/molecules location),
2. a cutoff distance,
3. the boundary conditions.

With this input, the `NeighborLocator` must primarily be able to return all the neighbors of a given atom. In addition, because a force calculation uses Newton's third law, the `NeighborLocator` can be requested to return only "half" of the neighbors: when looped over all atoms, this `HalfNeighbors` function returns all the bonds exactly once (for example, for atom i , `HalfNeighbors` may return all the neighboring atoms with index $j > i$). This is particular useful for pair potentials.

As a secondary responsibility, we found it extremely useful if the NeighborLocator is able to return the index of all the atoms which are located within a given distance of a given point (not necessarily an atom). It is useful because it allows e.g. extensions of a NeighborLocator to deal with type-dependent cutoffs (see below) and it is a natural responsibility since a NeighborLocator usually stores the information that is needed to answer this question (like a cell list).

Finally, for efficiency reasons as well as for flexibility, we also included in the NeighborLocator's tasks the possibility of returning only the neighbors of a certain type.

Examples

The most trivial example of NeighborLocator is the one that tests each time all the atoms whether their separation is shorter than the cutoff distance. A force calculation using this type of "SimpleNeighborLocator" would be $O(N^2)$. It has the advantage that it certainly works under any circumstances. In addition having such a simple component is handy when the number of atoms we have to work with is small.

In the opposite case, however, more clever methods must be implemented. The principles of such tricks like Verlet neighbor lists and cell lists are well documented in textbooks [3]. However making them as efficient as possible and decoupled enough from the other components is oftentimes delicate (see below).

Separating a NeighborLocator component from the others is extremely useful because it can be used for far more than just the regular force calculation between atoms. One could use it for other tasks, like the localization and visualization of crystalline defects based on the coordination. A NeighborLocator could also

perform more sophisticated tasks. For example we may want to break a subset of the atomic bonds (e.g. remove atoms from the neighbor list which lie across a half-plane in order to open a crack from a crystal). This could be done by a specially designed NeighborLocator that would post-process the calculations of any NeighborLocator (it would “decorate” another NeighborLocator, in the Design Pattern’s language); there would be no need to dig in, or rewrite the Potential.

Implementation, Efficiency, Flexibility

The NeighborLocator is primarily used by the Potential to calculate forces and energies. It is a means to make the force calculation more efficient by avoiding unnecessary computation. In addition, the Potential generally must also compute the vector separating a pair of interacting atoms, as well as their separating distance. However these quantities are also computed by the NeighborLocator so it is very natural that an inquiry for the neighbors of an atom, say i returns at least:

1. the index of the neighboring atoms,
2. the separation vector between atom i and its neighbors,
3. the squared distance between atom i and its neighbors.

The NeighborLocator generally stores some internal data. Depending on the concrete type of NeighborLocator, this could be a list of neighbors for each atom, or a cell list (a region enclosing the atoms is decomposed into cells, associated with each of which is a list of the atoms it contains). This data may become invalid after the atoms have moved too much; at this time, the NeighborLocator must rebuild its data. A nice way to make sure that the data is always up to date is to implement a Subject/Observer-kind of relationship between the atomic positions

and the NeighborLocator: each time the positions are changed a signal is sent to the NeighborLocator that their “subject” was modified and that it must check whether its internal data is still accurate. This signal typically is an “Update” function; a SimpleNeighborLocator would do nothing, but a NeighborList would compute the maximum atomic displacement since the last building of the neighbor list and decide whether or not the list should be rebuilt.

As we said many times earlier the overall goal is to make the components of our MD program as independent as possible and concrete instances of a component as interchangeable as possible. This means in particular that a NeighborLocator should work without the knowledge of which kind of boundary conditions is in effect (of course the associated overhead must be acceptable). If we want to build a cell list (this is also true for a neighbor list, because to make a neighbor list, it is more efficient to use a cell list), one must be careful to correctly identify which cells are within the “sphere of influence” of another. In fact this question is so closely related to the original one, that a general method is to make no assumption about which cells are neighbors and delegate the determination of neighboring cells to another “upper” NeighborLocator with a suitable cutoff. The center of the non-empty cells (squares/cubes or rectangles) are given to the upper NeighborLocator, which a cutoff c_{up}

$$c_{\text{up}} = c + \text{Diagonal of the cells}, \quad (4.1)$$

where c is the original cutoff. This way one builds a whole hierarchy of NeighborLocators until the number of cells is small enough that a SimpleNeighborLocator can be effectively used. The nice aspect of this approach is that it works with any kind of boundary conditions, even if images (via the boundary conditions) of cells

overlap.

Note that if two cells are separated by a distance less than the cutoff c_{up} , they are not necessarily relevant neighbor cells (i.e. it is possible to make sure that no atoms they contain can be closer than c). So it is possible to get rid of some of these neighboring cells by applying a criterion based on the vector between the center of one cell and the corners of the other.

Alternative Choices

The most efficient NeighborLocator is called CellNeighborList, since it uses a cell list to construct neighbor-lists which are then used to provide neighbor information. Neighbor-lists have a large number of ints or pointers per atoms and tend to dominate the memory requirements. It was thought that CellNeighborLocator, a neighbor-locator based on cell lists along, would be quite useful for larger systems, but while it does take dramatically less memory, there are so many candidates for being neighbors that it is too slow. The problem is that one needs not only to search the current cell (say, with a cell length equal to the cutoff distance) but 26 other cells surrounding it. So, instead of $\frac{4}{3}\pi r^3$, the volume to be searched is $27r^3$, a ratio of almost 6.5.

There are still unresolved problems in the case of time-dependent boundary conditions, for example when a periodic supercell is sheared. This can cause unnecessary rebuilds of the neighbor-lists. Ideally one should allow for the shearing of the old positions in computing the maximum distance moved.

4.2.5 BoundaryConditions

Responsibilities

The boundary conditions have basically two clear responsibilities.

1. They must make sure that the position of every atom satisfy the boundary conditions: we later refer to this task as *EnforceBoundaryConditions*.
2. They must determine the actual separation between a pair of atoms, identifying their closest images through the boundary conditions. We'll call this *DifferenceBoundaryConditions*.

Examples

The traditional type of boundary conditions employed in an MD simulation is `PeriodicBoundaryConditions`. We have implemented a few variants of these, offering for example the choice to switch off wrapping in certain directions, or allowing a general parallelepiped shape rather than a rectangle—this being useful for studying systems under shear for example, or studying surfaces of various orientations.

The simplest kind of boundary conditions is the kind that does nothing at all, known as `FreeBoundaryConditions`. This is useful in simulations where a region of freely moving atoms is surrounded by a region of fixed or constrained atoms which provide effective boundaries. Examples of the use of `FreeBoundaryConditions` form the subjects of Chapters 2 and 5.

Implementation, Efficiency, Flexibility

Because of the cache limitations (and because the functions are typically virtual) it's much more efficient to enforce the boundary conditions on large contiguous

arrays of data. The same is true for `DifferenceBoundaryConditions`: it's a good idea to stack the requests for the distance to the closest image of an atom and compute them all at once.

Alternative Choices

A frequently used technique in MD when using periodic boundary conditions is to store “scaled” positions which take values between zero and one in each direction. This allows certain tricks to be used in applying periodic boundary conditions, such as adding and subtracting a “magic” number which has the effect of putting a value outside the interval $(0, 1)$ back into it in the appropriate way². This avoids the need for “if” statements which in principle are detrimental to performance. It has the disadvantage that positions must be re-scaled in order to compute energies and forces. In our experience the performance difference has been negligible, and we feel that using real space coordinates is simpler and more intuitive.

To use the most general kinds of boundary conditions, one may have to add some responsibilities to the `BoundaryConditions` component. For example one can think of some complicated boundary conditions for which the two positions must be provided to get their separation vector (not just the difference between them). Another example is reflecting boundary conditions, where the *velocities* of the atoms should be changed (note however that this kind of `BoundaryConditions` could also be implemented as a `Constraint`).

²This is processor dependent.

4.2.6 Constraint

The simplest MD application uses periodic boundary conditions (PBC). There are tricks one can use to allow application of stress or strain to the system using PBC, but there are often cases, particularly when an atomistic simulation is coupled to a larger length scale simulation, when more physical boundary conditions are necessary, such as a layer of atoms which is fixed, or moves uniformly, or has a constant force on it. To effect such behavior we add a *Constraint* to the appropriate branch of the ListOfAtoms.

Responsibilities

There currently exist two interfaces to Constraint objects. They are not both implemented for all existing Constraints, because the nature of some constraints makes it difficult to implement one or the other; in some case it is even ambiguous how exactly a given interface should behave for a particular constraint (e.g., the generalized coordinates for a FixedCenterOfMassConstraint, where one would have to make an arbitrary choice in order to define the generalized coordinates).

The “Adjust” Interface This is the most commonly used method of incorporating the constraints. Constraint classes provide methods to Adjust position, velocity and force arrays. As long as these functions are called (for example by a mover) any time that the positions and velocities are incremented or set, or any time that the forces are calculated, the appropriate Adjust function will ensure that the positions satisfy the constraint, and the velocities and forces are such that the positions continue to satisfy the constraint when the velocities or forces are used to increment them. Some constraints deal explicitly with velocities and forces rather than positions (e.g. Uniformly-

MovingBody and ExternalForceConstraint) . These are not constraints in the usual mechanics sense, although if the velocities (or momenta) are considered on an equal footing with positions, then a velocity constraint has an equivalent status to a position constraint. A constraint on forces—our example involves the addition of additional force, as in an *external* force—is in principle really a change of the potential, but it is more easily implemented as a constraint using the Adjust interface.

The Generalized Coordinates Interface This interface is less well developed; it more explicitly corresponds with the classical notion of a constraint on dynamical degrees of freedom (DOFs). To use this interface means rather than using the standard interface with the ListOfAtoms (IncrementCartesianPositions, IncrementCartesianVelocities etc) and calling instead IncrementGeneralizedCoordinate and CalculateGeneralizedForces. At this time there is no method to access the generalized velocities, which if one was properly using generalized coordinates would be stored instead of, or in addition to actual velocities (which would be slaved to the generalized velocities).

Examples

The base class Constraint is itself a valid constraint—it is the null constraint, in that its Adjust functions do nothing. A ListOfAtoms is born having its Constraint object be a base class instance. To set a new constraint, the new constraint is created externally, then passed in a call to ListOfAtoms::SetConstraint(), which deletes whichever constraint was previously there, and reassigns the LOA's pointer.

The most frequently used Constraint³ is FixedBody. When this is attached

³at least by the author

to a `ListOfAtoms`, subsequent calls to change the positions will have no effect. The `Adjust` interface simply stores the initial positions and whenever the positions are changed and `Adjust` is called, they are changed right back. The `GeneralizedCoordinate` interface exists in that it returns zero as the number of independent DOFs and thus allows no changes to the positions.

For simulations which involve applying traction to the surface of a sample, one separates the outer few layers of atoms into a separate branch of the `ListOfAtoms` tree, and adds an `ExternalForceConstraint` to that branch. Only the `AdjustForces` method does anything; it adds a given external force divided by the number of atoms in the branch to each atom in the branch. A weakness with this is that if one called `AdjustForces` more than once per call to `CalculateForces` from the `Potential`, the actual extra force would be several times what was intended. Ideally an `Adjust` function should be idempotent—calling it several times should have the same effect as calling it once.

Implementation, Efficiency, Flexibility

Movers do not communicate with the constraint directly. In the case of `AdjustPositions`, nothing extra needs to be done because this is handled by the `SetCartesianPositions()` and `IncrementCartesianPositions()` methods. These call an `AdjustPositions()` method of the `ListOfAtoms`, which does two things: calls the corresponding method on its own `Constraint`, and calls the corresponding method on its branches. `IncrementVelocities()` works the same way⁴. `AdjustForces` is called by Movers, since it is they that own force arrays. Again first the method of the LOA's own `Constraint` is called on the whole list, then the same method on each

⁴Although it is not automatically called by `Set/IncrementCartesianVelocities`—not sure why not.

branch of the LOA is called on the corresponding branch of the force array.

We have to add an additional Adjust function to allow certain movers to behave properly: AdjustForceIncrements. This is applied to for example the random amounts that are added to the forces by LangevinAtomsMover, in order to zero out or average certain components of the force array. It is mostly identical to AdjustForces, except for ExternalForceConstraint where it is important not add the external forces—since these are separately added to the force array, and must only be added once!

Alternative Choices

The Constraint component is very incomplete. It is easy to imagine constraints which cannot easily be implemented using either interface and movers which would break the present mechanism for enforcing constraints. Also for each currently existing constraint one can imagine possibly more efficient alternative implementations. A case where the present mechanism is demonstrably weak is the FixedBody constraint. The AdjustForces function of this constraint zeros the forces for a ListOfAtoms with this constraint. This is clearly necessary if one is using LangevinAtomsmover because the algorithm for Langevin dynamics has a step which requires incrementing the positions by an amount proportional to the forces. However in some applications one might want to know the forces on those atoms for reasons other than to move them—these being the reaction forces required to hold these atoms in place, which are often of interest. In particular, sometimes the atoms in the FixedBody *do* move; they are not moved by an MD algorithm, but rather by an external object, such as an ElasticFieldTransformer whose parameters are being evolved by some high level algorithm. The high level algorithm

might need to calculate forces on those parameters, which forces are a function of the atomic forces. To avoid zeroing the forces one might make a separate call to calculate forces and not call `AdjustForces()`. However this might result in unnecessary repetition of the force calculation. Furthermore if the high level procedure is taking place in Python, it may not be possible to control whether the constraint is applied or not—the current interface to `CalculateForces` for all `Potential` objects, from Python, includes a call to `AdjustForces`. To get around this one needs to remove the `FixedBody` Constraint (which is done in practice by replacing it with a base class `Constraint` object). However this requires the user to be more aware of the details of what Constraints are attached to what `ListOfAtoms` objects than is desirable, although maybe not more aware than is necessary; that is to say, such specificity may be unavoidable.

It may be safer to give the constraints more responsibility instead of having the post-processing step we currently use. So rather than `IncrementPositions` and then `Adjust`, pass the `Increment` call on to the constraint so that it can handle the entire process. This would speed up the `FixedBody` for example, because then the `Constraint` could simply not pass on the instruction—which saves time on incrementing and then resetting to the original positions. The same could be said for `UniformlyMovingBody`.

For `ExternalForceConstraint`, it would be safer to pass the `CalculateForces` command to the constraint which would calculate the potential forces from scratch and then the external force. Subsequent calls would repeat this rather than accumulate many times the desired external force.

Rather than having two interfaces within one class, it might make sense to separate them into two different classes. This has been done in other MD implementa-

tions by Jakob Schiøtz, where the GeneralizedCoordinate interface corresponds to a “Filter” class. Other objects, such as minimizers interact with the ListOfAtoms through a Filter object which connects the generalized coordinates to the actual atomic positions.

4.2.7 AtomsInitializer

While not as crucial as Potential and NeighborLocator, a certainly significant component of our software is the AtomsInitializer. Since for the most part, initialization is something that occurs once in a simulation, efficiency is not the key issue here. The purpose of AtomsInitializer classes is to save *user time* rather than *computer time*. If one were to have to think about the coding details of getting orientations of crystals and axes right every time one wanted a new simulation, one might be tempted not to change the simulation geometry very frequently. The key benefit of having a set of Initializer classes implemented is that with very little work—generally a few lines of a python script—one can set-up a wide variety of configurations. The flexibility comes from separating the lattice to be used for filling from the shape defining which region of space is to be filled. Further flexibility derives from the facility to compose different shapes in various ways.

Responsibilities

An AtomsInitializer must possess a *Create* function. An empty ListOfAtoms is passed to the Create function after which it is no longer empty but has a number of atoms and a set of positions and velocities determined by what type of initializer it is, and what parameter values were passed to it. Typically one wants to fill regions of space with atoms. For solids typically studied using atomistic modeling these

are in a crystalline array. This necessitates a lattice class of some sort. Our lattice class is called *BravaisLatticeWithBasis* and gives a general crystalline lattice, with arbitrary lattice vectors and arbitrary number and positions of atoms within a unit cell. Subclasses with specific lattice vectors and bases have been defined for the common lattices: SimpleCubic, FCClattice, DiamondLattice etc. Methods include rotations and translations, operations to convert between real coordinates and lattice coordinates etc.

The most important type of Initializer is the *ClusterInitializer*. This is a base class for several different concrete subclasses (see below for examples). The base class provides the Create() function, and takes a BravaisLattice as a constructor argument. The subclass must provide an Inside() function, which takes a point in space and returns a boolean value if the point is inside the region to be filled.

If the initial state of a simulation is not a homogeneous crystal occupying some region of space, but perhaps has a strain field of some sort applied, or has one or more defects (dislocations, notches, cracks, vacancies, etc.) one uses an initializer such as a ClusterInitializer and subsequently applies a Transformer to the ListOfAtoms. Transformers are designed to make geometrical and topological changes to an already existing ListOfAtoms; see section 4.2.3 for more on Transformers.

Implementation, Efficiency, Flexibility

Since, as we have said, initialization is only performed once, efficiency is not as crucial as with the NeighborLocator or the Potential. However there is one important place where some thought can be usefully spent in order to reduce start-up time. The Inside() of a ClusterInitializer can only say whether a given position is or is

not within the region to be filled. It does not by itself give suggestions for candidate positions. Since we cannot loop through every lattice position in space we need a reasonably good estimate of a bounding region which has a simple shape, which can be looped over, passing each lattice position to the `Inside()` function to be tested. This bounding box is defined by the values of the arrays *maxSize* and *minSize* which are computed by the function `SetMaxSize()`, which must be implemented separately in each subclass. These arrays specify upper and lower bounds for indices (corresponding to lattice vectors) to take for the purposes of looping to find candidate positions. Care has to be taken to ensure that the region of space thus defined definitely includes the region to be filled, regardless of how skewed the lattice vectors are. Another point of care here is the proper treatment of positions on the boundary itself. This is important for example when filling a rectangular region with atoms and applying periodic boundary conditions to that region—it is very easy to end up having two different atoms occupying sites which are equivalent by periodicity, so the two atoms are effectively on top of each other (this is bad!). We treat these situations by not including sites near the “negative” boundary, while including sites near the positive one.

Further flexibility is achieved with the `CompositeClusterInitializer` class. This allows one to have as the region to be filled the union, intersection or difference of two other regions, defined by other `ClusterInitializers`. This is very useful in mixed atomistic-continuum simulations where the central region of atoms is a sphere or disk or cylinder, and this is to be surrounded by a shell or annulus of constrained atoms. The `Inside()` function of the `CompositeClusterInitializer` performs the appropriate boolean operation on the results from the `Inside()` functions of the other `ClusterInitializers`.

Examples

Examples include

- `SphericalClusterInitializer`: fill a sphere of given center and radius.
- `RectangularClusterInitializer`: fill a rectangular region with sides parallel to coordinate axes given center and lengths of the sides.
- `CylindricalClusterInitializer`: fill a cylinder of given center, radius and height.
- `PolyClusterInitializer`: fills a simple (by default convex) polygon or polyhedron in two or three dimensions respectively.
- `ParallelepipedClusterInitializer`: fills a parallelepiped or specified edge vectors and center.
- `AtomicSurfaceInitializer`: this was designed for a more specific application, measuring surface energies. It sets up layers of atoms of specified thickness and orientation relative to crystal axes. It also sets the lengths of the boundary conditions appropriately.

4.2.8 ListOfAtomsObserver

A key innovation of our work is the separation of core computation from measurement. The code for these is often found together but it does not need to be. One should be able to code an `AtomsMover` without thinking about what measurements are to be made on the system during a simulation run. To implement this separation we have used the so-called *Subject-Observer* Design Pattern [24]. For the purpose of this pattern, a *measurement* is any function or operation that looks

at the simulation's (`ListOfAtoms`'s) data but does not modify it. This includes statistical averaging of various kinds, graphics and visualization, saving simulation states to disk and others.

Responsibilities

The subject observer pattern designates one class, in our case `ListOfAtoms`, as a *subject*. As a subject, `ListOfAtoms` has a function `AddObserver()`, to which a `ListOfAtomsObserver` (referred to as `Observer` for brevity) object is passed. A pointer to it is then kept on a list by the `ListOfAtoms`. The other part of the pattern on the `ListOfAtoms` side is a function called `Notify`. A subclass of `Observer` must provide an `Update` function. When `Notify` is called on the `ListOfAtoms`, it iterates over the list of attached `Observers` calling each `Update` function in turn, passing itself (the `ListOfAtoms`) as an argument. The observer can then access data of the `ListOfAtoms` and perform its task. It may not alter the `ListOfAtoms`.

Implementation, Efficiency, Flexibility

The implementation of the Subject-Observer pattern in the Digital Material follow Ref. [24] quite closely. The `Notify` function is generally called at the outer level as part of the outer loop. Thus suppose one wants to run a simulation for 10000 time steps, recording the potential and kinetic energy every 100 steps. One first would attach an `EnergyObserver` to the `ListOfAtoms`, set the number of steps in the `AtomsMover` to be 100, and then have an outer loop with 100 iterations, in which one calls first the `Move` function of the `Mover` and then `Notify()` on the `ListOfAtoms`.

An enhancement to the standard `Observer` pattern that we have implemented

is to associate an integer, called the *notify level*, to each observer as it is created. The default value of the notify level is 0. The `Notify()` function of `ListOfAtoms` now takes an integer argument, called `notifyLevel`, also with a default value of zero. For nonzero argument, only those observers whose own notify levels are less than or equal to the `notifyLevel` argument of `Notify()`, are actually updated. For example, suppose we want to measure the energy for the purposes of averaging (other some other function of the atomic state) every 10 time steps, and save the atomic configuration to a file every 100 time steps. These processes would be handled by say an `EnergyObserver` and a `FileObserver`, respectively. We could create the `EnergyObserver` with a notify level of 0 and the `FileObserver` with a notify level of 1. In the main loop, having set the minor time step to be 10, when the major time step is a multiple of 10 we call `Notify` with an argument of 1, otherwise we call it with an argument of zero. Alternatively we could imagine a more complicated high level control structure with nesting of loops, the inner one(s) calling `Notify` with argument 0 and the outer one(s) with argument unity. In general if an observer is not to be always updated, it should have a notify level greater than zero.

Examples

- `EnergyObserver`: measures potential and kinetic energy and stores totals for the purpose of statistical analysis of these quantities (mean, variance and related quantities).
- `PlotAtomsObserver`: plots atoms using the `PlotAtoms` package.
- `RasMolObserver`: same using the `RasMol` package

- CheckPointObserver: a simple implementation of saving the state of a simulation to disk, using Serialization (see 4.3.2 below).
- StressIntensityObserver: for crack simulations, it computes an estimate of the stress intensity factor around a crack.
- PythonLOAObserver: this is very important as it allows new observers to be defined purely within Python and be attached and Notified exactly as if they were C++ observers. The Python observer creates an instance of this class, passing a pointer to its own Update function.

4.3 Infrastructure

4.3.1 Parallelization

It is clear that any modern MD code must be parallelizable. Modern scientific computation is relying more and more on clusters of processors, especially as it becomes easier to build these from “off-the-shelf” hardware. Furthermore, MD as used for materials modeling lends itself very well to parallelization since interactions are typically short-ranged. Thus if different processors handle distinct regions of space, a given processor needs only to know about the positions of atoms on “neighboring” processors which are within a cutoff distance of itself. For a large enough number of atoms per processor, this is a reasonably small fraction of a processor’s atoms whose positions need to be communicated to other processors each time step.

We have implemented parallelization in a way which is almost transparent to the user. To make an application (either a `main` function or a Python script) run

in parallel, typically only a few lines have to be added. These involve creating parallel versions of `Potential`, `NeighborLocator` and `AtomsInitializer`, which wrap the serial versions of these—thus, it is still necessary to create the ordinary serial object. But then one passes its address to the parallel version and from then on refers to the parallel version (e.g. the `AtomsMover` is given the `ParallelPotential`, and the `ListOfAtoms` is given the `ParallelNeighborLocator`).

To implement parallelization we need the following:

1. A means of defining which atoms belong to which processors
2. A means of distributing atoms among processors
3. A way for a processor to know the positions of atoms on other processors that it needs to properly compute forces on its own atoms
4. A way to redistribute atoms between processors when their positions have changed sufficiently, which must also take redistribute corresponding items in related arrays (velocities, forces, etc.)

Ghost Atoms and Synchronization

All the atoms of the system are distributed somehow among the processors, such that each atom “belongs” to exactly one processor. The definition of “belongs” is assigned to the `DomainSubdivision` abstract class, which so far has one subclass, `HomogeneousDomainSubdivision`, which divides a rectangular space equally into domains, and associates each domain with a processor. A processor is responsible for computing the forces on all of its own atoms, as well as updating velocities and positions. All processors are running the same program, and thus work with `ListOfAtoms` objects of identical tree structure. However the number of atoms on

each leaf will differ from processor to processor. The function `GetNumber()` will return just the number for that processor—to get the total number of atoms, an `AllReduce`-type operation must take place. To calculate forces, a processor needs to know the positions of certain atoms belonging to other processors. Such atoms are called “ghost atoms” from that processor’s point of view. The process of obtaining information about ghost atoms (their amount, types and positions) is called `Synchronization`, and is handled by the `Synchronizer` class. This information is accessed by a given `DMArray` through the `SynchronizationKit` class, which provides an extension of a `DMArray`, containing separate `DMArrays` `ghostAtoms` (which will actually contain the ghost atoms’ data), and `extendedArray`, which exists to be a parent array for the given array and `ghostAtoms`. It the `extendedArray` which eventually is seen by the serial `NeighborLocator` on each processor. `Synchronizer`’s main methods are (1) `Rebuild()`, which constructs for the current processor the lists of its own atoms which are “interesting” to each other processor—it consults the `DomainSubdivision` for the definition of “interesting”—and (2) `Synchronize()`, which communicates the positions of the interesting atoms to the appropriate processor.

Migration

The process of transferring atoms between processors when their positions have changed appropriately is called `Migration`, handled by the `Migrator` class. We need to be careful about what it means to “transfer atoms”. In some OOP implementations of MD, there is an `Atom` class, which contains the position, velocity, force, mass etc. As discussed in subsection 4.2.1 this is not efficient when it comes to updating the positions or the velocities, but it would make more obvious what block

of data should be transferred to the other processor. We could say that what ever arrays are in a given `ListOfAtoms` should be transferred, but that would leave the force arrays, which are managed not by the `ListOfAtoms`, but by the `AtomsMover`, unchanged. The mechanism we have designed to automatically handle the transfer of all atomic data between processors is called the *sibling mechanism*. `DMArrays` which are siblings of each other will all be migrated when one is—the one being the position array, typically. The information about what siblings an array has is stored in the `SynchronizationToken` class. An array is made a sibling of a previously existing one if the old array is passed to the constructor of the new one. The `Migrate()` function of `Migrator` assembles data to be communicated, from all its sibling `DMArrays`, in the communication buffers, makes an `AllToAll` communication, transfers the newly received data from other processors to its own `DMArrays`. Finally it calls the `Synchronizer`'s `Rebuild()` function to recreate the lists.

For the user: parallelization wrapper classes

A user writing a Python script or main function for a parallel simulation must create instances of `DomainSubdivision`, `Synchronizer` and `Migrator`, and as well of parallelization “wrappers” around the usual `AtomsInitializer`, `NeighborLocator` and `Potential`:

`ParallelAtomsInitializer`: The regular `AtomsInitializer` (e.g. `RectangularClusterInitializer`) is passed to this as a constructor argument. When `Create()` is called, the regular `AtomsInitializer`'s `Create()` is called on processor zero only. On other processors, copies of the resulting `ListOfAtoms` with the same tree structure, but empty leave, are created. The first call to the `NeighborLocator` to update will lead to atoms being `Migrated` for the first time. This

happens before any neighbor lists are constructed, so processor zero will not need to provide the large amount of memory that storing these for the whole system. However, the fact that it briefly stores the positions of all other atoms does put an eventual limit on the scalability; once the total number of atoms approaches 10^8 , the procedure would probably have to be modified.

ParallelNeighborLocator: Wraps around the regular `NeighborLocator`. Its `UpdateMyData()` function calls the `Migrate()` and `Synchronize()` as well as the regular `NeighborLocator`'s `UpdateMyData()`. Also, its `SetPositionArray()` causes arrays to be allocated for ghost atoms' positions, calls `Migrate()` and `Synchronize()`, and calls the regular `NeighborLocator`'s `SetPositionArray()`, passing it the `extendedArray` (i.e. including the ghost positions) obtained from the `SynchronizationKit`.

ParallelPotential: Wraps around the regular potential. The only extra tasks it performs are to sum the energy from all processors, and to ask the `SynchronizationKit` to allocate the ghost-atom arrays.

Interaction with MPI through DMProtocol

We have used MPI to send messages between processors, but have added a layer of abstraction between Digital Material code and MPI, known as the Protocol, so that the use of MPI is not hard-wired into the code. The base class `DMProtocol` provides an interface with methods such as `Broadcast()`, `AllToAll()`, `GetNumberOfProcessors`, `GetProcessorNumber()`, etc. These are overloaded in subclasses `MPIProtocol`, which connects these methods to actual MPI calls, and `LocalProtocol`, which has trivial implementations of the above methods for use in serial operation (i.e., return 0 for processor number, 1 for number of processors, do noth-

ing for other methods, etc.). General DigitalMaterial code only ever knows about a global pointer, `dmProtocol`, to the base class `DMProtocol`. This points to the Protocol object currently in use. Creation of the Protocol object is controlled by the class `ProtocolFactory` (Singleton pattern—only one object ever exists), whose `SetProtocol()` method allows different Protocols to be set. At present the only protocols are `MPIProtocol` and `LocalProtocol`, but the system could handle a different message-passing system if one were available. When using Python for a parallel application, a special version of the python executable must be used, known as `mpipython`. This is necessary in order that `MPI_Init()` be called before anything else. When the `ProtocolFactory` is instantiated (this happens statically) it checks if `MPI_Init()` has been called; if so it sets the Protocol as `MPI`. This will be the case in a Python application; the user need not do anything. In a pure C++ application, near the start of `main()` should be a call to `MPI_Init()` followed by a call to the `ProtocolFactory::SetProtocol()`, passing “`ProtocolFactory::World`” which is an enumerated type representing `MPIProtocol`.

Parallelization and new Digital Material classes

For the researcher who wishes to write a new interatomic potential within the DigitalMaterial framework, there is very little that needs to be kept in mind for the purposes of parallelization, since the `ParallelNeighborLocator` takes care of most details. The main thing is to be aware in force calculations that some of the neighbors returned by the `NeighborLocator` will have atom numbers apparently too high (i.e., $j > nAtoms$), these being ghost atoms. For such atoms no space exists in that processor’s force array and a dummy variable should be used to hold their forces:

```

double dummyForce[DIMENSION];
...
if(j<nAtoms) forceJ = (*forces)[j];
    else forceJ = dummyForce;

```

Incidentally, this is the reason that the `HalfNeighbors()` function of `NeighborLocator` returns neighbors j with $j > i$, rather with $j < i$ —it allows ghost atoms to be included.

In writing transformers, it is even more necessary to use array access to the `ListOfAtoms`; that is, use `SetCartesianPositions()` rather than looping and calling `SetCartesianPosition()`, since the latter entails the `NeighborLocator` updating itself, and thus communication between processors, for each loop iteration.

Alternative choices

We have not implemented any kind of dynamic load-balancing scheme. In solid mechanics atoms do not tend to move a whole lot, nor does density tend to change much so that if the atoms are well distributed at the start of the simulation they will more or less remain so. However if one wanted to implement load balancing, which would amount to redefining processor boundaries dynamically, one could implement a new subclass of `DomainSubdivision` which would implement whatever algorithm was to be used for the load-balancing.

4.3.2 Serialization

Serialization is the process of storing objects, generally containing the data of the simulation, to a file, such that they can be recreated by the application running again at some later time. The term “serialization” comes from the fact that in a

file data is represented as a linear stream of bytes, and it is not necessarily trivial to determine how a complex data structure should be put into such a form. In molecular dynamics simulations we often wish to save the state of the system at regular intervals, perhaps every N time steps. There are two possible reasons for this: (1) We wish to defer certain analyses until after the simulation run and (2) We wish to be able to restart a simulation at the point where it left off, or perhaps from some intermediate point, but changing some parameters. Apart from the atomic state (positions and velocities, as well as the tree structure of the `ListOfAtoms`, type-names, masses etc.) it is useful to be able to save other objects in the system, such as the `Potential`, `Mover`, `NeighborLocator`, etc, so that if restarting at some time in the future there will be no doubt about which parameters are associated with which runs. Since Digital Material is intended to be run from a scripting language such as Python, the basic parameters of a given simulation will typically specified in the Python script. Ideally the values of all Python variables would be saved with the C++ objects in an automated way such that there is never any confusion associating C++ objects to appropriate parameters. Presently this is not the case, and applications must arrange their own manner of coordinating the saving of basic simulation parameters and C++ objects. A typical method is at each time step to store the main C++ objects (`ListOfAtoms`, `Potential`, `Mover`, etc.) in one file, and save the Python variables using the pickle module, in a different file but with a clearly related name (e.g. the C++ filename with `.pickle` appended).

Serializing C++ objects in Digital Material

We will now focus on how we serialize and de-serialize (restore from a file) C++ objects in Digital Material. In the spirit of OOP, we have separated the interface of the Serialization from the implementation. The interface is defined by three abstract base classes (two of which are closely related), `Serializable`, `DMWriter` and `DMReader`. The methods of these classes are shown in table 4.3.2 (most of the arguments are declared `const`, but this has been dropped in the table to save space, as has `void` for return values).

Thus a `DMWriter` provides methods for saving the primitive data types: integer, double, bool, etc. as well as arrays of these. In addition there is a `Put` function which takes a pointer to a `Serializable` object and saves the state of that object (the work will actually be work by the various `Put` functions). The `DMReader` provides methods for reading the same primitive types from a given file, as well as two functions for restoring `Serializable` objects: `Get` and `Fill`. The difference between them is that `Get` takes a name (a string), creates the appropriate class object and restores its state from the file; `Fill` takes a pointer to an already created, but “empty” `Serializable` object of the appropriate type, and “fills in” its member data. Note that a string-name is associated with every piece of data that is saved/put or loaded/gotten, including the entire object.

For a class to be `Serializable`, it must derive from the base class `Serializable`, and thus implement the three methods in table 4.3.2. Two have obvious purposes: the `Save` and `Load` methods, upon being passed a pointer to a `DMWriter` or `DMReader` respectively, call the latter’s methods in order to save or load the individual primitive objects making up the object’s state. The `GetType()` function always returns a string equal to the name of the class. The purpose of this is to allow

Table 4.1: Methods for Serializable, DMWriter and DMReader.

Serializable	DMWriter	DMReader
Save(DMWriter *)	Open()	Open()
Load(DMReader *)	Close()	Close()
string &GetType()	Put(Serializable *, string&)	string GetType(string &name)
	PutBool(bool, string&)	bool Fill(Serializable *, string &name)
	PutInt(int, string &name)	int GetInt(string &name)
	PutDouble(double, string &name)	double GetDouble(string &name)
	PutString(string &name, string &name)	string GetString(string &name)
	PutIntArray(int , vector<int> &, string &name)	int GetIntArray(vector<int> &, string &name)

objects of an appropriate type to be created given a string containing the class name. This is straightforward in Python (using the `exec` command for example) but requires some mechanism in C++, which for which we have used the Design Patterns “Abstract Factory” and “Factory Method”, with Abstract Factory itself using the “Singleton” pattern. The description of the process is a little complex to describe, but it uses surprisingly little code. When high level code wishes to serialize an object, it creates a `DMWriter`, and calls its `Put()` method, passing the `Serializable` object (as a pointer), as well as a name, by which the object can be retrieved (the name is necessary since one could save more than one object of the same type to a given file, so a means of distinguishing them is necessary. For example the name for a `ListOfAtoms` could be ‘`LOA_00012`’ for the 13th (counting from zero) `ListOfAtoms` to be saved to this file. The `DMWriter` gets the type name for the `Serializable` (e.g. ‘`DynamicListOfAtoms`’) and saves this along with the name that was passed. Next it calls the `Save()` method of the `Serializable`, passing itself. The `Save` method uses methods of the `DMWriter` to save all of its data.

The magic comes when we come to recreate an object from a file. Having created an appropriate `DMReader`, passing the appropriate file name, we call `Get()`, passing the same that was used to save the object originally (e.g. ‘`LOA_00012`’). The `DMReader` looks at the file and finds a string containing the class name. The thing is now to create an object of that type. For this two classes are used. `SerialFactory` is a singleton class (meaning only one instance of it ever exists at a time) which can take a string containing a class name and return a pointer to a newly created object of that type. It can do this because for each `Serializable` class there exists a corresponding class `SerialBuilder`; the correspondence being through a template argument. A global instance of the `SerialBuilder` is created for each

Serializable type (by a line at the top of its implementation file). The `SerialBuilder` has one chief method, called `Build()`. This method dynamically creates (using the `new` operator) a new object of the same type as its template argument and returns the pointer. There is one more piece of the mechanism: When each `SerialBuilder` is created, it registers itself with the (unique, *a la* Singleton) instance of `SerialFactory`, providing both the string containing the appropriate class name, and a pointer to itself. Thus the `SerialFactory` has a map from strings (containing class names) to pointers to objects which can create the corresponding objects.

To make this work, there is one more requirement of a `Serializable` (in addition to providing the three member functions listed in table 4.3.2: It must have a public constructor that takes no arguments. Otherwise the corresponding `SerialBuilder` would not be able to construct one. In general not every single member variable is saved/loaded: only those data which cannot be recreated later. For example, when a `NeighborLocator` is serialized and then de-serialized (restored from file) the actual neighbor-lists are not saved, because these can be recreated when the `NeighborLocator` is reconnected to a `ListOfAtoms`. This brings up a point which is a general issue in Serialization: what do with references (pointers) to other objects. In Digital Material, the main object references are that from a `ListOfAtoms` to its `BoundaryConditions` and `NeighborLocator` (and vice versa for the latter) as well as from an `AtomsMover` to the `Potential`. One cannot save the actual pointer value because this will certainly not be the same when the pointed-to object is recreated. One could imagine mechanisms whereby a map “objectNamePtrs”, mapping strings (containing object names) to pointers (to the respective objects) would exist within the simulation, as well as a map “nameToNameRefs” of strings to strings representing object references. `nameToNameRefs`

would be serialized, but not `objectNamePtrs`, since the pointer values would be meaningless later. `objectNamePtrs` would be recreated as each object was de-serialized, and once `nameToNameRefs` was de-serialized from it the pointers could be reset. However it is not clear that this could be done in a truly general way and we have decided for now to let the “high-level” application reconnect the objects by calling `SetNeighborLocator()`, `SetBoundaryConditions()`, etc., after de-serializing the respective objects.

In Digital Material, we have implemented two concrete classes (pairs of classes) as subclasses of `DMWriter/DMReader`. One saves everything in an ASCII format. The other uses the freely available binary file format known as NetCDF [49].

4.3.3 Graphics/Visualization

Visualization is an important tool for the analysis of MD simulation results. Particularly in large scale simulations one does not know *a priori* what processes are going to take place (e.g. dislocation motion). Ideally the software would be able to automatically identify defects and extract their properties and trajectories from the atomistic data. However we are not able to do this yet. Human visual analysis is still crucial. This requires visualizing the atoms in such a way that a person can identify features such as dislocations, cracks, and other defects. This generally comes down to choosing an appropriate way to color the atoms (where “color” can include transparent, i.e., leaving them out). This in turn involves constructing functions whose values near defects are clearly distinct from those in defect-free regions. The simplest such functions are the atomic energy and the (mis-)coordination number (number of neighbors within a cutoff distance). These two differ in their conceptual basis, one being purely geometrical in nature,

while the other depends upon the interatomic potential. Other potential-based functions include various components of the local (atomic) stress tensor, or in a dynamical simulation the force magnitude (which is zero for all atoms in relaxed state of course). A recently introduced geometrical measure of mis-coordination is the “centro-symmetric deviation” [45] which can be applied to materials in whose ground state lattice the neighbors of each atom occur in oppositely positioned pairs. The quantity computed is the sum over such pairs of neighbors of the square of the sum of the deviations of their positions from their ideal lattice positions with respect to the given atom. This quantity is zero for a homogeneous deformation.

In keeping with our general intent not to re-invent the wheel—namely, not to re-implement features which have already been well implemented by others, but rather to make use of existing freely available packages for standard tasks (linear algebra, binary file storage) we have not developed our own visualization package, but sought to make it easy to use existing packages. We have not made an extensive investigation of all the different packages (OpenDX, VMD, etc.) that are in use for MD visualization, but we have learned some things about how to incorporate visualization into the simulation of materials.

There are two “modes” of visualization that may be employed in MD simulations: *real-time visualization* and *post-processing visualization*. Real-time visualization is only feasible when the system is small enough that the state of the system changes noticeably over the period during which the simulator cares to observe it. However it has some important uses: (1) demonstration applications of the software (2) educational applications of the software and (3) debugging of scientific applications, where if it is known that “something bad happens” within a short time of starting a simulation, visualization of the atoms can often give

an immediate understanding of the problem (e.g., the time step was too big and atoms ended up overlapping too much and thus the system exploded).

We have used the following three visualization tools, the first two of which have been implemented as Observers of the ListOfAtoms.

PlotAtoms A simple two-dimensional program written as part of the LASSPTools package[57]. Its strength is its smooth presentation of real-time updates of the atomic state.

RasMol [55] A powerful program for visualizing molecules—it has features for highlighting parts of proteins etc.—which is useful too for materials MD simulations. Its strength is its 3D rendering, and its facility for the user to interactively translate, zoom and rotate the “molecule”. It is not particularly suitable for real-time visualization, and when used for such, presents a flickering image.

Chime [55] An enhancement to Rasmol, designed to run as a plug-in for the Netscape web browser. Our experience with it is fairly limited. An advantage over bare Rasmol is that it supports animations made from separate configuration files concatenated into a single multiple-frame file. However it is not clear that it can handle real time updates as well as PlotAtoms.

For real time visualization, say in the case of a demonstration application with a few hundred atoms, we can attach the appropriate observer (PlotAtomsObserver, RasMolObserver) in the Python script, and the graphics display will update every major time step (assuming Notify() is being called on the ListOfAtoms every major time step). When real time visualization is not practical, we can make use

of an Observer which saves the state of the ListOfAtoms (and the NeighborLocator, Potential, ...), every major time step, or at whatever interval is considered appropriate (see Serialization, subsection 4.3.2 above). Using a separate Python script we can read these snapshots from disk, attach the appropriate Observer and display the snapshots in sequence, creating an effectively real-time animation of the trajectory. From the same script we can create configuration files in formats appropriate to other visualization tools if desired. We also also perform elementary transformations of the positions such as rotations and translations, or take subsets of the configuration (this is necessary for PlotAtoms, but not for RasMol).

ColorMethod

As discussed above it is crucial to be able to “color” the atoms in a useful way. The process of coloring is abstracted as the ColorMethod base class, whose subclasses implement the coloring methods described above: EnergyColorMethod, CoordinationColorMethod, etc. The chief requirement for subclasses is to overload the function call operator to take a ListOfAtoms and an integer (an atom number) and return a double, representing a color value. Subclasses are also allowed to have an Update() function, taking a list of atoms, which is intended to be used for calculating the colors of all atoms at once rather than one at a time as requested by the graphics observer class. This can be important for efficiency in real time visualization. At this time we actually have PlotAtomsObserver implemented both in C++ (with Python wrappers provided by SWIG) and in Python, which is somewhat redundant. The pure Python implementation allows pure Python ColorMethods to be defined which is convenient, except that it may often be the case that efficiency requires a C++ implementation. An alternative method of coloring is according

to which branch or leaf a given atom is on, which is useful when it is desired to indicate boundary atoms, for example. This is the default coloring method of `PlotAtomsObserver` when no `ColorMethod` is specified (the user can choose which leaves have which colors, including the value -1 for “do not display”).

4.4 Summary

We have given a fairly detailed exposition of our view to write a modern molecular dynamics code, paying strict attention to modern software design principles. Some examples of its use will be presented in subsequent chapters. We hope that with the descriptions provided in this paper, a person could implement a code more or less similar to ours, although this would probably not be the case for parallelization and serialization, which involve more detail than has been described here.

We would like to point out a further benefit of using Python. One of the magic things about Python is that a function call only needs the function name to be correct in order to work—there is no type checking. This means that if a another MD code was written with quite different low level details, but with the same high level interface as the Digital Material, existing Python scripts could be used with the other code. Python scripts which implement applications at a high level could be shared between researchers using different core MD code.

Chapter 5

Overlapping Finite

Elements/Molecular

Dynamics (OFE/MD)

This chapter presents the development of a simulation tool which embeds molecular dynamics simulations within finite element continuum models. Since a key feature of the formulation is the atomistic and continuum regions *overlap*, it is being termed Overlapping Finite Elements/Molecular Dynamics or OFE/MD.

5.1 Motivation and purpose

The motivation for this research comes from the Adaptive Software Project (ASP), an engineering and computer science collaboration, which aims to develop software for solving large complex modeling problems. The problems envisaged involve solid mechanics (finite element elasticity and heat flow), fluid mechanics, complex geometries, multi-scale phenomena, and require the software to *adapt* in various

ways. Levels of adaptivity include (1) the *system level*, which involves adapting to changes in hardware resources (such as a change in the number of processors available, or a failure of one or more processors, necessitating some kind of recovery scheme) (2) the *algorithmic level*, which involves the usual kinds of changes of grid spacing and time step and (3) the *application level*, which involves changes in the mathematical model, i.e., the governing equations, used to describe the physical system. It is the latter kind of adaptivity which inspires the current work. Specifically, the idea is that a high level control mechanism in the software would decide that at some moment in the time evolution of the model, and at some particular region within the model, the continuum model—in this case a finite element (FE) elasticity model—has become singular and needs to be supplemented with new physics. This new physics is to emerge from a smaller scale model, involving for example treatment of individual grains in a polycrystalline material [40], or in this case, a molecular dynamics (MD) simulation.

Thus the aim of this work is to develop some capability to “spawn” an MD simulation from a continuum simulation. The purpose is as much an exercise in adaptive software as it is to develop a useful simulation tool. That, being said, it is of course true that coupling continuum models to atomistic models is a very fashionable pursuit (the theme of this thesis being a case in point). In the context of developing innovative software, a choice was made to work with a coupling mechanism which is fairly simple compared to say, the quasi-continuum (QC) method. Nevertheless, this coupling mechanism seems to be new, and moreover, seems to be free of some of the difficulties plaguing the QC method.

Once an MD simulation has been spawned it can have one of two purposes, which we shall refer to as *contexts*: Context (1) is to perform a diagnostic com-

putation, in order to answer a question such as “will a seed crack in this location grow, given the stress and temperature conditions at this point and time?” or “will the material melt, given the stress and temperature conditions at this point and time?”. In this case an answer to the question will signify the end of the MD simulation and further simulation of the model as a whole will involve only continuum mechanics. Context (2) is for the MD simulation to become a part of the overall dynamical model, enhancing it in the sense of correcting the continuum equations by including effects of nonlinearity as well as effects of inherently atomistic processes like melting, fracture or plasticity. These effects would be the weakening (from fracture or melting) or hardening (as in work hardening) of the material, but they not necessarily be identified as such (unlike context (1), whose point is to identify specific processes).

Both from a physical point of view and from a numerical point of view, context (2) is not practical when the region simulated with MD—called the *atomistic region*—is a very small fraction of the total model, which would be the case in typical (for example aerospace) engineering applications. This is because the contribution of a typical atomistic region of the model to the overall forces on the continuum degrees of freedom is almost negligible. Thus the difference between the linear and nonlinear contributions, even where significant atomistic processes take place, is indeed negligible—in such applications even individual elements are much larger than atomic sizes. In these applications only context (1) makes sense. However in the age of nanotechnology one can imagine that there will be engineering applications with significantly smaller length scales, which will not be so much larger than the atomistic length scale. Furthermore, given that this is primarily a software project, the techniques developed and insights gained from the experience

of coupling these two different kinds of models in at least a semi-automated way is expected to be of general benefit.

In the meantime it is envisaged that context (2) will be more useful in physics and materials science applications, where it is desired to study some particular atomistic process(es) with realistic boundary conditions produced by embedding the simulation in a continuum model. It is worth noting that the simulation techniques involved in these two contexts are not as different as one might think. In context (1) we effectively freeze the main simulation while carrying out the MD simulation. This would seem to imply that the atomistic simulation will have fixed-displacement boundary conditions (as we will see when we formulate the model), since the the finite element nodal displacements will be fixed. This would also imply that forces on the continuum degrees of freedom due to the atomistic configuration are not needed. However fixed displacement boundary conditions are not physical; in the real material the boundary of a small region will experience fixed stress, just as the correct thermodynamic boundary conditions are fixed temperature rather than fixed energy¹. Therefore we actually need to create a separate continuum model consisting a small subset of the main mesh, which includes the atomistic region. It could be defined simply as the set of elements which contain any atoms. Applied to the outer surfaces of this miniature continuum model are traction boundary conditions inferred from the stress state of the main continuum model. The coupled FE/MD simulation is then carried out on this reduced model and the atomistic region contained within. Therefore in both contexts (1) and (2) we need an FE/MD model which is truly coupled.

¹By “fixed” here I mean constant on the time and length scale of the MD simulation.

5.2 Principles of the coupled OFE/MD model

In this section we present the formulation of our model. A model of a physical system consists of two parts: the specification of the degrees of freedom and the rules governing their evolution. Since this is a mechanical model these rules will be dynamical equations of motion, and since we wish the model to have a firm physical basis, these will be derived from an energy functional, a Hamiltonian. In the next three subsections we present the kinematics (definition of degrees of freedom), energetics (energy functional) and derivation of forces.

5.2.1 Kinematics

In this subsection we will describe the atomistic degrees of freedom and their connection to the finite element degrees of freedom (nodal displacements). Terminology will be introduced and defined.

The **sample** is a region of space filled with solid material. The **continuum model** is a finite element model whose mesh, defined by set of N_n **nodes** and N_e **elements** connecting the nodes, covers the space occupied by the sample. The displacements of the nodes form a vector of length $3N_n$ denoted \mathbf{q} . The **atomistic region** is a subset of this region, containing all of the atoms explicitly represented in the simulation. It consists of a **core region** and a **boundary region**. The atoms in the core region are known as **free atoms**, **unconstrained atoms** or **core atoms**. The atoms in the boundary region are known as **fixed atoms**, **constrained atoms**, **slaved atoms** or **boundary atoms**. The distinction between free atoms and fixed atoms is as follows: The positions of the free atoms, denoted by $\{\vec{r}_i\}$, are new independent degrees of freedom in the coupled model.

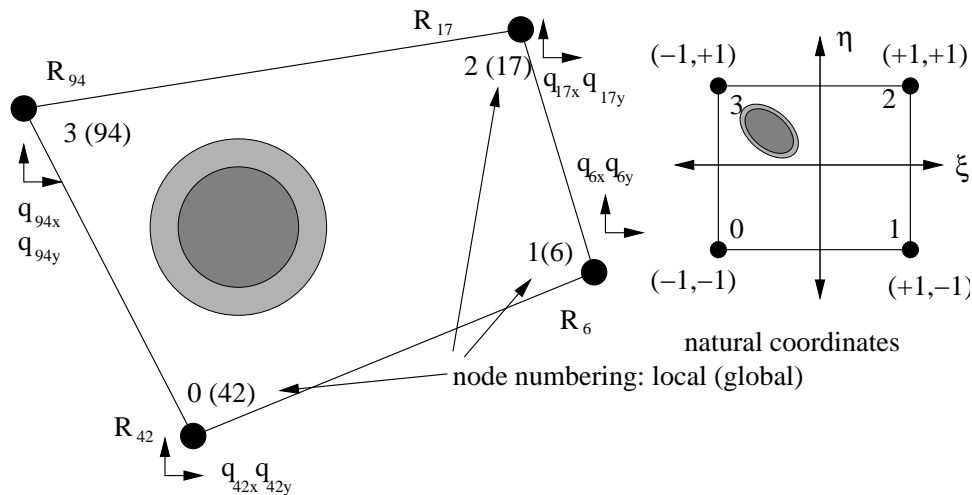


Figure 5.1: Two-dimensional atomistic region embedded entirely within a single element, and “natural coordinates” of the element.

The positions of the fixed atoms, however, are determined entirely by the current values of the nodal displacements via interpolation using the finite element **shape functions**, so these positions are not new degrees of freedom in the model. The position of a fixed atom whose undeformed position is in element e will always be determined by the displacements of the nodes of e . The purpose of the fixed atoms is to provide a boundary for the core atoms, and to determine the forces acting on the nodes due to the atoms. We call the number of core atoms N_c , the number of boundary atoms N_b and their sum, number of atoms in the simulation N_{at} . When we need to refer to the positions of all atoms, and not just the independent ones, we will use a prime: $\{\vec{r}'_i\}$. Fig. 5.1 shows a simple case in a 2D model where the atomistic region is entirely within a single element

Let us note two further points concerning the geometrical relationship between the atomistic region and finite element mesh: (1) The atomistic region is small com-

pared to any element and could generally fit inside an element, except that often it will be located near element boundaries at cracks, notches or sample boundaries, and thus several elements could contain parts of the atomistic region (i.e. atoms).

(2) The atomistic region *overlaps* the finite element mesh. This contrasts with other instances of coupled finite elements and molecular dynamics, in which the mesh does not overlap², and where the mesh is refined down to the atomic length scale near the atomistic region or where indeed the mesh is very much tied to the atomic lattice, as in the quasi-continuum method [70, 69, 51]. Because of the overlap, it is important not to double count the energy within the overlap region.

5.2.2 Energetics

Having defined degrees of freedom (DOFs) we now define an energy functional. If there were no atomistic region (and no DOFs other than the nodal displacements) the energy would be the continuum elastic energy. To begin with we consider only linear elasticity. The elastic energy is a sum over elements:

$$E_{\text{elastic}} = \sum_e \frac{1}{2} \mathbf{q}_e^T \cdot \mathbf{K}_e \cdot \mathbf{q}_e = \frac{1}{2} \mathbf{q}^T \cdot \mathbf{K} \cdot \mathbf{q} \quad (5.1)$$

where subscript e denotes elemental quantities: \mathbf{K}_e is an element stiffness matrix; \mathbf{q}_e is the vector of nodal displacements for the element e . We have also written the energy as a single matrix expression where \mathbf{K} and \mathbf{q} are now the *global* stiffness matrix and displacement vectors, respectively. The stiffness matrix is calculated by doing an integral of the elastic energy density and factoring out the nodal displacements.

²Except possibly at a very thin (one atomic distance thick) boundary region at the boundary of the atomistic region [2, 1, 14]

As we did in chapter 2, in the atomistic region we will replace the elastic energy density with the full atomistic energy density³. To avoid an abrupt transition from one energy density to the other—necessary since one is the integral of a continuous function while the other is a sum over discrete points—we will again use a smooth transition function $T(X)$. Again, the capital X indicates that the transition function is evaluated at *undeformed* positions. Let us temporarily omit explicit reference to \mathbf{qs} and \vec{r}_i : since both the elastic energy and (formally at least) the atomistic energy can be written as a volume integral of an energy density we can write the total energy as follows:

$$E = \int_{\text{sample}} d^3X (T(X) \mathcal{G}(X) + (1 - T(X)) \mathcal{F}(X)) \quad (5.2)$$

where $\mathcal{G}(X)$ is the atomistic energy density at undeformed position X and $\mathcal{F}(X)$ is the linear elastic energy density. $\mathcal{G}(X)$ is really a sum of delta functions—one at each undeformed atomic position—weighted by the ratio of that atom’s current energy to the atomic volume (volume per atom in the perfect crystal); the transition function $T(X)$ is a weighting function equal to unity within the part of the atomistic region occupied by free atoms, zero in the continuum region, and varying smoothly between the two in the border part of the atomistic region⁴. If we group terms which multiply $T(X)$,

$$E = \int d^3X (\mathcal{F} + T(\mathcal{G} - \mathcal{F})), \quad (5.3)$$

³It is important to note that we may speak of the atomistic *energy density* even away from the atomistic *region*, since one may always imagine decorating the continuum region with atoms and calculating the energy at a point corresponding to an atom location as the energy of the atom.

⁴It turns out that the region where $T(X) = 1$ does not correspond exactly with the core region. The reasons will be explained in the next subsection

then we can view the total energy as being the full linear elastic energy plus a correction from the atomistic region, namely the difference between the elastic energy and the atomistic energy there. We now reintroduce the nodal displacements \mathbf{q} and the atom positions $\{\vec{r}_i'\}$ (recall that the prime refers to fixed atoms as well as core atoms—the positions of all atoms needed to completely evaluate the atomistic energy). For the purposes of computation, we write the total energy as a sum of three terms: (1) the full elastic energy, which is as we saw above a matrix expression, where we now call stiffness matrix $\mathbf{K}_{\text{total}}$ to signify that it gives the elastic energy of the whole sample; (2) the atomistic energy, now written in its natural form as a sum of atomic energies weighted by transition function values, $E^{\text{atomistic}} = \sum_i T(\vec{r}_i^0) E_i$ and (3) a term which is the elastic energy weighted by the transition function, also written as a matrix expression, where the matrix is called the *partial stiffness matrix*. It is assembled from elemental partial stiffness matrices (one for each element which contains a part of the atomistic region). We have:

$$E = \frac{1}{2} \mathbf{q}^T \cdot \mathbf{K}_{\text{total}} \cdot \mathbf{q} + \sum_i T(\vec{r}_i^0) E_i - \frac{1}{2} \mathbf{q}^T \cdot \mathbf{K}_{\text{partial}} \cdot \mathbf{q} \quad (5.4)$$

5.2.3 From energy to forces

Given an energy functional the forces on the DOFs are obtained by differentiating with respect to them. Consider forces acting on the core atoms first. These forces come from terms which depend on the free atom positions \vec{r}_i , namely $E^{\text{atomistic}}$. It is desirable that these forces be the same forces that are calculated by standard MD functions is being used so that no new coding is needed. This means that the force on a core atom should not be affected by any atoms whose transition value

is not unity. Since atoms near the edge of the core are affected by the nearest atoms of the boundary region, this means that the transition function should be unity for a depth into the boundary region equal to the cutoff distance of the potential. This is true not just for pair potentials (e.g. Lennard-Jones) but also for three-body potentials (e.g. Stillinger-Weber) and many body potentials (e.g. EAM, MEAM or EDIP). It is true that for non-pair potentials the forces on core atoms will depend on the positions of boundary atoms up to two cutoff distances away, but only through the energies of atoms less than one cutoff distance away, so these contributions are still weighted by $T = 1$. In the following, letters $i, j, k \dots$ denote atom indices, letters $a, b, c \dots$ denote Cartesian coordinates, and Greek letters α, β, γ denote nodes. We have:

$$\vec{F}_i = -\frac{dE}{d\vec{r}_i} = -\sum_k \frac{dE_k}{d\vec{r}_i} \quad (5.5)$$

where the sum over k will include all atoms within one cutoff distance of atom i , *as long as the transition function is unity for all of these atoms!* Next, to derive the forces on the nodal displacements, first let us denote the displacement of the α -th node as a vector \mathbf{q}_α with Cartesian components $q_{\alpha a}$, $a = 1, 2, 3$. All three terms in equation 5.4 depend on the \mathbf{q} s. For the matrix expressions this is simply a quadratic dependence, which gives linear terms on differentiation. To obtain the atomistic contribution we employ the chain rule:

$$\vec{F}_{\alpha a} = -\frac{\partial E^{\text{atomistic}}}{\partial q_{\alpha a}} = -\sum_{i'} T_{i'} \frac{\partial E_{i'}}{\partial q_{\alpha a}} = -\sum_{i', k'} T_{i'} \frac{\partial E_{i'}}{\partial r_{j'b}} \frac{\partial r_{j'b}}{\partial q_{\alpha a}} \quad (5.6)$$

where $T_{i'} = T(\vec{r}_{i'}^0)$ is the transition function evaluated at the undeformed position of the i' th atom, the prime again signifying that in principle we sum over all atoms in the simulation, not just free ones. Consider the last factor in the last

expression, $\frac{\partial r_{j'b}}{\partial q_{\alpha a}}$. This can only be nonzero if the undeformed position of atom j' is in an element containing node α . When this is the case, since the interpolation gives the atomic displacements (and hence current positions) as a linear function of the nodal displacements, the derivative with respect to the latter is just the coefficient—the shape function:

$$\frac{\partial r_{j'b}}{\partial q_{\alpha a}} = N_{\alpha}([\vec{r}_{j'}])\delta_{ab} \quad (5.7)$$

The Kronecker delta appears because the interpolation maps the a th coordinate of the nodal displacements to the a th coordinate of the atomic displacements. The square brackets are a reminder that the the shape function is not an explicit function of the real space position—an intermediate step to convert into the element’s natural coordinates is required. The finite element library we use contains a function which takes a set of point forces within an element, makes the appropriate products with shape functions and returns the contribution to the nodal forces from that element. It remains for us to recognize that the point forces to be passed are not the “bare” atomic forces, but the atomic forces weighted in a certain way by the transition function—not that the force on a particular atom is multiplied by the transition value for that atom, but the contributions coming from different atoms’ energies are weighted by their transition function. Thus we need to have a way of obtaining from the Potential class not the complete set of forces, but the forces due to a single atom’s energy. For this purpose a member function `AtomicForcesEnergy` has been added to the Potential interface⁵.

Apart from that the implementation is simple. Let us call the resulting vector

⁵This need not involve extra code as in many cases the force calculations are already coded as a loop over such atomic energy contributions.

forces on the nodes due to the atomistic energy $\vec{F}^{\text{node-atomic}}$. Finally, we also include in the nodal forces constant loading forces, possibly derived from pressures/stresses acting on element surfaces, denoted \mathbf{F}_{load} . Now that we have forces we can write down equations of motion:

$$m \frac{d^2 \vec{r}_i}{dt^2} = \vec{F}_i \quad (5.8)$$

$$\mathbf{M} \cdot \mathbf{q} = -K_{\text{total}} \cdot \mathbf{q} + \vec{F}^{\text{node-atomic}} + K_{\text{partial}} \cdot \mathbf{q} + \mathbf{F}_{\text{load}} \quad (5.9)$$

Here \mathbf{M} is a mass matrix for nodal displacements which could be calculated by integrating the product of the density and the acceleration field over each element. We have so far only considered quasi-static situations rather than true dynamics so we do not need to calculate this mass matrix—we generally set the left hand side of equation (5.9) to zero.

5.3 Dynamics and algorithms

The nodal DOFs correspond to masses of material, which are large compared to atoms, thus their characteristic time scales are much longer—these are **slow** degrees of freedom. Because of the difference in time scales, it does not make sense to implement the coupled equations directly. If we did, the time step would be limited to the atomic time scale (femtoseconds) and as a consequence the amount a nodal displacement could change at once would be limited to atomic lengths. Thus, for instance, in the case of finding the equilibrium length of a uniaxially loaded beam with an atomistic region in the center, one would have to slowly increment the force in steps small enough that at no point would the nodal displacements ever change by more than an atomic distance. If the continuum model is much

larger than the atomistic region this would be appallingly slow. Clearly, this is not the way to go. Therefore we do not do direct dynamics; rather, given the natural distinction between the two sets of degrees of freedom (fast and slow), we make a distinction in how they are evolved in the simulation. Consider again the beam loaded uniaxially, and at zero temperature. The approach we take is analogous to “integrating out high frequency degrees of freedom” in statistical mechanics. We integrate out (literally) the atomic degrees of freedom for a given configuration of the coarse degrees of freedom \mathbf{q} . Mathematically we now have a model containing only the slow degrees of freedom, and we do separate dynamics on that.

In statistical mechanics, the thermodynamics of a system emerges from evaluating a partition function, which is a sum over all states of all degrees of freedom:

$$\mathcal{Z} = \sum_{\{\mathbf{q}\}, \{\vec{r}_i\}} \exp(-\mathcal{H}(\{\mathbf{q}\}, \{\vec{r}_i\})/T) = \sum_{\{\mathbf{q}\}} \left[\sum_{\{\vec{r}_i\}} \exp(-\mathcal{H}/T) \right] \quad (5.10)$$

The renormalization group concept is based on doing this sum a little piece at a time. Our procedure is analogous to a discrete renormalization transformation where we do the sum over the atomic degrees of freedom only. At zero temperature there is only one term⁶ in the partition function, the state with minimum potential energy. That is, if for any given configuration of \mathbf{q} s we minimize the system with respect to the atomic degrees of freedom then we have a well defined model involving only the \mathbf{q} s. The forces on the \mathbf{q} s in this case of zero temperature are simply the forces determined in subsection 5.2.3 by differentiating the energy, once the minimization with respect to free atoms has taken place.

⁶I’m speaking loosely here; in a system with continuous degrees of freedom the partition function is an integral rather than a sum, so one cannot literally talk about “one term in the sum”.

5.3.1 Zero temperature algorithm

Having integrated out the free atoms we have a nonlinear energy functional of the \mathbf{q} s. Consider a static load applied to the system; we wish to find the equilibrium displacements (thus this is a problem in statics rather than dynamics). The load appears in the energy as terms linear in \mathbf{q} . In terms of forces the loads are put on the right hand side and we have an equation like

$$K_{\text{full}} \cdot \mathbf{q} + \nabla_{\mathbf{q}} E^{\text{atomistic}} - K_{\text{partial}} \cdot \mathbf{q} = \mathbf{F}_{\text{load}} \quad (5.11)$$

which is a non-linear equation to be solved for \mathbf{q} . We use a modified Newton method given by the following algorithm:

1. Construct the full and partial stiffness matrices and the right hand side vector from the load, using standard finite element procedures, along with a new routine for evaluating the integrals for the partial stiffness.
2. Solve linear problem with $\mathbf{K}_{\text{total}}$ to give nodal displacements \mathbf{q}_n .
3. Displace boundary atoms according to \mathbf{q}_n .
4. Relax core atoms (minimize).
5. Compute full (nonlinear) forces.
6. Increment RHS by (load - force).
7. Repeat from step 2 until a an overall convergence criterion is satisfied.

This algorithm is a modification of Newton's method because the same stiffness matrix (which is the gradient at zero displacement) is used at every step, whereas the true Newton method involves computing the exact derivative (of the forces,

in this context)each time. The modified method is commonly used in nonlinear finite element modeling because although it takes more iterations to converge,this is more than compensated for by not having to compute the exact derivative of the forces each iteration, which would involve second derivatives of (in our case)the interatomic potential.

5.3.2 Finite temperature algorithm

At finite temperature, one has to sum the part of the partition function corresponding to the atomic degrees of freedom; this leads to an effective Hamiltonian for the \mathbf{q} s at that temperature. If we separate the linear and nonlinear parts of the “microscopic Hamiltonian” as $\mathcal{H}(\{\mathbf{q}\}, \{\vec{r}_i\}) = \mathcal{H}_{\text{lin}}(\{\mathbf{q}\}) + \mathcal{H}_{\text{nonlin}}(\{\mathbf{q}\}, \{\vec{r}_i\})$ we can write the partition function as follows

$$\mathcal{Z} = \sum_{\{\mathbf{q}\}} \exp(-\mathcal{H}_{\text{lin}}/T) \sum_{\{\vec{r}_i\}} \exp(-\mathcal{H}_{\text{nonlin}}/T) = \sum_{\{\mathbf{q}\}} \exp(-\mathcal{H}_{\text{eff}}(\{\mathbf{q}\})/T) \quad (5.12)$$

where

$$\mathcal{H}_{\text{eff}}(\{\mathbf{q}\}) = \mathcal{H}_{\text{lin}}(\{\mathbf{q}\}) + F_{\vec{r}}(\{\mathbf{q}\}) \quad (5.13)$$

$$\exp(-F_{\vec{r}}(\{\mathbf{q}\})/T) = \sum_{\{\vec{r}_i\}} \exp(-\mathcal{H}_{\text{nonlin}}/T) \quad (5.14)$$

Taking derivatives of $\mathcal{H}_{\text{eff}}(\{\mathbf{q}\})$ with respect to \mathbf{q} gives the effective forces on the nodal degrees of freedom as a standard thermodynamic average of the “bare forces” with respect to the atomic degrees of freedom. In practice this can be implemented as a time average. Thus, the zero-temperature algorithm for finding

the equilibrium state under a static load described above can be generalized to finite temperature by replacing steps 4 and 5 with the following

- i. Initialize average nodal-forces to zero.
- ii. Run thermal MD simulation on the free atoms for n_{minor} time steps.
- iii. Compute full (nonlinear) forces and add to average.
- iv. Repeat n_{major} times.
- v. Divide by n_{major} to get the force average.

5.3.3 Applying the displacement field to core atoms

Separated out the fast dynamics as described above still does not overcome the obstacle of having to make very small increments in nodal displacements. Suppose we move only the boundary atoms when updating the nodal displacements. If they move more than a fraction of an atomic distance, they will be moved on top of core atoms, or even far away from them, leaving them behind; so we must move the core atoms as well. In the first iteration of the algorithm as described in section 5.3.1, we may displace the core atoms along with the boundary atoms, using interpolation from the nodal displacements. We do not lose any information then because the core atoms have not been allowed to evolve yet, but we cannot do so again without losing the information carried by the core atoms. Now, interpolation from the nodal displacements corresponds to a slowly varying background displacement field as far as the atoms are concerned. We would like to be able to add this to the core atom positions while preserving the detailed local arrangements corresponding to whatever atomistic processes and relaxations have

happened so far. This is achieved by referring, not to the original undeformed positions, but to the positions at the previous major iteration. The appropriate displacement field is just the difference between the current nodal displacements and the previous nodal displacements. Using this to interpolate allows us to move the core atoms along with the boundary atoms. It is a kind of shortcut of the full dynamics, but shortcuts are what a coupled FE/MD simulation is all about.

5.4 Model geometries

5.4.1 One-Brick; sphere of atoms

Here the finite element model consists of a single brick element with nodal positions chosen to form a cube in real space a cube (Fig. 5.2). The atomistic region is a sphere whose center coincides with the cube's. One face is held fixed, while the opposite face is loaded in tension. The advantages of this model are: (1) the mesh is simple so it is easy to see what is going on; it is also easy to set up homogeneous deformations; (2) by constraining nodes to zero displacement in the y and z directions we have an effectively one-dimensional problem; (3) the atomistic region does not intersect any boundaries of the body, thus we do not need to worry about periodic boundary conditions or the possible effects a free surface will have. Rather we can be sure that the atomistic effects will be limited to those associated with homogeneous deformations in the bulk. We have used both a linear element (8-noded) and a quadratic element (20-noded). The reason for using the linear element (these are generally considered to be poor elements) was that in the linear case any set of nodal displacements gives a homogeneous deformation; otherwise only certain combinations of nodal displacements do. In trying to obtain approx-

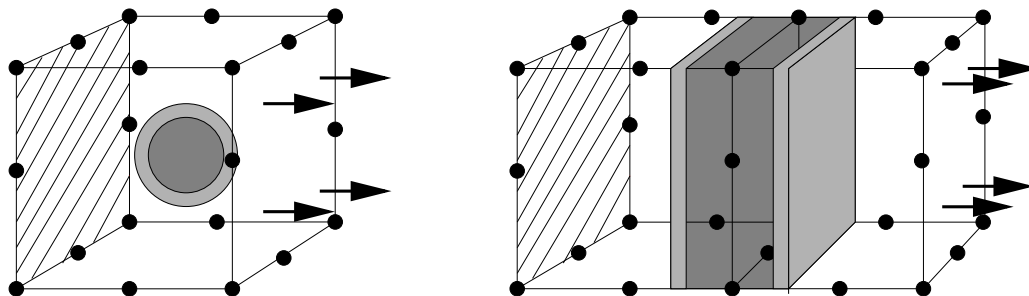


Figure 5.2: Meshes for “OneBrick” and “TwoBrick” models.

imate agreement between the atomistic energy and elastic energy, as discussed in subsection 5.5.1, it is important to have homogeneous deformations.

Figure 5.4.1 shows a stress strain curve for the One-Brick model in the case that only displacements in the direction of loading are allowed. The sphere of atoms must fit entirely within the cube. Because of the region of zero transition value, this condition limits the volume fraction of atoms to 20%⁷ This explains why the OFE/MD curve is closer to the linear elasticity curve than to the atomistic curve: it is a weighted average (80:20) of the two. In a sense the OFE/MD is an *interpolation* between linear elasticity and a full atomistic calculation. The width of the cube in this case was 13.5, the radius of the non-zero transition value region was 5.0, and the potential was Holian’s Lennard-Jones potential (cutoff 1.7; the nearest neighbor distance (FCC) is 1.107). There were no free atoms (for a homogeneous deformation, there is essentially no difference compared to having free atoms).

⁷This ratio is that of the sum of transition values over all atoms times the atomic volume divided by the cube’s volume.

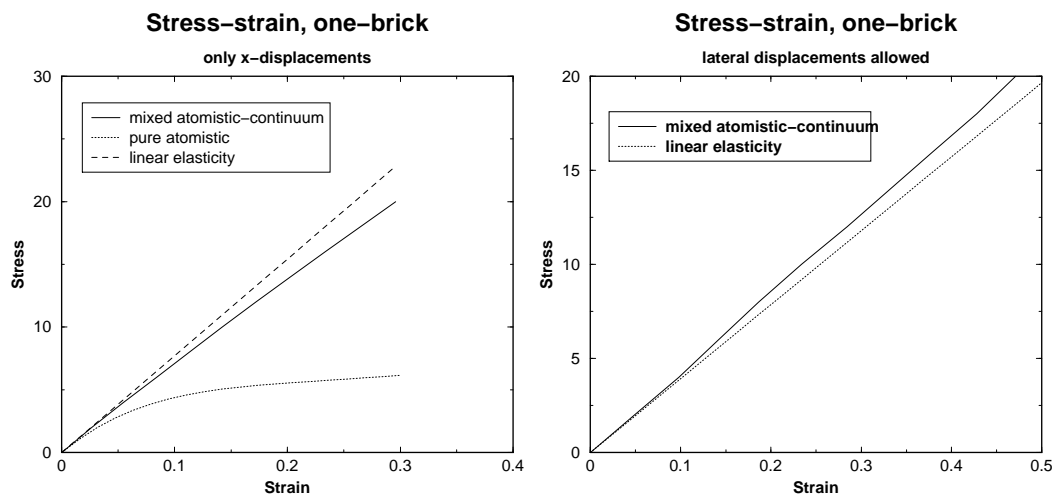


Figure 5.3: Stress-strain curves for One-Brick model, first with x -displacements only, then with lateral relaxation allowed.

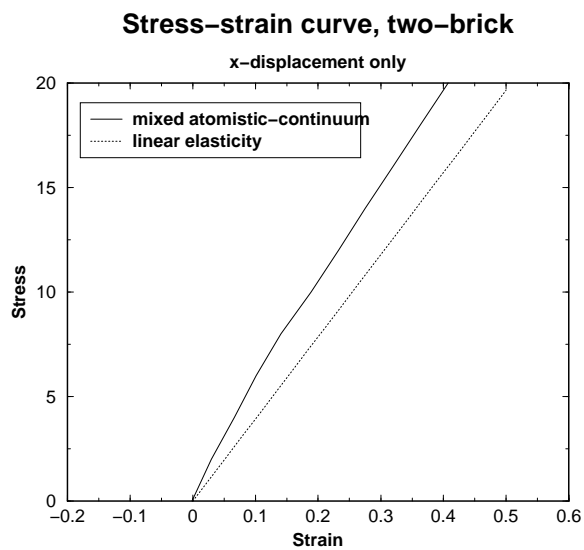


Figure 5.4: Stress-strain curves for Two-Brick model, first with x -displacements only, then with lateral relaxation allowed.

5.4.2 Two-Brick; layer of atoms

This model is a rectangular block whose long faces are free surfaces. It is fixed at one end and loaded in tension or compression at the other (see Fig. 5.2). The FE mesh consists of two cubic 20-noded hex elements with a total of 32 nodes (there being 8 nodes in common). Like the One-Brick model, this can be used as an effective one-dimensional model. The main difference is that the atoms have a free surface and therefore there might be additional effects due to surface relaxations or stresses. At first, periodic boundary conditions were imposed in the directions perpendicular to the long axis but it was realized that these are inconsistent with the simple finite element models used so far. It should be possible, however, to make periodic boundary conditions work—see section 5.5.3 for further discussion.

5.4.3 Cracked silicon plate

This is our first example containing a crack. It is a thin silicon plate, only one layer of elements thick. The mesh was generated using the Franc2D program for fracture analysis in 2D[15] as a 2D mesh containing a crack surrounded by quarter point 6-noded triangular elements. The rest of the mesh consists of 8-noded quadrilaterals and standard 6-noded triangles; there are in total 194 elements and 1181 nodes. Fig. 5.5 shows the 2D mesh in the deformed state due to an applied tensile load on the top and bottom edges. A 3D mesh can be generated in a straightforward manner by mapping triangles to (15-noded) wedge elements and quadrilaterals to (20-noded) hexahedral “brick” elements. The thickness cannot be too small if we wish to use the faster iterative solver to solve the matrix because a large aspect ratio means a more singular matrix (eigenvalues associated with out of plane deformations will be much smaller than those associated with in-plane

deformations). The atomistic region is situated at the tips of the wedge elements near the crack tip. It is a cylinder of radius about 40 Å with thickness equal to that of the mesh. The xy plane is the plane of the plate, with the z -axis coming perpendicularly out of it, and the lattice orientation is such that the $\langle \bar{1}\bar{1}\bar{1} \rangle$ surface is the xz plane, with the $\langle 2\bar{1}\bar{1} \rangle$ direction coinciding with the x -axis, and being the direction of crack growth. The common thickness is chosen to be an integer number of lattice repeat distances (the repeat distance being $a/\sqrt{2}$ where a is the lattice constant for silicon in the z -direction, with this lattice orientation). Three lattice repeat distances (about 23 Å) were sufficient to obtain reasonable convergence with the iterative solver.

Several atomistic effects show up in this model. Since the upper and lower planes are free surfaces their surface energy contributes to the OFE/MD energy, raising it significantly above the purely elastic energy. Similarly, once the model has been loaded and the first linear solve completed, the crack is opened and more surface energy is created. However if the loading is not sufficient for the surfaces to fully clear each other, there will be significant closure forces coming from the atoms. Finally, something which is, more of a technical problem than an effect, is an issue emerging from the use of Silicon with its diamond lattice. When a diamond lattice is strained homogeneously the atoms do not all respond according to the Cauchy-Born rule. The underlying Bravais lattice must respond according to the applied strain, but since there are two atoms per unit cell, they have some freedom of motion in addition to the applied strain. For a diamond lattice symmetry reduces this freedom to a single parameter (see appendix 5.E). The problem is that placing the boundary atoms according linear elasticity becomes non-trivial, since to do this correctly the internal relaxation should be included as well. See the discussion in

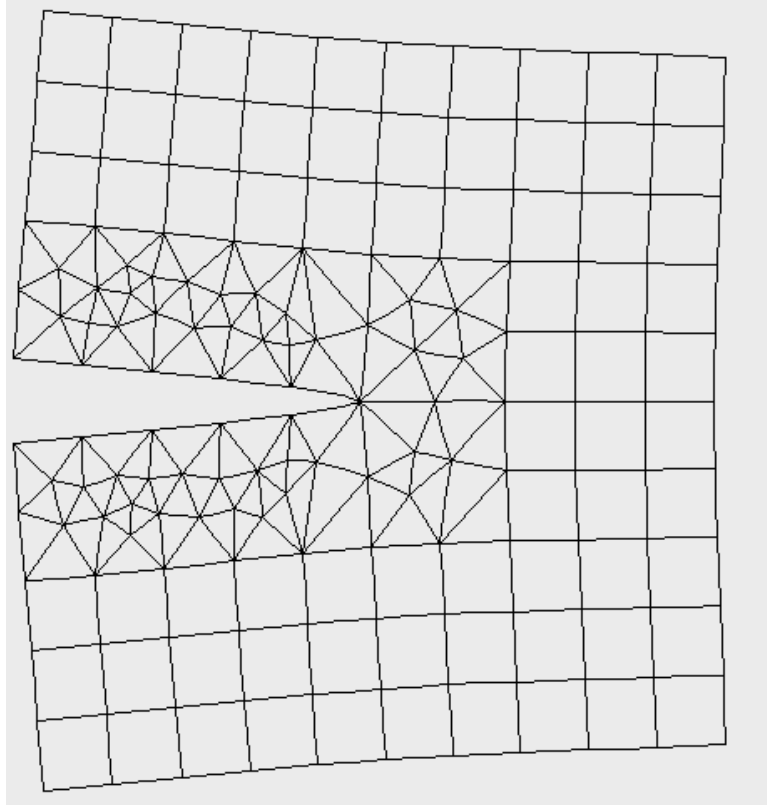


Figure 5.5: 2D Mesh for Silicon thin plate, in deformed configuration.

section 5.5.3.

5.4.4 Cracked Cube

As a test of the capability to interface with a non-trivial finite element model, we have applied OFE/MD to a model cube developed by the Cornell Fracture Group. In fact this is the principal model geometry for the purposes of the Adaptive Software Project, because the atomistic simulation has to be specified to fit the previously existing finite element model. It consists of 7243 10-noded tetrahedral elements, defined by 1531 vertices. The total number of nodes (vertices and edge-

nodes) is 10855. The mesh is shown in Fig. 5.6. The crack plane is parallel to the xz plane, and the crack front parallel to the x -axis. No physical units are associated with the mesh; the coordinates of the corners are just $(\pm 1, \pm 1, \pm 1)$, so that the volume is 8. In these coordinates the crack plane has $y = 0$ and extends from the front of the crack, $z = 1$ to one-quarter of the way in, $z = 0.5$. The mesh is obtained by accessing a data base. Available from the data base, in addition to the list of nodal positions and element connectivities, is information about which edges are part of the crack front and other qualitative details like that. The OFE/MD software chooses an edge near the middle of the crack and places the atomistic region halfway along this edge. We interpret the coordinates of the cube as thousands of atomic units, and so scale all lengths by a factor of 1000. This allows the placing of a sphere of a few thousand atoms on the chosen crack edge without it intersecting any tets than the ones which contain that edge.

5.5 Technical details

5.5.1 Partial Stiffness Matrices

In this section we show how the partial stiffness integrals are calculated. At present this is done case by case—it is desirable to have an automated method. The energy integral for an element e is

$$E = \int_e d^3 X T(X) \mathcal{F}(X) = \int_e d^3 X T(\frac{1}{2} \sigma_{ij} \epsilon_{ij}) = \int_e d^3 X T(\frac{1}{2} c_{ijkl} \epsilon_{kl} \epsilon_{ij}) \quad (5.15)$$

In Voigt two-index notation where $\epsilon_{11} \rightarrow \epsilon_1, \epsilon_{22} \rightarrow \epsilon_2, \epsilon_{33} \rightarrow \epsilon_3, \epsilon_{23} \rightarrow \frac{1}{2} \epsilon_4, \epsilon_{13} \rightarrow \frac{1}{2} \epsilon_5$ and $\epsilon_{12} \rightarrow \frac{1}{2} \epsilon_6$, the integral is written

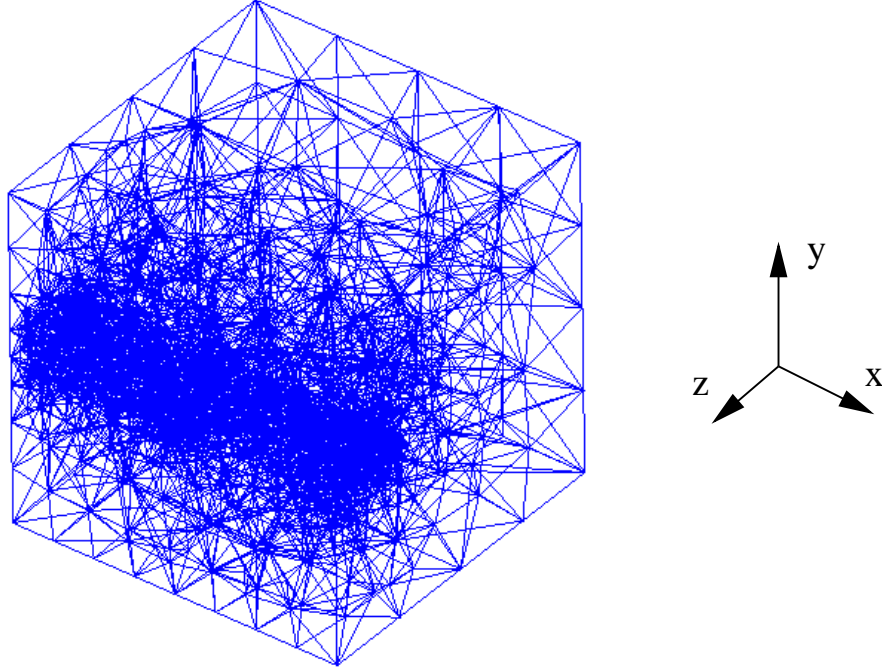


Figure 5.6: Mesh for cracked-cube model.

$$E = \int_e d^3 X T(X) \frac{1}{2} C_{IJ} \epsilon_I \epsilon_J = \int_e d^3 X T(X) \frac{1}{2} \epsilon^T \cdot C \cdot \epsilon \quad (5.16)$$

Now in the finite element formulation, the displacement field is prescribed throughout the element as a function of the internal “natural” coordinates of the element ξ_1, ξ_2, ξ_3 , which is an interpolation of nodal displacements \vec{q}_n ,

$$u_i(\xi_1, \xi_2, \xi_3) = \sum_{n=1}^{N_{sh}} N_n(\xi_1, \xi_2, \xi_3) q_{ni} \quad (5.17)$$

where n is the node index, and N_{sh} is the number of nodes which equals the number of shape functions. The same shape functions⁸ are used to interpolate the real space position of a point given its natural coordinates:

⁸We consider only *isoparametric* shape functions—whose definition is that the same shape functions are used to interpolate both positions and displacements.

$$x_i(\xi_1, \xi_2, \xi_3) = \sum_n N_n(\xi_1, \xi_2, \xi_3)x_{ni} \quad (5.18)$$

where x_{ni} is the i th coordinate of the n th node. The Jacobian matrix is the derivative of real space position with respect to natural coordinates:

$$J_{ij} = \frac{\partial x_j}{\partial \xi_i} = \sum_n A_{in}(\xi_k)x_{nj} \quad (5.19)$$

where $A_{in} = \frac{\partial N_n}{\partial \xi_i}$. The inverse of J is the derivative of the inverse function: $J_{jk}^{-1} = \frac{\partial \xi_k}{\partial x_j}$. When we differentiate u_i to get the strains, the \mathbf{q} s stay outside;

$$\frac{\partial u_i}{\partial x_j}(\xi_k) = \sum_n \frac{\partial N_n}{\partial \xi_k} \frac{\partial \xi_k}{\partial x_j} q_{ni} = \sum_n A_{kn} J_{jk}^{-1} q_{ni} \Rightarrow \partial \mathbf{u}_{ji} = \mathbf{J}^{-1} \cdot \mathbf{A} \cdot \mathbf{q} = \mathbf{D} \cdot \mathbf{q} \quad (5.20)$$

Finally, to get the strain vector ϵ_J , we take certain linear combinations of $\frac{\partial u_i}{\partial x_j}$. This can be written as a matrix B multiplying a column vector \mathbf{q} —the nodal displacements written out in single-column format with elements $q_{11}, q_{12}, q_{13}, q_{21}$ etc. B is a $6 \times (3N_{sh})$ matrix which has the following structure:

$$B = \begin{pmatrix} D_{11} & 0 & 0 & D_{12} & 0 & 0 & \dots & D_{1n} & 0 & 0 \\ 0 & D_{22} & 0 & 0 & D_{22} & 0 & \dots & 0 & D_{2n} & 0 \\ \dots & & & & & & & & & \\ 0 & D_{31} & D_{21} & 0 & D_{32} & D_{22} & \dots & 0 & D_{3n} & D_{2n} \\ \dots & & & & & & & & & \\ \dots & & & & & & & & & \end{pmatrix} \quad (5.21)$$

We can now substitute for the strain vector in eqn. 5.16, giving

$$E = \frac{1}{2} \mathbf{q}^T \cdot \left(\int_e d^3 X T \mathbf{B}^T \cdot C \cdot \mathbf{B} \right) \cdot \mathbf{q} = \frac{1}{2} \mathbf{q}^T \cdot \mathbf{K}_{\text{partial}} \cdot \mathbf{q} \quad (5.22)$$

which defines the partial stiffness matrix in terms of integration over the element e . Without the factor of $T(X)$ this is the usual elemental stiffness matrix; with it the domain of the integration is effectively restricted to a fraction of the full element. Now, let us consider how to do the integral numerically. Since the matrix B is naturally calculated as a function of natural coordinates, we do the integral in these coordinates:

$$\mathbf{K}_{\text{partial}} = \int_e d^3\xi |J| T(X(\xi)) \mathbf{B}^T(\xi) \cdot C \cdot \mathbf{B}(\xi) \quad (5.23)$$

where $|J|$ is the determinant of the Jacobian matrix. Evaluation of the integrand is straightforward and is the same procedure for any type of element, as long as the type is specified so that the shape functions and their derivatives for that type can be calculated for arbitrary ξ_k . The transition function is a “user”-provided function in the sense that it must be provided for each geometry; it is given as a function of real space and is straightforward to calculate as a function of ξ_k by first evaluating the shape functions to obtain the real space position.

The hard part of the integral is determining the boundary of the domain in natural coordinates. In fact there are two boundaries: an “inner” boundary within which $T(X) = 1$ and an outer one—between the two $T(X)$ is a smooth function of real space position. We will cover the different models one at a time.

One-Brick: sphere in a cubic element

Our simplest case, a single cubic finite element, has a spherical atomistic region whose center coincides with that of the cube (section 5.4.1). The cube is a quadratic element, with 20 nodes (8 corner nodes and 12 midpoint nodes). The transition function is a function of distance from the center. Two “boundary radii”

are specified, Λ_1 and Λ_2 . $T = 1$ is one for $r < \Lambda_1$ and $T = 0$ for $r > \Lambda_2$. Because T depends only on R it is convenient to use spherical polar coordinates. This is true even in the natural elemental coordinates because the origin of the latter coordinates is also at the center of the sphere. The spherical polar coordinates corresponding to the ξ_k are ρ, θ, ϕ . The integral can then be broken into a radial part and an angular part. We do the radial part “first” (meaning it is the innermost loop of the computation). We need the values of ρ corresponding to Λ_1 and Λ_2 . This is simple, for even though the shape functions are quadratic, because the edge nodes are at the midpoints between corner nodes, they do not contribute to the position interpolation. Therefore position is simply a linear interpolation of the corner nodes and thus a linear function of ξ_k . Then, because the shape of the element in real space and natural coordinate space is the same and the origins coincide, the linear relationship is simply a scaling factor. The natural coordinates of the corners are $(\pm 1, \pm 1 \pm 1)$, giving a width of 2. So to convert from radial distance in real space to ρ we scale by $2/L$ where L is the real space width of the cube.

For a given angular position (θ, ϕ) , then, we do the radial integral in the $T = 1$ region and then the $0 < T < 1$ region, using standard Gauss quadrature. Only a few Gauss points are needed for the $T = 1$ region because the integrand is a low order polynomial—we have used 4 points⁹. In the varying T region we use 10 Gauss points, which seems to be sufficient. This procedure is enclosed in an outer loop for the angular integration, for which we use a degree-11, 50-point formula from Ref. [64], rather than separate loops over θ and ϕ .

⁹Probably two points would be enough: the displacements are quadratic (in ξ_k), so the strains are linear; the energy density being quadratic in strains is therefore quadratic in ξ_k .

Two-brick: Slab, two brick-elements

The second case is more complicated in that there are two elements, but the integration is easier than the One-Brick model, since there is no angular integration. The transition function depends on x only—the coordinate along which the two elements are stacked. $x = 0$ corresponds to the boundary between the elements. $T = 1$ for $|x| < X_1$, $T = 0$ for $|x| > X_2$, and smoothly interpolates between X_1 and X_2 (using the same function as in the One-Brick/sphere case). For each element we calculate the partial stiffness integral by separate integrations over the three natural coordinates ξ_1, ξ_2, ξ_3 . The ξ_1 integration corresponds to the x integration and is the innermost loop. The integration is done exactly like the ρ integration in the One-Brick model. The only thing different is how the limits are obtained—even though the transition function is symmetric with respect to the global $x = 0$ plane, this plane has a different relation to the natural coordinates for the two elements. In one element it is $\xi_1 = 1$ and in the other it is $\xi_1 = -1$. Thus the function that finds the integration limits must be passed the nodal positions for the current element; then it decides whether the element is to the “left” or “right” of $x = 0$, and chooses the ξ_1 limits accordingly.

Silicon plate: apices of wedge elements

In the Silicon-plate model, the elements containing the atomistic region are 15-noded wedge elements, see Fig. 5.5.1; see appendix 5.B for the shape functions. Moreover the edge nodes on in-plane edges connecting to nodes at the prescribed crack-tip are at the quarter-points, rather than the mid-points. This makes the relationship between natural coordinates and real space position quadratic rather

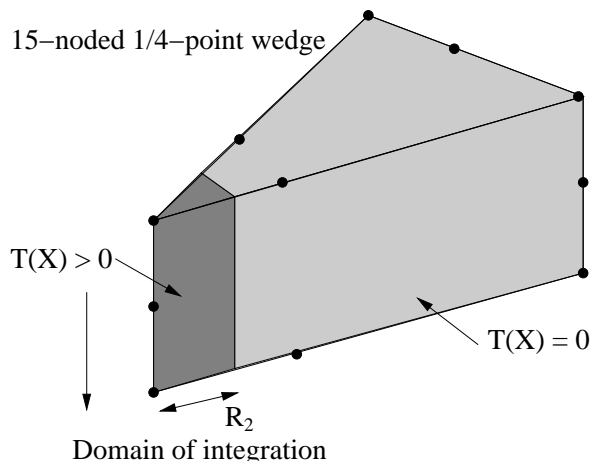


Figure 5.7: 15-noded wedge element with quarter-points.

than linear¹⁰; see Fig. 5.5.1, which shows the coordinates within a $\xi_3 = \text{const}$ plane. The edge nodes on edges pointing out-of-plane are still at the midpoints¹¹, so there is still a linear relationship between real space and natural coordinates in this direction. We use cylindrical natural coordinates ρ, ϕ, ζ . The transition function depends on the real space radial coordinate r in an identical way to how it does on the spherical radial coordinate in the One-Brick model. Thus again we do the radial integral first (innermost), followed by the angular integral, then the height integral. The limits for the radial integral need to be computed at run-time. Those for the angular integral turn out to be $0 < \phi < \pi/4$ (this will be shown below), and for the height integral, $-1 < \zeta < +1$.

To find the limits of integration in ρ at a given angle ϕ we need the inverse of the shape functions. Because the wedges are constructed by replicating a triangular 2D element, the in-plane nodal coordinates are independent of plane

¹⁰This is also what generates the square root singularity at the wedge ends.

¹¹as are the edge nodes on sides of the triangles making up the upper and lower faces of the element which are furthest from the designated crack tip.

(top/middle/bottom). This means that for the purposes of determining the in-plane position as a function of in-plane natural coordinates, which we need to determine integration limits, we can use the triangle shape functions. This can be seen by summing over shape functions corresponding to nodes with the same in-plane coordinates, which removes the ζ - (ξ_3 -)dependence and yields the triangle functions (they can also be obtained by setting $\zeta = -1$ and identifying the non-zero functions). The standard numbering of the triangle functions comes out almost automatically. Here are the triangle functions; the numbers in parentheses indicate which wedge functions are summed to obtain them. The full set of wedge functions is listed in the appendix.

$$N_0 = (1 - \xi_1 - \xi_2)(2(1 - \xi_1 - \xi_2) - 1) \quad (0, 3, 12) \quad (5.24)$$

$$N_1 = \xi_2(2\xi_2 - 1) \quad (1, 4, 13) \quad (5.25)$$

$$N_2 = \xi_1(2\xi_1 - 1) \quad (2, 5, 14) \quad (5.26)$$

$$N_3 = 4\xi_2(1 - \xi_1 - \xi_2) \quad (6, 9) \quad (5.27)$$

$$N_4 = 4\xi_1\xi_2 \quad (7, 10) \quad (5.28)$$

$$N_5 = 4\xi_1(1 - \xi_1 - \xi_2) \quad (8, 11) \quad (5.29)$$

This definition corresponds to numbering the nodes as shown in Fig. 5.5.1. Notice that the numbering goes clockwise. However the real space positions of crack-tip elements in the simulation are arranged in a counter-clockwise order, and the node actually at the crack-tip is node 1 (counting from zero!)¹² for each element, as shown in Fig. 5.5.1. The element shown has one side on the x -axis; obviously this will not be true in general. It turns out, however, that we do not need to deal with

¹²This is just how the mesh generator ordered the nodes.

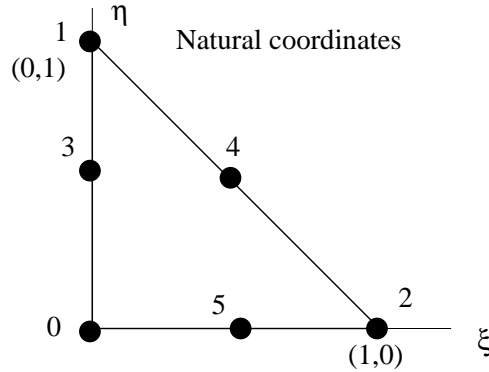


Figure 5.8: Natural coordinates and node numbering for wedge element.

the orientation explicitly. To convert from polar coordinates with origin at node 1, it is convenient to introduce alternative Cartesian natural coordinates ξ_1^*, ξ_2^* with origin at node 1. Thus, $\xi_1^* = \rho \cos(\phi)$, $\xi_2^* = \rho \sin(\phi)$. The transformation to then get the standard ξ_1, ξ_2 is then $\xi_1 = (\xi_1^* - \xi_2^*)/\sqrt{2}$, $\xi_2 = 1 - (\xi_1^* + \xi_2^*)/\sqrt{2}$. It is clear from the right hand diagram in Fig. 5.5.1 that the range of ϕ is $(0, \pi/4)$. All of the crack-tip elements in the model also have 45° angles about the crack tip, although this need not be assumed for the integration. Note that the boundary circles $r = R_1, R_2$ are not circles in natural coordinates. This is easily seen from the fact that the distances between nodes 0 and 1 and nodes 1 and 2 are equal in the model, whereas the corresponding lengths in natural coordinates are 1 and $\sqrt{2}$, respectively.

What is assumed for the integration is the following: the ordering of the nodes (i.e., node 1 is at the crack tip—the origin for the integration, and the counter-clockwise order), and that the triangles are isosceles with the edges containing node 1 both having length L . Denote the unit vectors along these edges by \hat{n}_1, \hat{n}_2 and the angle between them by γ (so $\hat{n}_1 \cdot \hat{n}_2 = \cos \gamma$). Now we can write the nodal

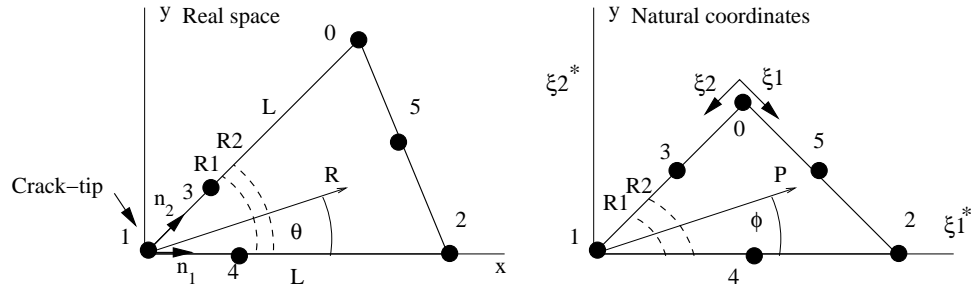


Figure 5.9: Wedge configuration in real space, and natural coordinates correspondingly rotated.

positions, with respect to the origin of the polar coordinates, in terms of these, as

$$\vec{R}_0 = L\hat{n}_2, \vec{R}_1 = \vec{0}, \vec{R}_2 = L\hat{n}_1, \vec{R}_3 = (L/4)\hat{n}_2, \vec{R}_4 = (L/4)\hat{n}_1, \vec{R}_5 = \frac{1}{2}L(\hat{n}_1 + \hat{n}_2) \quad (5.30)$$

Multiplying by the shape functions, summing and rearranging, we get the position within the plane in terms of in-plane natural coordinates ξ_1, ξ_2

$$\vec{r} = L(1 - \xi_2)(\xi_1\hat{n}_1 + (1 - \xi_1 - \xi_2)\hat{n}_2) \quad (5.31)$$

Next we substitute for ξ_1, ξ_2 in terms of ξ_1^*, ξ_2^* giving

$$\vec{r} = L\frac{1}{\sqrt{2}}(\xi_1^* + \xi_2^*)\left(\frac{1}{\sqrt{2}}(\xi_1^* - \xi_2^*)\hat{n}_1 + \sqrt{2}\xi_2^*\hat{n}_2\right) \quad (5.32)$$

$$= L(\xi_1^* + \xi_2^*)\left(\frac{1}{2}(\xi_1^* - \xi_2^*)\hat{n}_1 + \xi_2^*\hat{n}_2\right) \quad (5.33)$$

$$= L\rho^2(\cos\phi + \sin\phi)\left(\frac{1}{2}(\cos\phi - \sin\phi)\hat{n}_1 + \sin\phi\hat{n}_2\right) \quad (5.34)$$

We need the relationship between real space radial distance, which is $|\vec{r}|$, and ρ , for any given value of ϕ . Thus we take the magnitude of \vec{r} :

$$|\vec{r}| = L\rho^2(\cos\phi + \sin\phi) \left(\frac{1}{4}(\cos\phi - \sin\phi)^2 + \sin^2\phi + (\cos\phi - \sin\phi)\sin\phi\cos\gamma \right)^{\frac{1}{2}} \quad (5.35)$$

which gives ρ in terms of a real space radius R (such as one of the boundary radii R_1, R_2 in the definition of the transition function) and ϕ :

$$\rho = \left(\frac{R}{L(\cos\phi + \sin\phi) \left(\frac{1}{4}(\cos\phi - \sin\phi)^2 + \sin^2\phi + (\cos\phi - \sin\phi)\sin\phi\cos\gamma \right)^{\frac{1}{2}}} \right)^{\frac{1}{2}} \quad (5.36)$$

Now, when doing the integral, in the innermost loop, there is a particular value of $\zeta (= \xi_3)$ and ϕ . Eqn. 5.36 is used to provide the limits for ρ for the $T = 1$ part ($0 < R < R_1$) and then for the $T < 1$ part ($R_1 < R < R_2$). For each ρ, ϕ, ζ triple, first the alternative Cartesian natural coordinates $\xi_1^*, \xi_2^*, \xi_3^* = \zeta$ are computed, then the normal natural coordinates $\xi_1, \xi_2, \xi_3 = \xi_3^*$, then the shape functions and their derivatives are used to generate real space positions and the stiffness integrand, etc.

Cracked cube: sections of tetrahedra

In the previous case matters were simplified somewhat by the fact that the same geometrical relation between the integration region and the natural coordinates holds for all atom-containing elements, and thus the same coordinate transformation could be used in all cases. This was fortuitous; we should not expect this in general, and indeed, it is not the case for the tetrahedra (“tets”) in the Cracked-Cube model. Of course, the geometry is still quite special: the atomistic region is a sphere centered on the midpoint of an edge between two nodes. The atom-

containing elements are all tetrahedra which contain this edge; there turn out to be five, though it does not matter to the code. The other thing that does matter, the other assumption being made, is that the cube is small enough so that it does not intersect any *other* tets outside the five. The integration is done as a series of integrals in spherical polar (natural) coordinates ρ, θ, ϕ (we use ρ as the radial coordinate as a reminder that these are natural coordinates rather than real space coordinates). These coordinates are always chosen so that the polar axis corresponds to the mesh edge containing the center. As an aid in constructing the transformation rules an alternative set of Cartesian natural coordinates ξ'_a is introduced, where $\xi'_1 = \rho \sin \theta \cos \phi$, $\xi'_2 = \rho \sin \theta \sin \phi$, $\xi'_3 = \rho \cos \theta$.

The outermost integration is the azimuthal angular ϕ one, then the polar angular θ one, then the radial one. The limits for the θ integral are always 0 to 2π . Those for ϕ depend on the node ordering, and turn out to be one of two possibilities. The ρ integral, as in previous cases, is broken into two parts, one for $T = 1$ and one for $0 < T < 1$, the bounds of these regions being set by the boundary radii. The corresponding limits for ρ are again found by inverting an expression for R in terms of ρ, θ, ϕ . A further complication, compared to the wedge case, is that this must be done numerically.

The midpoint nodes for the tetrahedra in this model are all in fact at the midpoints in real space of their edges, and so do not play a role in the relation between natural coordinates and real space position. Thus we need only be concerned with the corner nodes (0,1,2,3). The problem is then for a given tet, which nodes are which: there are $4! = 24$ different possibilities, corresponding to the different permutations of (0, 1, 2, 3). Looking in natural coordinate-space, as in Fig. 5.5.1, the atomistic sphere can be on the midpoint of any of the six edges. Notice that at

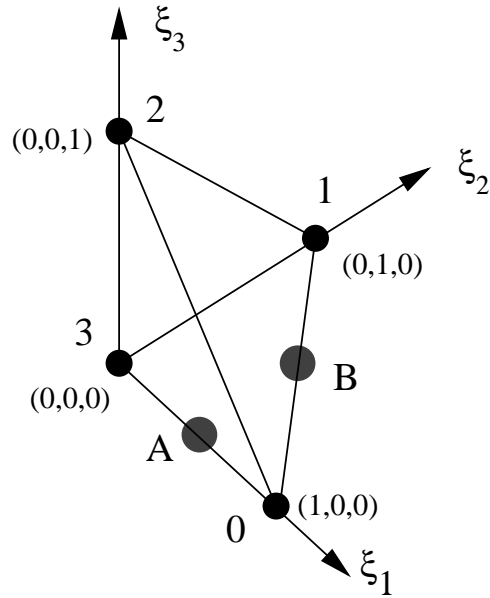
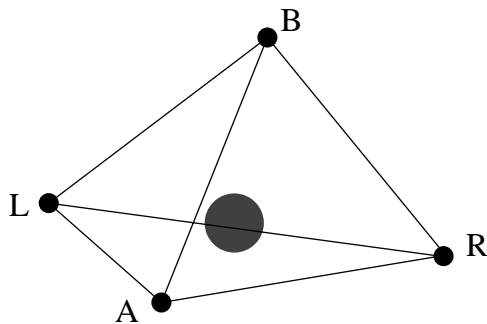


Figure 5.10: Natural coordinates and node numbering for a tetrahedron, and possible locations of the atomistic region.

three of these positions, such as that marked A in the figure, the planes forming the edge meet at right angles, whereas at the other three (e.g. position B) the planes meet at a different angle, which turns out to be $\cos^{-1}(1/3)$.

To determine the transformation from ξ'_a to ξ , and the limits for ϕ , we must consider each of the 24 cases in turn. To organize the labeling, consider labeling the nodes in terms of their real space positions as in Fig. 5.5.1. We imagine looking at the crack front from the front of the cube (the face into which the crack has been cut, which happens to be the face with normal $(0, 0, 1)$). The two nodes bounding the mesh edge on which the atomistic region is sited we call L and R. These are common to all of the tets containing atoms. To label the other two nodes for each tet we make a loop over the tets starting from the one touching the upper crack surface. For each tet we label the non-LR nodes A and B in the order encountered



caption Labeling of nodes according to real space positions.

when making this loop. We have some choice in the orientation of the ξ'_a —we have already chosen the polar axis (ξ'_3) to lie along the given mesh edge, but we must also choose the orientation of ξ'_1 and ξ'_2 axes, equivalently the origin for ϕ . For cases where the range of ϕ is $\pi/2$, it makes sense to have the ξ'_1 and ξ'_2 axes coincide with the planes forming the edge. When the angle between the planes is $\cos^{-1}(1/3)$, we can only choose one of the axes to coincide with a plane. We choose the plane which is a coordinate plane (i.e., a $\langle 100 \rangle$ -type plane) to include a ξ' axis (the other plane being $\langle 111 \rangle$ plane). Which axis (i.e., ξ_1 or ξ_2) is determined by the requirement that the ξ' coordinate system be right-handed¹³ and that the range of ϕ within the tet is within the range $0 < \phi < \pi/2$. The actual ranges are then either $0 < \phi < \cos^{-1}(1/3)$ or $\pi/2 - \cos^{-1}(1/3) < \phi < \pi/2$. This is simply a choice that was made—other choices of defining the orientation would be equally valid.

¹³Note that what really has handedness are the transformations between coordinates and real space positions, not the coordinate systems themselves, since we can always make a diagram of coordinate axes and draw them in a right-handed sense, even if they are left-handed. Standard usage calls a coordinate system right handed if the unit vectors in real space which point in the respective directions of increasing coordinate form a right-handed triad. The ξ (no prime) system is left handed for half of the 24 cases and right-handed for the rest, but by our prescription the ξ' is right-handed in all 24.

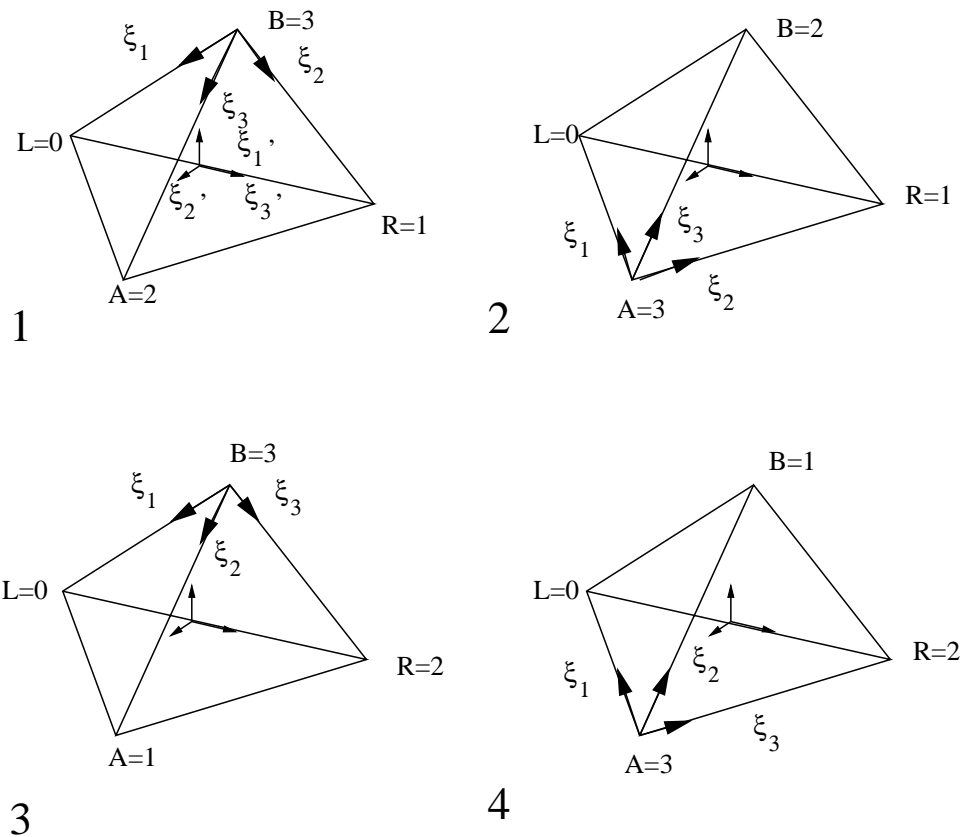


Figure 5.11: Coordinate systems for first four tetrahedron orientations.

We label the different cases by the list of internal node-numbers corresponding to nodes L,R,A,B respectively. For example, case 1 is labeled 0123, meaning node L has internal node number 0, node R has internal node 1, A and B internal nodes 2 and 3 respectively. Fig. 5.5.1 illustrates the setting up of the ξ' coordinate system for the first four cases (0123, 0132, 0213 and 0231).

From studying each figure one can write down expressions for the ξ' in terms of the ξ . For case 1 we have

$$\xi'_1 = \xi_2 \quad (5.37)$$

$$\xi'_2 = \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}(\xi_1 + \xi_3) \quad (5.38)$$

$$\xi'_3 = \frac{1}{\sqrt{2}}(-\xi_1 + \xi_3) \quad (5.39)$$

the inverse of which is

$$\xi_1 = \frac{1}{2} - \frac{1}{\sqrt{2}}(\xi'_2 + \xi'_3) \quad (5.40)$$

$$\xi_2 = \xi'_1 \quad (5.41)$$

$$\xi_3 = \frac{1}{2} - \frac{1}{\sqrt{2}}(\xi'_2 - \xi'_3) \quad (5.42)$$

The transformations for all 24 cases can be written in terms of a vector v and a matrix m , where $\xi_a = v_a + m_{ab}\xi'_b$. The complete table of v and m is listed in appendix 5.D.

Testing the partial stiffness matrices

In order to test that we have coded the integration correctly, we first replace the stiffness integrand by a matrix filled with ones, set the two boundary radii to be equal (so that there is a sharp cutoff in the transition function) and check that the result of the integration is the volume of the portion of the element within the boundary radius. The next check is to load the system so that there is a strain, making all of the atoms constrained and thus subject to the nodal displacements, and check whether the partial elastic energy (the elastic energy computed using the partial stiffness matrix and the nodal displacements) roughly agrees with the atomistic energy. The nodal forces may also be compared. Of course they should

not agree exactly—the difference is the whole point—but for small deformations one would expect reasonably close agreement. However several things can interfere with this.

1. Free surfaces cause a large contribution to the atomistic energy which is not reflected in the elastic energy. For testing, sometimes periodic boundary conditions can be applied to remove these effects.
2. When using quadratic elements, the strain is in general inhomogeneous, unless special care is taken to set up a homogeneous strain. Because the interatomic potential is non-local, it is sensitive to strain inhomogeneities in way that the continuum elastic energy is not (being purely local). The energies agree when a homogeneous strain is applied, but even then the nodal forces differ noticeably.
3. When the lattice is not a simple Bravais lattice—Silicon, for example, if the actions to be described in section 5.5.3 have not been taken, for a proper comparison between atomistic and continuum elastic energy, it is important to use the “unrelaxed” elastic constants of the potential, which are higher than the relaxed ones (the latter being the correct ones, presumably close to the experimental values).

5.5.2 Linear solve step

The linear solve step solves the equation

$$\mathbf{K}_{\text{total}} \cdot \mathbf{q} = \mathbf{F}'_{\text{load}} \quad (5.43)$$

where the right-hand side (RHS) vector, $\mathbf{F}'_{\text{load}}$, is the applied nodal loads plus the corrections coming from previous iterations of the OFE/MD process. Eqn. 5.43 is subject to displacement boundary conditions (there need to be always sufficient displacement boundary conditions to obtain a unique solution, given that the stiffness matrix always has six zero eigenvalues corresponding to rigid-body motions.). These are discussed in the next section. The solver we use depends on the size of the matrix. For our small models, One-Brick and Two-Brick, it is convenient to use Python's Linear Algebra module, which ultimately uses Lapack routines to solve the equations. For the Cracked-Cube model, however, this does not work, because there are over 10,000 nodes, and thus 30,000 degrees of freedom. Storing the full matrix would require space for a billion doubles, or 8 Gigabytes of RAM. Luckily this is not necessary, because the stiffness matrix is sparse (matrix elements between different degrees of freedom are non-zero only if they come from nodes on the same element). The finite element library we use includes a sparse matrix class, and a sub-class designed specifically for elasticity—the difference being that matrix elements are stored in 3×3 blocks (reflecting the three components of displacement for each node, hence nine pairings for each pair of nodes). The solver associated with this class is an iterative one, and uses the conjugate gradients algorithm to find the solution. For meshes that are large but can still be solved directly, the sparse solver is much faster, although when there are elements with large aspect ratios (such as in the silicon model if the thickness is chosen too small) it has difficulty converging.

Adjustments for boundary conditions

To account for displacement boundary conditions three things need to be done in the linear elasticity equations:

1. For each fixed displacement (out of a possible $3N_{\text{nodes}}$) the elements of the corresponding row and column of the stiffness matrix need to be set to zero, and the diagonal element replaced with unity. This effectively removes this degree of freedom from the coupled set of equations.
2. The corresponding entry in the right hand side vector should be replaced by the fixed value that that displacement is supposed to hold; because of the 1 on the diagonal of the stiffness matrix this value will simply be copied into the solution vector.
3. If the fixed value of the boundary condition is not zero, terms corresponding to those which were set to zero in the stiffness matrix need to be added (with a minus sign) to the corresponding elements of the right hand side vector (see Ref.[16], p. 65).

Items (1) and (3) need only be done once, at the beginning of the iteration procedure (the existence of the nonlinear terms does not change these). Item (2) needs to be applied to each correction term that gets added to the right hand side.

5.5.3 Subtleties

Subtleties with periodic boundaries

If the atomistic region has periodic boundary conditions in some direction, the location of the boundaries should correspond with the finite element model's bound-

aries. To make the finite model consistent with periodic boundaries, some adjustments should be made to the finite element model. First, corresponding boundary nodes on opposite sides from each other should be constrained to have equal displacements, and zero displacement in the periodic directions¹⁴. The degrees of freedom associated with these nodes would be reduced in number; for example if there were periodic boundaries in the y and z directions, corner nodes with common x -position would be considered one node, with one degree of freedom: displacement in the x -direction. Accordingly the forces on the four nodes would be summed to give the force on the constraint degree of freedom.

Placing atoms “according to linear elasticity”

Generally when we wish to apply a displacement formula from a linear elastic solution to find displaced atomic positions, we calculate a displacement by plugging the undeformed position into the formula for a Lagrangian description or do as described in Chapter 2 for an Eulerian description. However this is only correct as it stands for lattices with just one atom per unit cell (primitive Bravais lattices). Otherwise there may be internal relaxation within the unit cell which cannot be predicted by linear elasticity. This has been discussed in the context of the quasi-continuum method by Tadmor et al.[71]. Let us consider the case of silicon. The primitive lattice is FCC and there are two atoms per unit cell. If we choose one, atom 0, to be at the origin then the other, atom 1, is at $(1/4, 1/4, 1/4)$ (in units of the lattice constant). We can think of the whole lattice as being composed of two FCC sublattices A and B, with origins at positions of atoms 0 and 1 just mentioned. We can characterize the internal relaxation in different ways, one way

¹⁴Though it may be possible to implement variable periodic lengths in the manner of Parinello-Rahman dynamics.

being the following. Suppose a uniform strain ϵ_{ab} is applied. Each lattice will be deformed according to this strain, and in addition there will be some relative displacement of the lattices.

Without loss of generality we can set the relaxation displacement of sublattice B (which has a site, atom 1, at $(1/4, 1/4, 1/4)$) to be zero—we say this sublattice deforms *normally*; then we ask what the displacement of atom 0 is. Note that atom 0 does not move in normal deformation since it is at the origin. Using lower case for current positions and upper case for undeformed positions, we can write for a general atom in sublattice A:

$$\vec{x}^{(1)} = \vec{X}^{(1)} + \epsilon \cdot \vec{X}^{(1)} + \vec{v}(\epsilon) \quad (5.44)$$

where lengths are in units of the (undeformed) lattice constant. We assume the \vec{v} is an analytic function of the strain and consider linear response. Thus

$$v_i = \sum_{j,k} K_{ijk} \epsilon_{jk} \quad (5.45)$$

for some tensor K . Symmetry considerations (see appendix 5.E) show that K has the form

$$K_{ijk} = K_{123} |\epsilon_{ijk}| \quad (5.46)$$

where ϵ_{ijk} is the usual alternating tensor. K_{123} is quite straightforward to measure within a script which calculates elastic constants. Values for silicon potentials are shown in table 5.1. Note that if we changed which sublattice was held fixed, the definition of K would change by a minus sign. In the literature there exists another characterization of the internal strain. Kleinman[46] defines a parameter ζ in terms of the relaxation along the $[111]$ direction during a uniaxial strain in

Table 5.1: Internal relaxation parameters K_{123} and ζ for Si potentials.

potential	K_{123}	ζ
SW(1)	-0.81	1.63
SW(2)	-0.70	1.40
EDIP	-0.26	0.52
MEAM	-0.37	0.74

that direction. It is zero for zero relaxation and unity for the amount of relaxation which makes the nearest neighbor bond lengths equal. The experimental value[20] of ζ is 0.73. It is straightforward to relate K_{123} to ζ ; the relation turns out to be simply a factor of -2, which is evident in table 5.1.

What consequences does internal relaxation have for OFE/MD? They are the following: we can no longer simply apply the calculated elastic displacements (interpolated from the nodal displacements) to the undeformed positions of the atoms to get their correct current positions. We should also add the internal relaxation, which we know as a function of strain. From a practical point of view, the first thing this means is treating the two sublattices separately. The DigitalMaterial framework naturally handles this since the ListOfAtoms class can have an arbitrary tree-structure. Thus the branch that contains the constrained atoms would be split into two separate branches, one of which would be sublattice A, the other sublattice B. The second practical issue is that the relaxation is a function of the strain, which should therefore be available along with the displacement—this is straightforward to compute from the shape function derivatives, which are available. Finally, a conceptual point: first, to avoid favoring one sublattice over the

other, it would probably be best to give half of the relative displacement to each one, that is, apply a relaxation-displacement of $\frac{1}{2}\vec{v}$ to sublattice A atoms, and one of $-\frac{1}{2}\vec{v}$ to sublattice B atoms. These relaxation displacements have not yet been included in the OFE/MD software, but they should certainly be among the next steps in its development.

Ghost forces

The term “ghost forces” comes from the quasi-continuum method, where it is observed that the ground state lattice for the interatomic potential being used is not, in fact, the ground state structure of the quasi-continuum model, which is clear from the fact that starting with an unstrained perfect lattice, atomic relaxations occur, yielding a different (and non-periodic) structure as the minimum energy structure. The cause can be traced to atoms near the boundary between “local” and “non-local” regions. Non-local atoms contribute their own atomic energy to the total energy. Take a non-local atom within a cut-off distance of a coarse element. It has a full set of neighbors from which to compute its energy. However some of these neighbors are in the coarse element. In the perfect lattice this atom should have zero force—not just because forces due to its neighbors cancel out, but because forces on this atom due all atomic energies which depend on its position, the atomic energies of all its neighbors, cancel out. Thus it makes a difference if the atomic energy of one or more neighbors is not counted. This is precisely the case for those neighbors which lie in the coarse element. Their energies are not counted individually. Rather the energy of the whole element is given as a function of the strain. Thus an imbalance arises which is entirely due to the difference between a local energy functional (even if it contains the nonlinearity of

the interatomic potential for homogeneous deformations) and the true non-local interatomic interactions. The term “ghost-forces” was originally introduced to describe non-conservative forces added to cancel the imbalance, although it is also used to refer to the force imbalances themselves. Adding non-conservative forces is undesirable because it makes dynamics difficult if not impossible (instabilities tend to occur).

Do ghost forces occur in OFE/MD? The answer is yes, but the real question is what effect they have on simulations. Their effects are most easily seen when no load forces are applied to the nodes. Initially the nodal displacements are zero and the atoms are in their perfect-lattice undeformed positions. The partial/full stiffness matrices cannot contribute to nodal forces, but there are non-zero forces on atoms near where the transition function is varying. Because atoms in the varying T region have different weights, the forces on them do not cancel out. Force appear on all the constrained atoms in the simulation (even those with zero transition function).

To quantify the ghost forces and their affects, consider the One-Brick model, and assume the atomistic region is a small fraction of the cubic element’s volume. We can quantify the dependence of the strain at the element center when the ghost forces have been relaxed in terms of the radius of the core region R_{core} , the width of the transition region (over which T goes from 1 to 0), w_T and the element size L . The atomic forces are distributed more or less evenly and symmetrically about the sphere; the total force is of course zero. The distribution may be characterized by its dipole moment tensor

$$m_{ab} = \sum_i X_a f_b \quad (5.47)$$

where the sum is over all atoms (only constrained atoms contribute). It turns out that this tensor is proportional to the identity matrix, so the distribution is characterized by a single number, m . m is observed to be a positive number, and a quadratic function of R_{core} (not surprising given that the number of boundary atoms varies as R_{core}^2) as well as a quadratic function of w_T . The sum involves a good bit of cancellation—typical terms are 3 orders of magnitude greater than the final total—so it is hard to estimate this sum analytically. From the observed quadratic dependence and dimensional analysis it must have a form like

$$m = \sum_{i,a} X_a f_a \alpha \sim E_0 (\beta R_{\text{core}}^2 + \gamma w_T^2) / a^2 \quad (5.48)$$

where E_0 is an atomic energy, a the lattice spacing and α, β, γ are numerical constants; α in particular is very small. Let us estimate the strain that a force characterized by m causes. This is where the approximation that the atomistic sphere is small compared to the cubic element comes in: we can calculate the effect of a point dipole moment, consisting of six equal point forces $\pm m/(3\delta)$ placed at positions $(\pm\delta/2, 0, 0), \dots$ in natural coordinates, for convenience (taking be care to put the coordinate scale factor $L/2$ in the right place), and take the limit $\delta \rightarrow 0$. If the atomistic region were not small, higher order moments would have to be included. This is straightforward, involving products of the point forces and the shape functions evaluated at the appropriate points, and gives a well-defined set of forces on the nodes, proportional to m/L , with coefficients of order 0.1 (see appendix 5.F). Applying these forces as a right hand side for the finite element equations, using the boundary conditions of the One-Brick model gives nodal displacements from which we can calculate the strain at the center of the

element¹⁵. Since $K \sim CL$, where C is an elastic modulus, the strain is $\epsilon \sim q/L \sim (f/(CL))/L = f/(CL^2) \sim 0.1m/(CL^3)$. For an element size of 30, using Holian's Lennard-Jones potential, this is about $10^{-6}m$. In fact, in this case the ghost-force-strain tensor is

$$\epsilon_{GF} = m10^{-6} \begin{pmatrix} 1.098 & 0 & 0 \\ 0 & 2.06 & -0.0165 \\ 0 & -0.0165 & 2.06 \end{pmatrix} \quad (5.49)$$

For R_{core}, w_T of order (10, 3), for this potential, m is of order 0.01; thus the strains are of order 10^{-8} , and therefore unlikely to affect the simulation to any extent. Note that cubic symmetry is broken by the boundary conditions being applied to the $x = -L/2$ face. Since the ghost-force-strain scales as $1/L^3$, it should decrease as $1/L$ if the element and atomistic region are scaled together (since in that case, $m \sim L^2$). To get a significant strain, say 10^{-3} , R_{core} would certainly have to be greater than L , which cannot be (assume $\alpha = 1, E_0 \sim Ca^3$, then $(R/L)^2$ would have to be greater than $10(L/a)$).

Presumably a similar picture holds for other geometries. The scaling with L is certainly general. The full force-moment tensor would be required in non-spherical geometries, but the point that the force distribution can be characterized in this way is still valid (subject of course to the assumption regarding the relative size of atomistic region and element). The dependence of the force-moment of the size of the atomistic region may be different when non spherical shapes are used, or when it is less symmetrically placed within the element.

¹⁵This is of course a linear analysis—the actual nodal displacements upon relaxing the system will be different because atomistics will be involved; however the difference is probably no bigger than the error from concentrating the dipole at the origin.

5.6 Infrastructure

5.6.1 Software engineering

We will now describe, without going into excessive detail, the software design. From a software point of view OFE/MD may be considered an application which uses the DigitalMaterial MD library as well as a library, FemLib, developed by Paul Wawrzynek to carry out the computation of elemental quantities such as elemental stiffness matrices, integration of surface tractions, etc. It also includes a sparse-matrix class incorporating a conjugate gradient solver. Like DigitalMaterial, it has a Python interface (although this was hand-coded, without using SWIG). A python class PyFemModel provides acts as a container for mesh information, which it reads from a file. All of the OFE/MD application code is written in Python. For the most part there is no performance penalty associated with this: most of the time is spent solving the matrix (C++, sparse solver) and doing the atomistic dynamics/minimization (DigitalMaterial).

The Python code is organized in various classes, whose names and relationships, and some of whose functionality, is indicated in the class diagram in Fig.5.6.1. There are three main classes which are independent of the particular problem being considered: FEMMD_Mover, FEMMD_MainObjects and FEMMD_Coupling. I will briefly describe these:

FEMMD_Mover Implements the iterative algorithms described in section 5.3.

This has been separated from the rest in the anticipation that future movers will be implemented for different dynamical algorithms, but will interact with the main data structure in the same way.

FEMMD_MainObjects The main data structure, a container for the primary

pieces of the simulation: the `ListOfAtoms`, the potential, the mover for the MD as well as the `PyFemModel`. `FEMMD_MainObjects` also contains a `FemLib` solver object—which contains the full stiffness matrix (adjusted for displacement boundary conditions) and has methods for solving the linear stiffness equations, the partial stiffness matrix, and a list of the values of the transition function for all of the atoms. As well as access to these objects, the `FEMMD_MainObjects` provides some methods which do some computation¹⁶): `CalculateWeightedForces()` and `CalculateWeightedEnergy()` for the atomic forces and energy, and `SolveMatrix()` which is a wrapper around the solve method of the `SparseCG` object. Having such functions in the same class as the data is appropriate since these will be always involved no matter what mover algorithm is employed, whereas the mover algorithm is expected to be variable and should be separated from those aspects which are not.

FEMMD_Coupling Provides the lowest level methods for computing atomistic quantities from finite element quantities and vice versa. These methods are (1) the constructor, to which is passed the list of undeformed positions of the atoms and which makes lists containing the elements containing each atom, the natural coordinates within the element for each atom, which elements contain atoms, which atoms a given element contains, and so forth; (2) `GetDisplacements`, which given an array of nodal displacements, returns the displacements for all atoms, computed from the shape function and (3) `NodalForcesFromAtomic`, which returns nodal forces having been passed atomic forces, again having used the shape functions. The class has been implemented in terms of NumPy arrays rather than data structures peculiar to

¹⁶thus it is not strictly a container class

the MD or FEM libraries to avoid it depending on their interfaces.

5.6.2 Interfacing with a continuum model via data base

A direction the Adaptive Software Project has taken is toward the use of relational data base, such as those used in the commercial world, for storing data associated with computations. This include mesh and boundary condition data used to specify the computation, as well as the results of a stage of the computation: displacements, temperatures, etc. We have therefore arranged the OFE/MD application to communicate with the data base in the case of the cube test model—since the mesh information for this model had been entered into the data base previously. Thus we obtain the mesh information from the data base, including details such as the location of the crack, so that the user does not need to know this separately, and the MD region will be placed at the crack front. At the end of the calculation, results from the MD computation are placed in the data base, to be made available to other computational modules.

Another designed feature of the Adaptive Software Project is the ability to run simulations via HTML, for example using a web browser. In the end, a researcher should be able to create and run large complicated jobs sitting at a terminal with a web browser, and the software needed to create the initial geometry of a model. The different parts of the computation, taking place at geographically remote locations will be scheduled automatically, with data being posted as \langle XML \rangle files available through http. As an intermediate step, the researcher will be still in the loop to facilitate the transfer of data from one step to the next—this is intended to be done using a web browser. We have implemented a simple web-interface to the OFE/MD application. It works as follows (see Fig. 5.14):

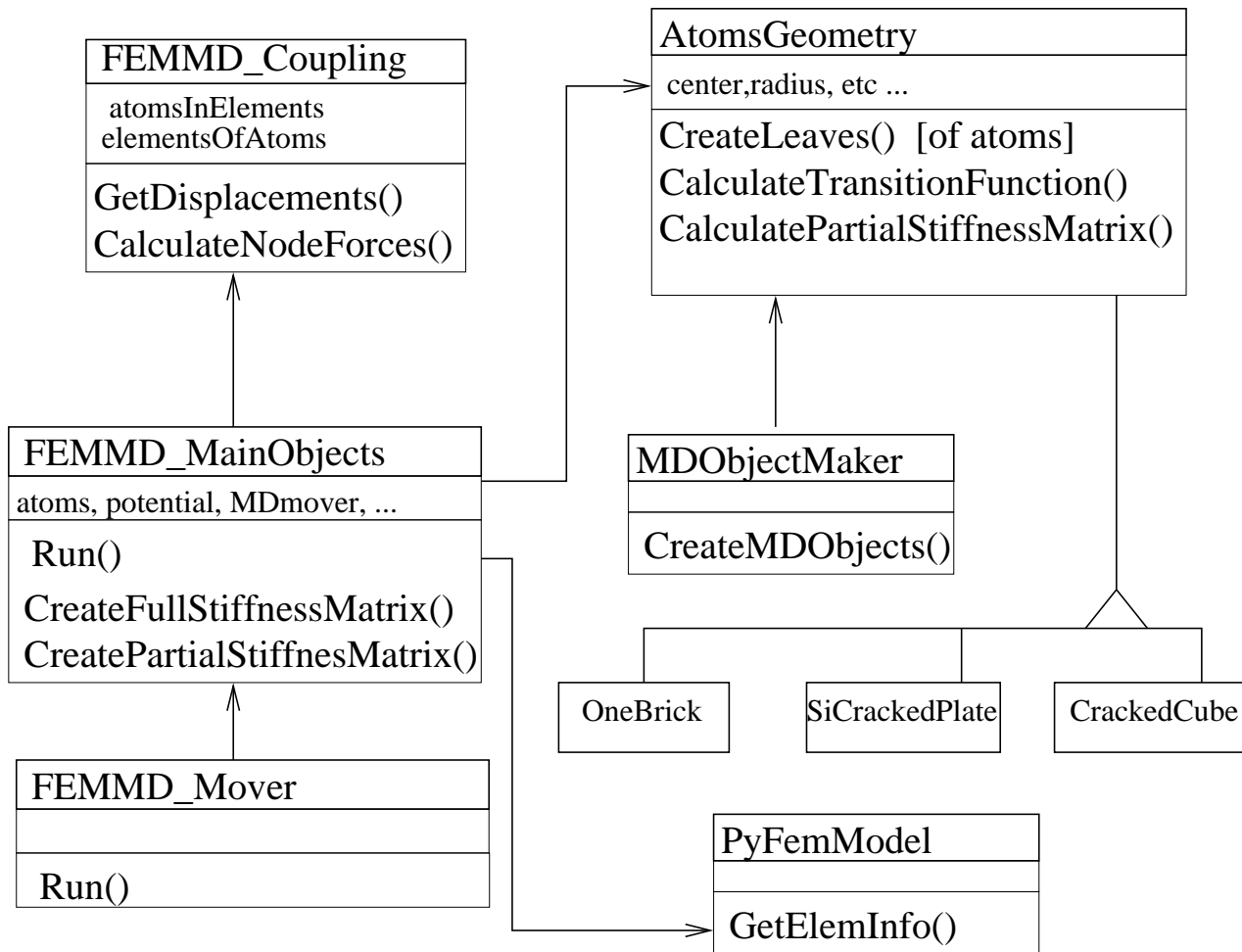


Figure 5.12: Class diagram for OFE/MD simulation software.

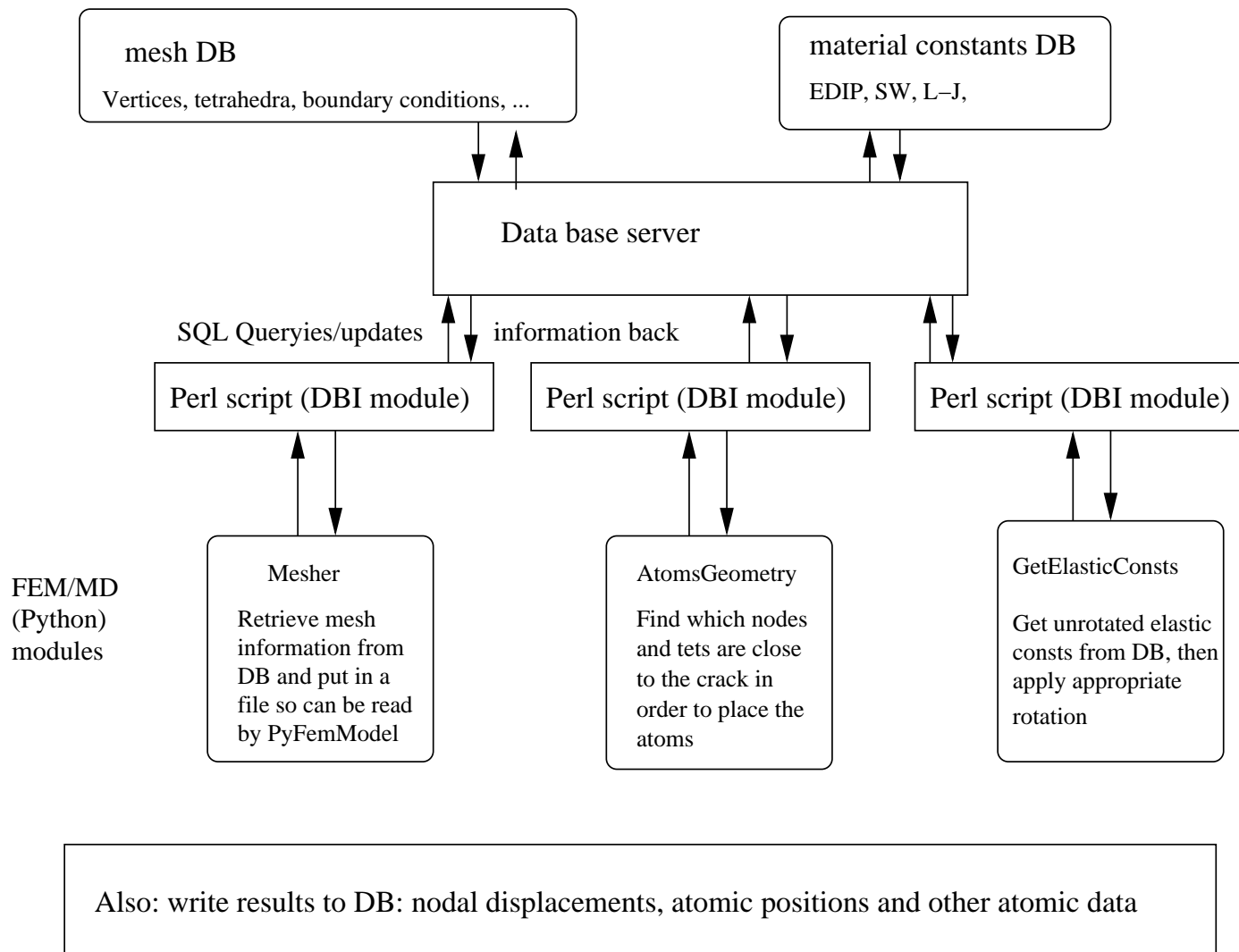


Figure 5.13: Use of a relational data base.

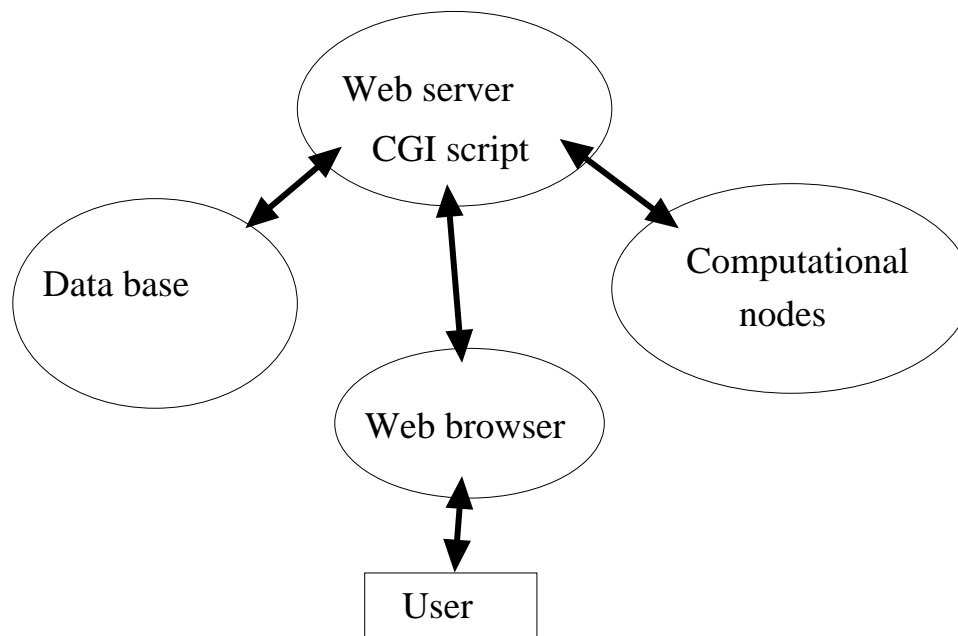


Figure 5.14: Web interface to OFE/MD simulations.

1. The user enters simulation parameters in a standard HTML form. These include the type of model, the location of the data base where mesh details are to be found, the type of material (interatomic potential), the crystal orientation for MD, the size of the atomistic region, etc.
2. The web server passes the parameters to the main OFE/MD program and runs it. Eventually it will send the job to specialized computational nodes rather than do the computation itself.
3. Results are returned in one or all of three ways: (i) simulation output may be used to update quantities such as displacements on the data base; (ii) print statements converted to HTML may be shown on the user's browser; (iii) a file containing simulation output may be emailed to the user.

5.6.3 Diagnostics, visualization and feature detection

An important intended feature of this software is the ability for the software to infer information about changes in the state of the system—at least in the atomistic region—at a high level. By “high level” we mean information that could be used to make decisions to adapt at the application level. Simply returning the final displacements of the nodes of the atomistic elements, which is the kind of information available from the software at this stage, is not particularly useful; this is very low-level. The next step would be to report changes in material properties such as stiffness and hardness by calculating stress-strain relations. Ultimately we would like the software to be able to identify features such as the following:

1. Nucleation of dislocations, characterized by the appearance of Burgers vectors around reasonably small loops of material.
2. Nucleation of voids, characterized by the appearance of points within the material which have no neighboring atoms.
3. Initiation or continuation of cleavage, characterized by the breaking of bonds across a given plane.
4. Change of phase, for example, melting, characterized by correlation functions/structure factors.

There do not exist algorithms, let alone software for most of these functions, except the last, since equilibrium thermodynamics, phase diagrams, etc., have long been applications of MD [3]. The automatic detection of different kinds of defects, particularly dislocations, is an area of active research by many researchers. We

have implemented a simple algorithm (as a class of type `ListOfAtomsObserver`, named `CrackPlaneObserver`) which monitors bond lengths across a given plane. It is not clear what is the most suitable function of the bond lengths across a whole plane to use to infer the growth of a crack. Perhaps the direction of growth should also be specified (a crack system, like a slip system requires two vectors to specify it fully: the normal of the crack plane and the direction of crack growth). The question then arises as to which crack system to monitor. Should one try to monitor all possible systems? This is probably too expensive. It is clear that there is much work to be done in determining the most efficient means to detect crack growth.

It is clear that we are not near the fully-automated stage; this is the case in the other aspects of the ASP also. It is recognized that it is appropriate to assume that humans will still be in the loop for a little longer. In the context of feature detection, this means using visualization to answer high-level questions about the state of the system. That is what has been used so far to identify crack growth in the model geometries (those which involve a crack). In this case it is known a priori where the crack is, and so a particular slice of the atomistic region can be visualized to highlight the crack.

5.7 Future applications of OFE/MD

Apart from use as a tool in the context of engineering-type modeling, we believe our formulation of coupling finite elements to molecular dynamics could be useful for physics applications—investigations of specific material processes, in particular with the aim of extracting parameters for high-level material descriptions. Some

possibilities are now discussed.

5.7.1 Cohesive law extraction

A typical technique used in continuum models of fracture is the *cohesive zone*. This replaces the picture of strictly stress-free crack faces near the crack tip; the faces within the cohesive zone feel an attractive force determined by some force-separation law—the cohesive zone law. Simple piecewise linear forms are usually used for the cohesive laws. In principle the parameters for these should emerge from atomistic simulations; however differences between the atomic length scale and the length scale of continuum models of fracture make it difficult to obtain realistic parameter values. The atomistic models are generally too small to contain the amount of damage (dislocations, voids, impurities) that exist in real materials, and thus the stress required to cause decohesion is unnaturally high. A technical problem that arises concerns the application of the load to the system—if forces are applied to atoms far from the crack region, at the high strain-rate generally required by atomistic calculations, separation of the pulled atoms from the free atoms can take place, which is not physical. Use of our OFE/MD formulation may alleviate this problem because there is more control over the whole region due to the overlapping descriptions. The simplest scheme would involve something like our existing “Two-Brick” model, but with a zero-volume cohesive zone element inserted between the two bricks on a larger scale containing significantly more atoms. A typical simple cohesive zone law would govern the separation of its two faces. The role of this would be to control the separation of the two halves. The actual force-displacement law would be expressed as the actual relative displacements experienced by the nodes in the cohesive zone as a function of the loading on the

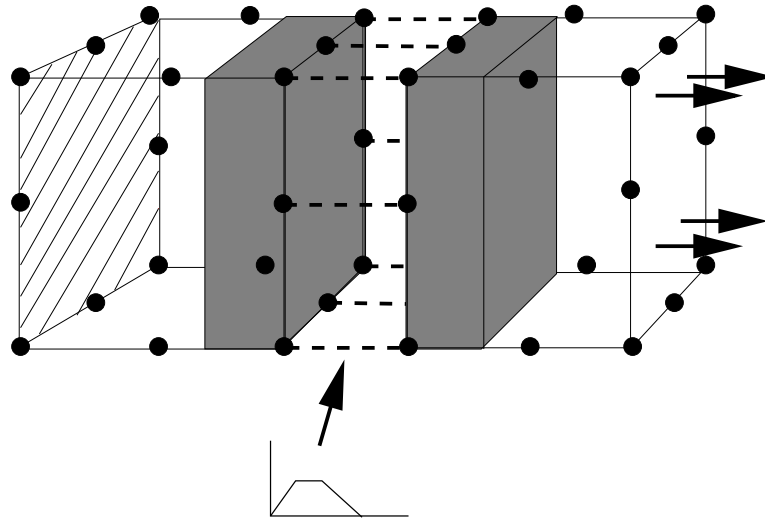


Figure 5.15: Simple cohesive zone model, with expanded view of the cohesive zone.

far faces of the model. In such an application periodic boundary conditions within the cleavage plane would be appropriate (see section 5.5.3). It would be possibly useful also to have several elements within the plane, to allow inhomogeneous separation across the plane.

5.7.2 Dislocations/surface/grain boundary interactions

While two of the model geometries presented have involved cracks, nothing done so far has involved dislocations. A goal of all designers of mixed atomistic-continuum formulations is to be able to study dislocations, and their interactions with each other and with other defects (such as surfaces or grain boundaries). In many cases atomistic simulations involving dislocations have free surfaces, making sure these are sufficiently far away that surface effects do not interfere with the dislocation behavior. One of the primary advantages of embedding an atomistic simulation in a continuum model is to provide realistic, physical boundaries to the core atoms,

and this is something that OFE/MD does well, without getting in the way of the dynamics of the core atoms. In order to apply OFE/MD to dislocation physics, it makes sense to choose a geometry that involves some nontrivial large scale elastic deformation, to make use of the capabilities of the finite elements. An example might be a free surface with an corner.

5.7.3 Continuum crack growth

An appealing possible application of the OFE/MD presented here is the direct measurement of crack growth from the simulation. The idea would be to make the *position* (not just the displacement) of the crack tip node(s) a dynamical degree of freedom. In a pure linear-elastic continuum model this would give a force tending to grow the crack because this relieves the elastic strain energy. With the atoms in the picture this would be opposed by the cohesive forces between atoms. One could then study the force and trajectory of this degree of freedom to obtain information about the crack growth at the continuum level, based on atomistic input. There would have to be significant enhancements made to OFE/MD to incorporate this new degree of freedom; changes in the position of a node cause changes in the natural coordinates of the atoms within their elements (and atoms near element boundaries may cross them)—these would have to be updated each time the node was moved (note that this need not be as frequently as updates are made to the nodal displacements). The derivatives of the natural coordinates would also need to be calculated in order to compute the force on the node. Furthermore the partial stiffness matrices would have to be recalculated each time—and derivatives of these with respect to the nodal position would also be required.

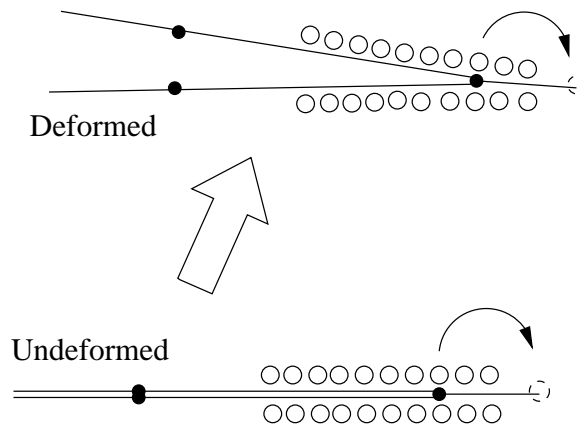


Figure 5.16: A crack growth step. Moving the actual node means changing how things look even in undeformed coordinates.

5.A Quadratic shape functions for hexahedral (brick) elements

Here are the shape functions for 15-noded (quadratic) wedge-elements. Figure 5.A shows the location of the nodes by number.

$$\begin{aligned}
N_0 &= \frac{1}{8}(1 - \xi_1)(1 - \xi_2)(1 - \xi_3)(-\xi_1 - \xi_2 - \xi_3 - 2) \\
N_1 &= \frac{1}{8}(1 - \xi_1)(1 - \xi_2)(1 + \xi_3)(-\xi_1 - \xi_2 + \xi_3 - 2) \\
N_2 &= \frac{1}{8}(1 - \xi_1)(1 + \xi_2)(1 - \xi_3)(-\xi_1 + \xi_2 - \xi_3 - 2) \\
N_3 &= \frac{1}{8}(1 - \xi_1)(1 + \xi_2)(1 + \xi_3)(-\xi_1 + \xi_2 + \xi_3 - 2) \\
N_4 &= \frac{1}{8}(1 + \xi_1)(1 - \xi_2)(1 - \xi_3)(\xi_1 - \xi_2 - \xi_3 - 2) \\
N_5 &= \frac{1}{8}(1 + \xi_1)(1 - \xi_2)(1 + \xi_3)(\xi_1 - \xi_2 + \xi_3 - 2) \\
N_6 &= \frac{1}{8}(1 + \xi_1)(1 + \xi_2)(1 - \xi_3)(\xi_1 + \xi_2 - \xi_3 - 2) \\
N_7 &= \frac{1}{8}(1 + \xi_1)(1 + \xi_2)(1 + \xi_3)(\xi_1 + \xi_2 + \xi_3 - 2) \\
N_8 &= \frac{1}{4}(1 - \xi_1)(1 - \xi_2)(1 - \xi_3^2) \\
N_9 &= \frac{1}{4}(1 - \xi_1)(1 + \xi_2)(1 - \xi_3^2)
\end{aligned}$$

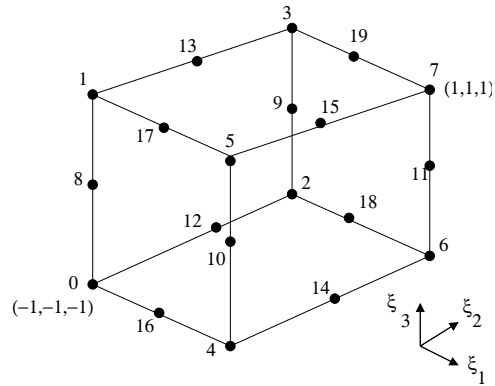


Figure 5.17: Node-numbering for 20-noded wedge elements.

$$N_{10} = \frac{1}{4}(1 + \xi_1)(1 - \xi_2)(1 - \xi_3^2)$$

$$N_{11} = \frac{1}{4}(1 + \xi_1)(1 + \xi_2)(1 - \xi_3^2)$$

$$N_{12} = \frac{1}{4}(1 - \xi_1)(1 - \xi_2^2)(1 - \xi_3)$$

$$N_{13} = \frac{1}{4}(1 - \xi_1)(1 - \xi_2^2)(1 + \xi_3)$$

$$N_{14} = \frac{1}{4}(1 + \xi_1)(1 - \xi_2^2)(1 - \xi_3)$$

$$N_{15} = \frac{1}{4}(1 + \xi_1)(1 - \xi_2^2)(1 + \xi_3)$$

$$N_{16} = \frac{1}{4}(1 - \xi_1^2)(1 - \xi_2)(1 - \xi_3)$$

$$N_{17} = \frac{1}{4}(1 - \xi_1^2)(1 - \xi_2)(1 + \xi_3)$$

$$N_{18} = \frac{1}{4}(1 - \xi_1^2)(1 + \xi_2)(1 - \xi_3)$$

$$N_{19} = \frac{1}{4}(1 - \xi_1^2)(1 + \xi_2)(1 + \xi_3)$$

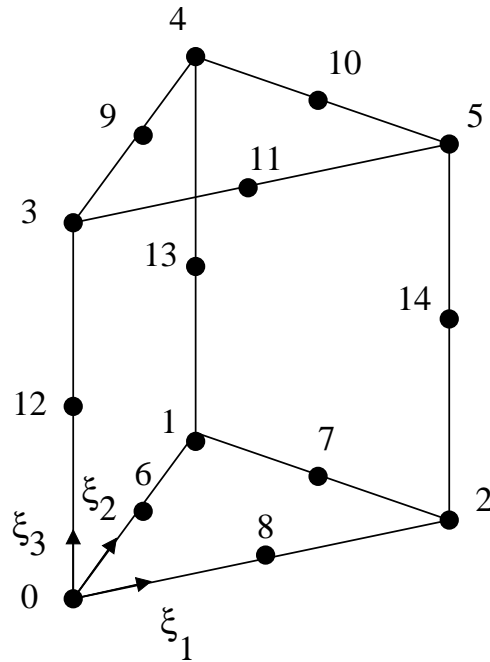


Figure 5.18: Node-numbering for 15-noded wedge elements.

5.B Quadratic shape functions for wedge elements

Here are the shape functions for 15-noded (quadratic) wedge-elements. Figure 5.B shows the location of the nodes by number. First define $\tau = (1 - \xi_1 - \xi_2)$.

$$N_0 = \frac{1}{2}\tau((2\tau - 1)(1 - \xi_3) - (1 - \xi_3^2))$$

$$N_1 = \frac{1}{2}\xi_2((2\xi_2 - 1)(1 - \xi_3) - (1 - \xi_3^2))$$

$$N_2 = \frac{1}{2}\xi_1((2\xi_1 - 1)(1 - \xi_3) - (1 - \xi_3^2))$$

$$N_3 = \frac{1}{2}\tau((2\tau - 1)(1 + \xi_3) - (1 - \xi_3^2))$$

$$N_4 = \frac{1}{2}\xi_2((2\xi_2 - 1)(1 + \xi_3) - (1 - \xi_3^2))$$

$$N_5 = \frac{1}{2}\xi_1((2\xi_1 - 1)(1 + \xi_3) - (1 - \xi_3^2))$$

$$N_6 = 2\xi_2\tau(1 - \xi_3)$$

$$N_7 = 2\xi_1\xi_2(1 - \xi_3)$$

$$N_8 = 2\xi_1\tau(1 - \xi_3)$$

$$N_9 = 2\xi_2\tau(1 + \xi_3)$$

$$N_{10} = 2\xi_1\xi_2(1 + \xi_3)$$

$$N_{11} = 2\xi_1\tau(1 + \xi_3)$$

$$N_{12} = \tau(1 - \xi_3^2)$$

$$N_{13} = \xi_2(1 - \xi_3^2)$$

$$N_{14} = \xi_1(1 - \xi_3^2)$$

5.C Quadratic shape functions for tetrahedral elements

Here are the shape functions for 10-noded (quadratic) tet-elements, as shown in Fig. 5.C. First define $\tau = (1 - \xi_1 - \xi_2 - \xi_3)$.

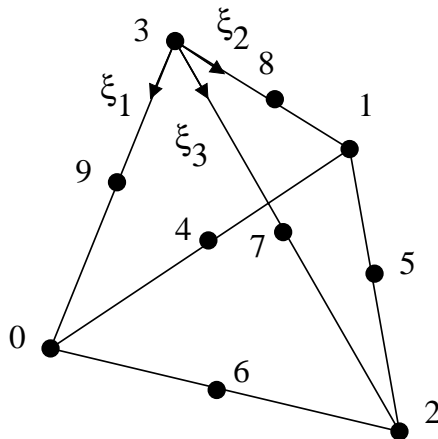


Figure 5.19: Node-numbering for 10-noded tetrahedron elements.

$$N_0 = (2\xi_1 - 1)\xi_1$$

$$N_1 = (2\xi_2 - 1)\xi_2$$

$$N_2 = (2\xi_3 - 1)\xi_3$$

$$N_3 = (2\tau - 1)\tau$$

$$N_4 = 4\xi_1\xi_2$$

$$N_5 = 4\xi_2\xi_3$$

$$N_6 = 4\xi_1\xi_3$$

$$N_7 = 4\xi_3\tau$$

$$N_8 = 4\xi_2\tau$$

$$N_9 = 4\xi_1\tau$$

5.D Transformations between natural coordinates in cracked-cube model

Here is the table listing the vector v and matrix m needed to construct the $\xi' \rightarrow \xi$ transformation for all 24 orientations of tet.

Table 5.2: Transformation vectors and matrices for transforming tetrahedron natural coordinates.

(LRAB)	v	m
(0,1,2,3)	$(\frac{1}{2}, \frac{1}{2}, 0)$	$\begin{pmatrix} -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \end{pmatrix}$
(0,1,3,2)	$(\frac{1}{2}, \frac{1}{2}, 0)$	$\begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \end{pmatrix}$
(0,2,1,3)	$(\frac{1}{2}, 0, \frac{1}{2})$	$\begin{pmatrix} -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}$
(0,2,3,1)	$(\frac{1}{2}, 0, \frac{1}{2})$	$\begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$
(0,3,1,2)	$(\frac{1}{2}, 0, 0)$	$\begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$

Table 5.2: Transformation vectors and matrices for transforming tetrahedron natural coordinates.

(LRAB)	v	m
(0,3,2,1)	$(\frac{1}{2}, 0, 0)$	$\begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$
(1,0,2,3)	$(\frac{1}{2}, \frac{1}{2}, 0)$	$\begin{pmatrix} -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \end{pmatrix}$
(1,0,3,2)	$(\frac{1}{2}, \frac{1}{2}, 0)$	$\begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \end{pmatrix}$
(1,2,0,3)	$(0, \frac{1}{2}, \frac{1}{2})$	$\begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}$
(1,2,3,0)	$(0, \frac{1}{2}, \frac{1}{2})$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$
(1,3,0,2)	$(0, \frac{1}{2}, 0)$	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}$
(1,3,2,0)	$(0, \frac{1}{2}, 0)$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$

Table 5.2: Transformation vectors and matrices for transforming tetrahedron natural coordinates.

(LRAB)	v	m
(2,0,1,3)	$(\frac{1}{2}, 0, \frac{1}{2})$	$\begin{pmatrix} -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix}$
(2,0,3,1)	$(\frac{1}{2}, 0, \frac{1}{2})$	$\begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$
(2,1,0,3)	$(0, \frac{1}{2}, \frac{1}{2})$	$\begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix}$
(2,1,3,0)	$(0, \frac{1}{2}, \frac{1}{2})$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$
(2,3,0,1)	$(0, 0, \frac{1}{2})$	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$
(2,3,1,0)	$(0, 0, \frac{1}{2})$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$
(3,0,1,2)	$(\frac{1}{2}, 0, 0)$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$

Table 5.2: Transformation vectors and matrices for transforming tetrahedron natural coordinates.

(LRAB)	v	m
(3,0,2,1)	$(\frac{1}{2}, 0, 0)$	$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$
(3,1,0,2)	$(0, \frac{1}{2}, 0)$	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$
(3,1,2,0)	$(0, \frac{1}{2}, 0)$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$
(3,2,0,1)	$(0, 0, \frac{1}{2})$	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
(3,2,1,0)	$(0, 0, \frac{1}{2})$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

5.E Symmetry considerations for internal relaxation parameter in a diamond-cubic lattice

We present the derivation of eqn. 5.46. Consider the what the form of K must be using cubic symmetry considerations. First note that it must be symmetric in the last two indices since these are contracted with the ϵ which is symmetric. Then using the equivalence of the three crystal directions we can write down \vec{v} as (looking at the 1 component for convenience):

$$v_1 = A\epsilon_{11} + B_1(\epsilon_{22} + \epsilon_{33}) + B_2(\epsilon_{12} + \epsilon_{21} + \epsilon_{13} + \epsilon_{31}) + C(\epsilon_{23} + \epsilon_{32}) \quad (5.50)$$

So far we have used the symmetry of the undeformed cube, since K corresponds to derivatives of \vec{v} evaluated in the undeformed state. In addition we can impose for certain deformations additional symmetry constraints. For example for a uniform dilation, the crystal still has cubic symmetry, so \vec{v} must do also, which constrains \vec{v} to be zero in this case. Setting ϵ proportional to the identity matrix gives $A + 2B_1 = 0$, so now we have (renaming $B_2 \rightarrow B$):

$$v_1 = A(\epsilon_{11} - \frac{1}{2}\epsilon_{22} - \frac{1}{2}\epsilon_{33}) + B(\epsilon_{12} + \epsilon_{21} + \epsilon_{13} + \epsilon_{31}) + C(\epsilon_{23} + \epsilon_{32}) \quad (5.51)$$

Next consider a uniaxial strain along a crystal axis, say the z axis. The undeformed lattice has the symmetry that if we rotate about the given axis by 90 degrees and reflect in the xy plane, the lattice goes to itself. The reflection is necessary to make sublattice B come back to itself after the rotation. Now, let

both sublattices deform normally; then we ask whether a further motion of atom 0 breaks any symmetry of the normally deformed lattice. The rotation plus reflection symmetry is not broken by a ϵ_{zz} strain, thus any relaxation must preserve the symmetry too. Clearly motion in any direction will break either the rotation part or the reflection part (and they cannot cancel out, because there is just one atom to consider). The result of this consideration is that the coefficient A above must vanish. Finally consider a shear in the xy plane, ϵ_{xy} . The normally deformed state no longer has the 90° rotation plus reflection symmetry, but it does have a 2-fold rotation symmetry (no reflection). If atom 0 displaces in the xy plane it breaks the 2-fold rotation symmetry. However it is allowed to move in the z -direction because this does not break that symmetry. The result of this is that coefficient B vanishes but not coefficient C . Thus only components of K with all indices different are nonzero; one way we could write it is thus:

5.F Nodal forces due to symmetric force dipole at the center of a cubic element.

For a dipole moment of strength m , the nodal forces are

$$F_{\text{node}} = \frac{m}{24} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \\ -2 & -2 & 0 \\ -2 & 2 & 0 \\ 2 & -2 & 0 \\ 2 & 2 & 0 \\ -2 & 0 & -2 \\ -2 & 0 & 2 \\ 2 & 0 & -2 \\ 2 & 0 & 2 \\ 0 & 2 & -2 \\ 0 & -2 & 2 \\ 0 & 2 & -2 \\ 0 & 2 & 2 \end{pmatrix} \quad (5.52)$$

Bibliography

- [1] F. F. Abraham, N. Bernstein, J. Q. Broughton, and D. Hess. Dynamic fracture of silicon: Concurrent simulation of quantum electrons, classical atoms, and the continuum solid. *Materials Research Society Bulletin*, 25:27, 2000.
- [2] F. F. Abraham, J. Q. Broughton, N. Bernstein, and E. Kaxiras. Spanning the continuum to quantum length scales in dynamic simulation. *Europhys. Lett.*, 44:783, 1998.
- [3] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 1987.
- [4] T. L. Anderson. *Fracture Mechanics: Fundamentals and Applications*. CRC Press, second edition, 1995.
- [5] Adaptive software project, 2002. (Cornell University, Mississippi State University, College of William and Mary).
- [6] N. P. Bailey, T. Cretegnny, M. Rauscher, C. R. Myers, and J. P. Sethna. Functional forms for anisotropic materials properties in multiscale modeling. *To be published*, 2003.
- [7] H. Balamane, T. Halicioglu, and W. A. Tiller. Comparative study of silicon empirical interatomic potentials. *Phys. Rev. B*, 46:2250, 1992.

- [8] M. I. Baskes. Modified embedded-atom potentials for cubic materials and impurities. *Phys. Rev. B*, 46:2727–2742, 1992.
- [9] M. I. Baskes, 2002. Private communication.
- [10] M. Z. Bazant, E. Kaxiras, and J. F. Justo. Environment-dependent interatomic potential for bulk silicon. *Phys. Rev. B*, 56:8542, 1997.
- [11] D. M. Beazley and P.S. Lomdahl. Message-passing multi-cell molecular dynamics on the connection machine 5. *Parallel Computing*, 20:173–195, 1994.
- [12] N. Bernstein et al. . In *Proceedings of the 9th DoD HPC Users Group Conference, Monterey, CA*, 1999.
- [13] N. Bernstein and D. Hess. Multiscale simulations of brittle fracture and the quantum-mechanical nature of bonding in silicon. *Materials Research Society Symposium Proceedings*, 653:Z2.7.1, 2001.
- [14] J. Q. Broughton, F. F. Abraham, N. Bernstein, and E. Kaxiras. Concurrent coupling of length scales: methodology and application. *Phys. Rev. B*, 60:2391, 1999.
- [15] Cornell fracture group home page.
- [16] T. R. Chandrupatla and A. D. Belegundu. *Introduction to Finite Elements in Engineering*. Prentice Hall, 1991.
- [17] C. P. Chen and M. H. Leipold. *Am. Cer. Soc. Bull.*, 59:469, 1980.
- [18] Xi Chen. A Lennard-Jones potential smoothly cut-off after third neighbors . 1999. Private communication.

- [19] W. Curtin, 2002. Private communication.
- [20] H. d'Amour, W. Denner, H. Schultz, and M. Caradona. A uniaxial stress apparatus for single-crystal X-ray diffraction on a four-circle diffractometer: application to Silicon and diamond. *Journal of Applied Crystallography*, 15:148, 1982.
- [21] B. Devincre and L. Kubin. Mesoscopic simulations of dislocations and plasticity. *Mat. Sci. Eng. A*, 234:8–14, 1997.
- [22] M. L. Dunn, W. Suwito, S. J. Cunningham, and C. W. May. Fracture initiation at sharp notches under mode I, mode II, and mild mixed mode loading. *Int. J. Fract.*, 84:367–381, 1997.
- [23] J. D. Eshelby, W. T. Read, and W. Shockley. Anisotropic elasticity with applications to dislocation theory. *Acta Metallurgica*, 1:251–259, 1953.
- [24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [25] P. C. Gehlen, J. P. Hirth, R. G. Hoagland, and M. F. Kanninen. A new representation of the strain field associated with the cube-edge dislocation in a model of α -iron. *J. Appl. Phys.*, 43:3921–3932, 1972.
- [26] P. C. Gehlen, A. R. Rosenfield, and G. T. Hahn. Structure of the $\langle 100 \rangle$ Edge Dislocation in Iron. *J. Appl. Phys.*, 39:5246–5254, 1968.
- [27] D. M. Goodstein, S. A. Langer, and B. H. Cooper. An efficient algorithm for the simulation of hyperthermal energy ion scattering. *Journal of Vacuum Science and Technology A*, 6:703–707, 1988.

- [28] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer-Verlag, 1983.
- [29] J. A. Hauch, D. Holland, M. P. Marder, and H. L. Swinney. Dynamic fracture in single crystal silicon. *Phys. Rev. Lett.*, 82:3823, 1999.
- [30] G. Henkelman, G. Jóhannesson, and H. Jónsson. Methods for finding saddle points and minimum energy paths. In S. D. Schwartz, editor, *Progress on Theoretical Chemistry and Physics*. Kluwer Academic Publishers, 2000.
- [31] G. Henkelman and H. Jónsson. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.*, 113:9978–9985, 2000.
- [32] G. Henkelman, B. Uberuaga, and H. Jónsson. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *J. Chem. Phys.*, 113:9901–9904, 2000.
- [33] Konrad Hinsen. The molecular modeling toolkit: A new approach to molecular simulations. *Journal of Computational Chemistry*, 21:79–85, 2000.
- [34] J. P. Hirth and J. Lothe. *Theory of Dislocations*. Krieger Publishing Company, second edition, 1982.
- [35] B. L. Holian, A. J. DeGroot, W. G. Hoover, and C.G. Hoover. Time-reversible equilibrium and nonequilibrium isothermal-isobaric simulations with centered-difference Stoermer algorithms. *Physical Review A*, 41:4552–4553, 1990.
- [36] B. L. Holian, A. F. Voter, N. J. Wagner, R. J. Ravelo, S. P. Chen, W. G. Hoover, C. G. Hoover, J. E. Hammerberg, and T. D. Dontje. Effects of

- pairwise versus many-body forces on high-stress plastic deformation. *Physical Review A*, 43:2655, 1991.
- [37] D. Holland and M. Marder. Ideal brittle fracture of silicon studied with molecular dynamics. *Phys. Rev. Lett.*, 80:746, 1998.
- [38] D. Holland and M. Marder. Erratum: Ideal brittle fracture of silicon studied with molecular dynamics [phys. rev. lett. 80, 746 (1998)]. *Phys. Rev. Lett.*, 81:4029, 1999.
- [39] H. B. Huntington, J. E. Dickey, and R. Thompson. Dislocation energies in NaCl. *Phys. Rev.*, 100:1117–1128, 1955.
- [40] Erin Iesulauro, 2002. Private communication.
- [41] K. W. Jacobsen and J.J. Mortensen, 2002. Private communication.
- [42] C. B. Jiang, S. Patu, Q. Z. Lei, and C. X. Shi. Dislocation velocities in Ni₃Al single crystals. *Philosophical Magazine Letters*, 78:1–8, 1998.
- [43] H. Jónsson, G. Mills, and K. W. Jacobsen. Nudged elastic band method for finding minimum energy paths of transitions. In B. J. Berne, G. Ciccotti, and D. F. Coker, editors, *Classical and Quantum Dynamics in Condensed Phase Simulations*. World Scientific, 1998.
- [44] J. F. Justo, M. Z. Bazant, E. Kaxiras, V. V. Bulatov, and S. Yip. Interatomic potential for silicon defects and disordered phases. *Phys. Rev. B*, 58:2539, 1998.
- [45] C. L. Kelchner, S. J. Plimpton, and J. C. Hamilton. Dislocation nucleation

- and defect structure during surface indentation. *Physical Review B*, 58:11085–11088, 1998.
- [46] L. Kleinman. Deformation Potentials in Silicon. I. Uniaxial Strain. *Physical Review*, 128:2614, 1962.
- [47] P. E. W. Labossiere and M. L. Dunn. Calculation of stress intensities at sharp notches in anisotropic media. *Eng. Fract. Mech.*, 61:635–654, 1998.
- [48] C.R. Myers, R. Loge, C.-S. Chen, and P.R. Dawson. unpublished.
- [49] NetCDF home page. <http://www.unidata.ucar.edu/packages/netcdf>.
- [50] K. Ohsawa and E. Kuramoto. Flexible boundary condition for a moving dislocation. *J. Appl. Phys.*, 86:179–185, 1993.
- [51] M. Ortiz and R. Phillips. Nanomechanics of defects in solids. *Advances in Applied Mechanics*, 35:1–79, 1999.
- [52] R. Pérez and P. Gumbsch. Directional anisotropy in the cleavage fracture of silicon. *Phys. Rev. Lett.*, 84:5347, 2000.
- [53] Python home page.
- [54] S. Rao, C. Hernandez, J. P. Simmons, T. A. Parthasarathy, and C. Woodward. Green’s function boundary conditions in two-dimensional and three-dimensional atomistic simulations of dislocations. *Philosophical Magazine A*, 77:231–256, 1998.
- [55] RasMol home page. <http://www.umass.edu/microbio/rasmol>.

- [56] K. W. Schwarz. Interaction of dislocations on crossed glide planes in a strained epitaxial layer. *Phys. Rev. Lett.*, 78:4785–4788, 1997.
- [57] J. P. Sethna et al. LASSPTools: Graphical and Numerical Extensions to Unix, 1990.
- [58] G. C. Sih, P. C. Paris, and G. R. Irwin. On cracks in rectilinearly anisotropic bodies. *Int. J. Fract. Mech.*, 1:189–202, 1965.
- [59] J. E. Sinclair. Improved atomistic model of a bcc dislocation core. *J. Appl. Phys.*, 42:5321–5329, 1971.
- [60] J. E. Sinclair, P. C. Gehlen, R. G. Hoagland, and J. P. Hirth. Flexible boundary conditions and nonlinear geometric effects in atomic dislocation modeling. *J. Appl. Phys.*, 49:3890–3897, 1978.
- [61] M. R. Sorensen and A. F. Voter. Temperature-accelerated dynamics for simulation of infrequent events. *J. Chem. Phys.*, 112:9599, 2000.
- [62] F. H. Stillinger and T. A. Weber. Computer simulation of local order in condensed phases of silicon. *Phys. Rev. B*, 31:5262, 1985.
- [63] S. H. Strogatz. *Nonlinear Dynamics and Chaos*. Addison-Wesley, 1994.
- [64] A. H. Stroud. *Approximate Calculation of Multiple Integrals*. Prentice Hall, 1971.
- [65] W. Suwito, M. L. Dunn, and S. J. Cunningham. In *Transducers 97, Proceedings of the 1997 International Conference on Solid-State Sensors and Actuators*, page 611, 1997.

- [66] W. Suwito, M. L. Dunn, and S. J. Cunningham. Fracture initiation at sharp notches in single crystal silicon. *J. Appl. Phys.*, 83:3574, 1998.
- [67] W. Suwito, M. L. Dunn, S. J. Cunningham, and D. T. Read. Elastic moduli, strength, and fracture initiation at sharp notches in etched single crystal silicon microstructures. *J. Appl. Phys.*, 85:3519, 1999.
- [68] Swig home page. <http://www.swig.org>.
- [69] E. B. Tadmor, M. Ortiz, and R. Phillips. Quasicontinuum analysis of defects in solids. *Philosophical Magazine A*, 73:1529–1563, 1996.
- [70] E. B. Tadmor, R. Phillips, and M. Ortiz. Mixed atomistic and continuum models of deformation in solids. *Langmuir*, 12:4529–4534, 1996.
- [71] E. B. Tadmor, G. S. Smith, N. Bernstein, and E. Kaxiras. Mixed finite element and atomistic formulation for complex crystals. *Phys. Rev. B*, 59:235, 1999.
- [72] Cristian Teodosiu. *Elastic Models of Crystal Defects*. Springer-Verlag, 1982.
- [73] T. C. T. Ting. *Anisotropic Elasticity: Theory and Applications*. Oxford University Press, New York, 1996.
- [74] T. L. Veldhuizen. Arrays in Blitz++. In *Proceedings of the 2nd International Scientific Computing in Object Oriented Parallel Environments (ISCOPE'98)*, 1998.
- [75] L. Wickham, K. W. Schwarz, and J. S. Stölken. Rules for forest interactions between dislocations. *Phys. Rev. Lett.*, 83:4574–4577, 1999.
- [76] Zhiliang Zhang. A failure criterion for mode I fracture initiation of brittle solids with sharp notches. *To be published*, 2002.