INL/EXT-13-29406

Exploration of the Phase Field Framework MARMOT to Include Anisotropic Grain Boundaries with Molecular Dynamics

Aaron S. Butterfield

July 2013

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-13-29406

Exploration of the Phase Field Framework MARMOT to Include Anisotropic Grain Boundaries with Molecular Dynamics

Aaron S. Butterfield

July 2013

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

EXPLORATION OF THE PHASE FIELD FRAMEWORK MARMOT TO INCLUDE ANISOTROPIC GRAIN BOUNDARIES WITH MOLECULAR DYNAMICS

by

Aaron S. Butterfield

A senior thesis submitted to the faculty of

Brigham Young University - Idaho

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics

Brigham Young University - Idaho

July 2013

ABSTRACT

EXPLORATION OF THE PHASE FIELD FRAMEWORK MARMOT TO INCLUDE ANISOTROPIC GRAIN BOUNDARIES WITH MOLECULAR DYNAMICS

Aaron S. Butterfield

Department of Physics

Bachelor of Science

Using molecular dynamic (MD) calculations, meso-scale phase field simulations behave differently by including anisotropic grain boundary energy in the model for transient energy minimization in uranium-dioxide. As a preliminary step, MD will be used to find face centered cubic (FCC) copper grain boundary energies and to explore its anisotropic behavior in phase field simulations. MARMOT is a C++ code that uses object-oriented programming to implement the Allen-Cahn and Cahn-Hilliard equations for phase field modeling. MARMOT uses constants for mobility and grain boundary energy for the entire domain of the simulation. For MARMOT to reach its full capability, these constants need to vary within the domain. This paper explores the expansion of MARMOT to include an anisotropic view of grain boundary energy in transient phase field simulations.

Contents

List of Figures

Chapter 1

Introduction

1.1 Nuclear Power in the Future

The United States led the way in the early development of nuclear energy. As the DOE looks toward the future, it is apparent that nuclear energy has a significant place in the United States' energy security; however, today's world is a much different place than in the mid 1900's. After the terrorist attacks in New York City and Washington DC, nuclear non-proliferation is of paramount concern. The recent accident in Fukoshima, Japan and past accidents at Three Mile Island and Chernobyl create questions about the safety of existing nuclear reactors. Amid these challenges, the world has a growing need for energy and it is only going to increase in the coming years. "Domestic demand for electrical energy is expected to grow by more than 20% from 2011 to 2040."[1] As a result, the Department of Energy (DOE) is very interested in cost-effective, environmentally friendly, safe, and sustainable energy sources, particularly nuclear energy.

1.2 Light Water Reactor Sustainability

The DOE is taking a two-pronged approach to nuclear research. They are working on maintaining and revitalizing their existing fleet of Light Water Reactors (LWRs) and designing next-generation nuclear reactors. Many of the existing reactors contain outdated sensors and control systems that need to be replaced to make them safer and more automated. Due to the harsh conditions inside a reactor, the state of the components (walls, cladding, coolant pipes, etc.) deteriorates and the reactor becomes costly to re-certify. In an effort to maintain and revitalize the existing LWRs, the DOE is funding research for computational models that are accurate enough to target the deteriorating components to replace for re-certification and to find innovative ways to make the reactors more efficient. "The Light Water Reactor Sustainability Program is developing the scientific basis to extend existing nuclear power plant operating life beyond the current 60-year licensing period and ensure long-term reliability, productivity, safety, and security."[1] This will allow the current LWRs to operate while research is being done to develop the next generation reactors. [1]

1.3 Modeling in the Future

Computational modeling maximizes the important results gained from experiments. Computational modeling and experimental methods have their strengths and weaknesses, but if used together, their weaknesses are minimized and their strengths are maximized. Building large scale experiments are more expensive than using computational methods; however, computational methods are only as correct as their model. As a result, computational modeling can be used to guide costly experiments to look for specific data that would be of the most worth. Experiments performed in test

Figure 1.1 Uranium Oxide Fuel Pellets

reactors can take years to run. This extended time scale causes the data from the experiment to be gathered at a trickle compared to running simulations in a cluster or supercomputer. The data gathering is further prolonged if other preliminary experiments need to be preformed. Computational modeling has the ability to run some of the preliminary experiments numerically first. Often, scientists will use a "bake and check" method, where a material left in a reactor is cut open and the grain structure is analyzed to make inferences about what occurred on the meso-scale during its time in the reactor. These inferences can be supported by computational simulations that check if our understanding is correct.

The current models used for nuclear engineering for LWR design need to be adapted for next generation designs. Computational models will simulate next generation reactors to anticipate specific design adaptations. For example, the High Temperature Gas-Cooled Reactor (HTGR) uses a compressible fluid instead of an incompressible fluid as its coolant. "HTGRs are intrinsically different from conventional. . . LWRs, which leads to issues in adapting traditional numerical methodologies. For example, a large temperature difference between the reactor inlet and outlet creates a significant density variation in the helium coolant. In this case, employing

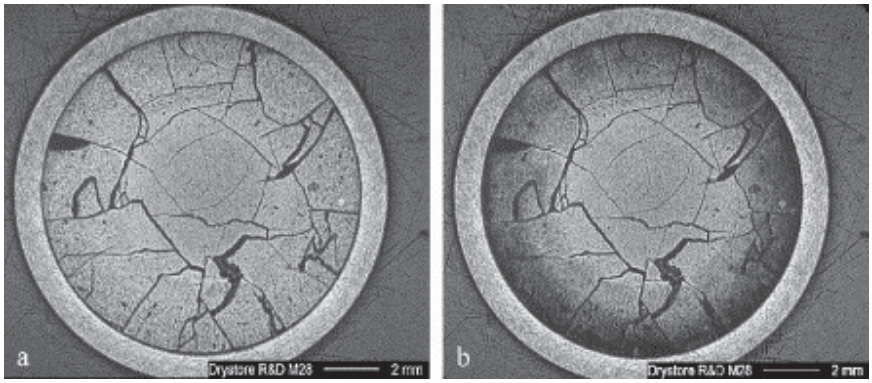


Figure 1.2 Uranium Oxide Fuel Cracking and Fatigue

an incompressible assumption and using the Boussinesq approximation for buoyancy may not give accurate heat transfer results."[2] Currently there are no next generation reactors, so computational methods will be heavily relied on as they are built.

As the cost of computing has decreased over the years, it has allowed computational research, particularly in LWRs, to reach a new level for existing technologies. "Modeling and simulation has a long history with researchers and scientists exploring nuclear energy technologies. In fact, the existing fleet of currently operating reactors was licensed with computational tools that were produced or initiated in the 1970s. Researchers and scientists in Nuclear Energy Advanced Modeling and Simulation (NEAMS) are developing new tools to predict the performance, reliability and economics of advanced nuclear power plants. The new computational tools will allow researchers to explore in ways never before practical, at the level of detail dictated by the governing phenomena, all the way from important changes in the materials of a nuclear fuel pellet to the full-scale operation of a complete nuclear power plant."[1]

As nuclear research moves forward, both computational modeling and experimental methods will be vital in engineering and researching current and future designs.

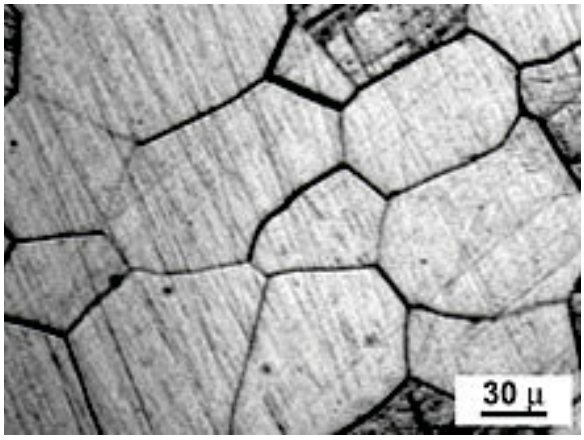


Figure 1.3 Grain Structure of Material

1.4 Grain Boundaries in Material Science

Grain sizes in materials affect the overall properties of the material. Often engineers will assume that the properties in a material is the same in every direction (isotropic), however, the properties of many materials vary based on the direction or orientation (anisotropic).

Because of the extreme conditions inside a nuclear reactor, understanding grain structure at the meso-scale is imperative. For example, temperature gradients across fuel pellets can change over a one-thousand degrees kelvin in a centimeter. Grain sizes greatly affect the thermal conductivity and thereby become an important attribute to understand for efficient and safe coolant flows. Not only have these effects been seen in models, but they have now been verified experimentally by looking at the spent cladding of nuclear reactors.

Chapter 2

Theory and Methods

The use of computational methods allows researchers to know how a reactor will behave from an engineering perspective. At Idaho National Laboratories (INL), computational modeling is leading the way for experiential research. They have developed two applications called MOOSE (Multi-Physics Object Oriented Simulation Environment) and MARMOT. MOOSE is an object oriented code that employs the finite element method (FEM) for nuclear research and design. MARMOT is an extension of MOOSE that allows for simulation of phase field models. In the current models that MOOSE and MARMOT employ, MARMOT assumes that the grain boundary energy across each grain boundary is only a factor of its area and not its orientation, relative to the adjacent grain. In reality, materials have several different kinds of grain boundaries; each differs in its energy per area due to its grain orientation with the adjacent grain. Changing the framework to include these cases will allow a more realistic view of materials at the meso-scale.[3]

Understanding how fuel pellets behave under the extreme conditions inside a nuclear reactor is central to the designing of safer, economical, and environmentally friendly reactors. Uranium dioxide fuel pellets power most of the reactors today. I

have chosen to use FCC copper for preliminary research to validate an anisotropic model that can be employed in the future to analyze the grain structure of uranium dioxide. There is a vast amount of data about FCC copper available in the research community to validate my findings and its simple lattice structure makes the molecular dynamics easier to implement.

2.1 Theory

In my research, I will be using three areas of computational science to build an anisotropic model of FCC copper:

1. Molecular Dynamics
2. The Finite Element Method
3. Phase Field Method

2.1.1 Molecular Dynamics

Molecular dynamics (MD) uses Newton's second law to calculate the positions and velocities of a system of particles. Given the initial conditions of the system, many of its thermodynamic properties can be obtained. MD can be used for any multi-body system; however, it tends to be used for the study the motion of molecules, hence, the name "molecular dynamics."[4]

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is an MD software package produced by Sandia National Laboratories. It uses the embedded atom method (EAM) to calculate position, acceleration, pressure, and energy. The EAM is represented by equation 2.1,

$$E_i = F_\alpha \left(\sum_{i \neq j} \rho_\beta(r_{ij}) \right) + \frac{1}{2} \sum_{i \neq j} \phi_{\alpha\beta}(r_{ij}), \tag{2.1}$$

where r_{ij} represents the distance between atoms i and j. ρ_β represents the contribution of atom j, of type β, on the electron gas cloud. F_α is the embedded energy function representing the amount of energy necessary to embed an atom i, of type α, within the electron gas cloud. ϕ_{ij} is a function that represents the pairwise interaction of the atoms i and j.

Because MD simulations calculate energy and forces of a given system, energy minimization through various methods is possible. LAMMPS has an energy minimization operating mode which translates atoms randomly to find a minimum energy state within the configuration. LAMMPS also allows you to minimize the system while holding some thermodynamics properties constant, such as volume, entropy, enthalpy, etc. The lowest energy can be achieved by using a series of minimizations with different thermodynamic properties.

Grain boundary energies are found by subtracting two minimized system energies. A monocrystalline material represents the lowest energy state possible. LAMMPS keeps a running total of the energy of the system as it minimizes. After obtaining the result for the monocrystalline (E_{MONO}) and polycrystalline (E_{POLY}) materials, the grain boundary energy, (E_{GB}), can be achieved by finding the difference between the two systems energies and dividing the energy by twice the area (A) of the grain boundary.

$$E_{GB} = \frac{E_{POLY} - E_{MONO}}{2A} \tag{2.2}$$

2.1.2 Finite Element Method

The Finite Element Method (FEM) is "a numerical analysis technique for obtaining approximate solutions to a wide variety of engineering problems...the basic premise of the finite element method is that a solution region can be analytically modeled or approximated by replacing it with an assemblage of discrete elements."[5]

Most people get exposed to the fundamentals of FEM in their first geometry class. You see a complex shape when you look at the following shape in figure 2.1. When asked to find the area of the shape, almost without thinking, you separate or discretize the domain into smaller pieces to simplify the problem. In FEM we separate a complex shape into smaller, more manageable, elements that the computer can easily handle.

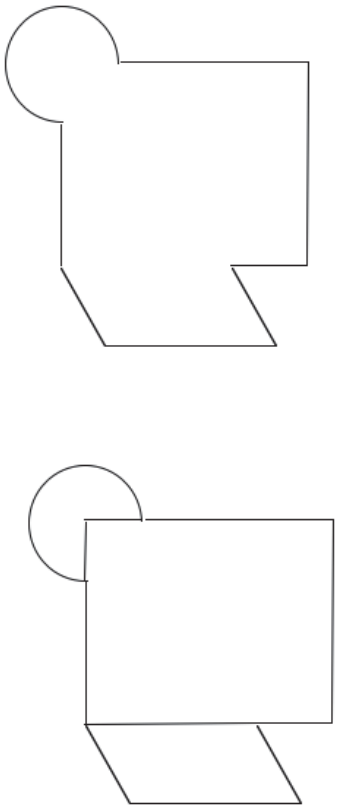


Figure 2.1 Simplifying Complex Geometric Shapes

Several numerical analysis methods have been developed over the years. One common model is the finite difference method; it uses a grid point system to discretize its domain, which means that information is only available for those points and the rest of your domain is left without data. This method is difficult to use for irregular geometries and unusual boundary conditions. A benefit of using FEM is that it is meant for complex shapes and you can have continuous data over your whole domain. [5]

In nuclear engineering, a continuous domain is important to maintain while looking at very complex shapes. For example, a nuclear reactor has several fuel rods that are

bathed in a pool of water. The fuel rods of a reactor also have control rods that slide over the them to control the nuclear reaction that generates the heat. The location of the control rods on the fuel rods greatly influence the performance of the reactor. Modeling the reactor using complex geometry is essential to understanding how quickly the water can remove the heat from the reactor.

2.1.3 Phase Field Modeling

The phase field method uses the FEM to analyze the behavior of interfaces between different materials (eg. oil and water) or phases of the same material (eg. ice and water). The model uses the Allen-Cahn and Cahn-Hilliard equations to track interfaces between different phases or materials as they evolve over time. Within the phase field framework, each phase or material is represented by "continuous variables that smoothly transition from one value to another...this description of the interfaces eliminates the need to explicitly discretize a[n]...interface." [3] These continuous variables represent the existence of a phase or material at a given point.

A one-dimensional example is the step function in mathematics. Consider the function $f(x) = x^2$; in order to make the function to exist between the values of $2 < x < 3$, we can utilize a step function. The step function is defined as 2.3

$$u(x) = \begin{cases} 0 & x < 0 \\ & \\ 1 & x \geq 0 \end{cases}, \tag{2.3}$$

using this function we can represent our variable as equation 2.4,

$$\phi(x) = f(x) * (u(x-2) - u(x-3)), \tag{2.4}$$

where ϕ is the order parameter. The order parameter is defined over the entire domain, but, $f(x)$ is only between $2 < x < 3$. In the phase field approach, micro-structural

features are described using continuous variables. These variables take two forms: conserved variables representing physical properties such as atom concentration or material density, and non-conserved order parameters describing the microstructure of the material, including grains and different phases.[3]

The evolution of these order parameters is a function of the derivative of the Gibbs free energy, which in turn is a function of the order parameters. The time evolution of the conserved order parameters proceeds according to the Cahn-Hilliard partial differential equation (PDE) 2.5,

$$\frac{\partial c_i}{\partial t} = \nabla \cdot M_i \nabla \frac{\partial F}{\partial c_i}, \tag{2.5}$$

where c_i is the conserved order parameter, M_i is the associated mobility, and F is the free energy functional. The time evolution of the non-conserved order parameters proceed according to the Allen-Cahn PDE 2.6,

$$\frac{\partial \eta_j}{\partial t} = -L_j \frac{\partial F}{\partial \eta_j}, \tag{2.6}$$

where η_i is the non-conserved order parameter and L_j is the non-conserved order parameter mobility. For a system using N conserved order parameters and M non-conserved order parameters, the free energy functional follows the form

$$F = \int_V [f_{loc}(c_i, \ldots, c_N, \eta_i, \ldots, \eta_M) + f_{gr}(c_i, \ldots, c_N, \eta_i, \ldots, \eta_M) + E_d], \tag{2.7}$$

where f_{loc} is the local free energy and f_{gr} is the gradient free energy, and E_d is any long distance energy provided by electromagnetic or pressure applied from the outside of the material.

For the cases of grain boundary motion equation 2.8 represents the f_{loc},

$$f_{loc} = \mu(\eta_i^3 - \eta_i + 2\sum_{j \neq 1}^{N} \eta_i \eta_j^2), \tag{2.8}$$

where μ represents a model parameter that is related to the grain boundary surface energy. The f_{gr} is defined as

$$f_{gr} = \sum_{i}^{N} \kappa_i/2|\nabla c_i|^2 + \sum_{i}^{M} \kappa_j/2|\nabla \eta_j|^2, \tag{2.9}$$

for all of the conserved and non-conserved order parameters. In MARMOT, the phase field model assumes that μ,κ_i, κ_j, M_i, and L_j are constants. This is what we will call the isotropic case. In reality, grain boundaries have different energies based on their orientation with adjacent grains. This is the called the anisotropic case. My anisotropic model continues to treat the order parameter mobilities (M_i and L_j) as a constants, but will allow μ, κ_i, and κ_j to vary throughout the domain.

2.2 Methods

Now that the numerical method's theory has been explained, we need to understand how it is applied. I will outline the MD procedure for calculating grain boundary energies and how that data will be used to implement anisotropic behavior in MARMOT.

2.2.1 Molecular Dynamics Method

In the LAMMPS input file I set the units in which I wanted the simulation to be performed. In this case, the unit's setting name is called "metal." The distance will be in angstroms, mass is in amu, time is in picoseconds, energy is in eV, etc. The lattice is set to a unit length of 3.61070$\mathring{A}$ with an orientation of x along [100]. I created a simulation box of 50 x 50 x 50 $\mathring{A}$ and split it into two regions, one for the top grain and one for the bottom. In the monocrystalline structure, the top and bottom regions have the same lattice. The monocrystalline structure is the minimum

energy state in which all the other simulations will be compared. Then I imported the EAM file which contains the potential for copper and specified a cut-off distance for the potential. The temperature for all the simulations was set to $0K$. Then I minimized the system twice, once fixing the volume as a constant, and once, allowing the volume to change in the z direction. After each minimization, I calculated the grain boundary energy from equation 2.2.

After calculating the minimum energy for the monocrystalline structure, I found the minimum energy for several orientations of polycrystalline structures. My orientation of the bottom grain was based on ϕ and θ of a spherical coordinate system. In order to use angles from a spherical coordinate system, I had to define a FCC copper unit cell in a custom way. I used a python script to loop through ϕ and θ from 0 to $\pi/2$ to batch-process each MD simulation.

2.2.2 Anisotropic Model in MOOSE and MARMOT

MOOSE is a compilation of other powerful software which provides rich framework to build our new anisotropic model by modularizing its code into classes. The modules and pieces of MOOSE are listed on figure 2.2.

In order to understand the new anisotropic model of grain boundaries, it will be important to understand four of the classes or modules in MOOSE and what they do. The four classes are Kernels, Materials, Boundary Conditions, and Initial Conditions.

Kernels

Kernels describe the physics in the simulation and they contain PDEs that have been reduced to their weak form. Higher-order equations are reduced by using the divergence theorem. Because of MOOSE's object oriented roots, scientists and engineers can focus on the PDEs and not on monotonous programing syntax. Instead,

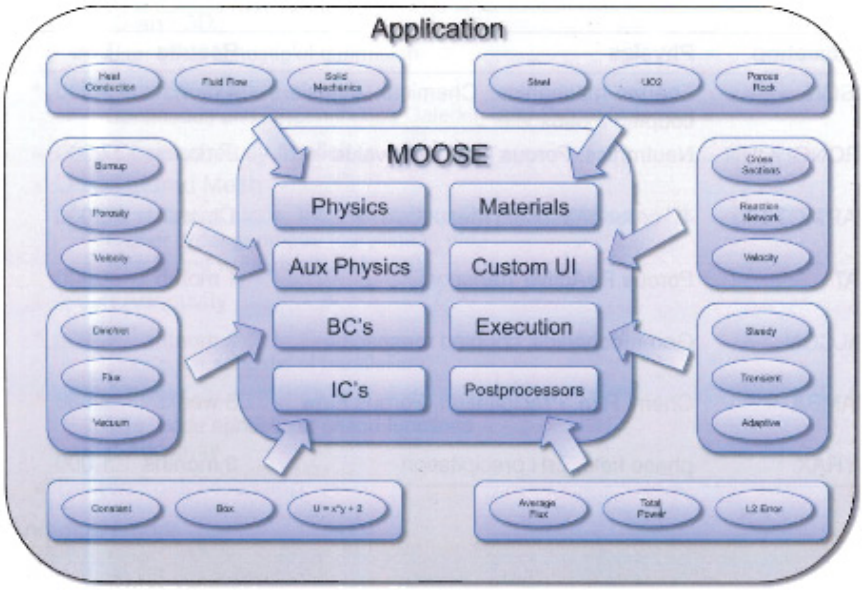


Figure 2.2 MOOSE Block Diagram

the PDEs can almost be added to the simulation exactly in its weak form. In the anisotropic model, the kernel that we will use is found in equation below. As we apply the divergence theorem and reduce the equation to its weak form, it looks like this:

$$(\frac{\partial \eta_j}{\partial t}, \phi_m) = -L(\kappa_j \nabla \eta_j, \nabla \phi_m) - L(\frac{\partial f_{loc}}{\partial \eta} + \frac{\partial E_d}{\partial \eta}, \nabla \phi_m) + L < \kappa_j \nabla \eta_j \cdot \vec{n}, \nabla \phi_m > . \quad (2.10)$$

The last term of the equation represents our boundary condition and the other parts represent the differential equation that includes the physics in our domain. MARMOT uses equation 2.10 as superclass that all other subclasses are based upon. MOOSE is open to any PDE under the sun; however, MARMOT limits itself to just the phase field perspective. As a result, the only functions that need to be provided are $\frac{\partial f_{loc}}{\partial \eta}$ and $\frac{\partial E_d}{\partial \eta}$. MARMOT simplifies the use of MOOSE for phase field simulations using the Allen-Cahn equation. For the purpose of grain boundaries, $\frac{\partial f_{loc}}{\partial \eta}$ is the derivative of equation 2.8.

Materials

Notice in the equations 2.5, 2.6, 2.8, and 2.9, there are several coefficients which are not provided in the kernel. These coefficients are calculated at each quadrature point. In FEM, an integral is taken to drive the residual to zero. Each node of the elements carries the value of the order parameter η. A test function is used to multiply by and an integral is taken to perform a best fit to the domain. MOOSE uses Newtonian Quadrature to integrate over the domain. As a result, the error on integration is zero due to every test function being a polynomial.

At every quadrature point throughout the domain, the kernel class calls the material class. The material class calls the computeQpProperties() function, which contains whatever code the user would like to run to provide the coefficients at each quadrature point.

Boundary and Initial Conditions

Boundary conditions (BCs) and initial conditions (ICs) are very intuitive; however, it is important to understand what MOOSE has built into it. MOOSE uses Dirichlet, Neumann, and periodic BCs. For the most part in our simulations at the meso-scale, we use periodic BCs which are already included and do not have to be programed.

ICs are set by overloading a function called value() in the IC‘s superclass. The input to the function is a point data type which represents arbitrary points in the domain. Using conditional statements, an individual provides the initial conditions for the transient simulation.

The Model

The new model exists as a material class in the MOOSE framework, and is called AnisotropicGrainBoundaryEnergy. This class will inherit all of the properties of the

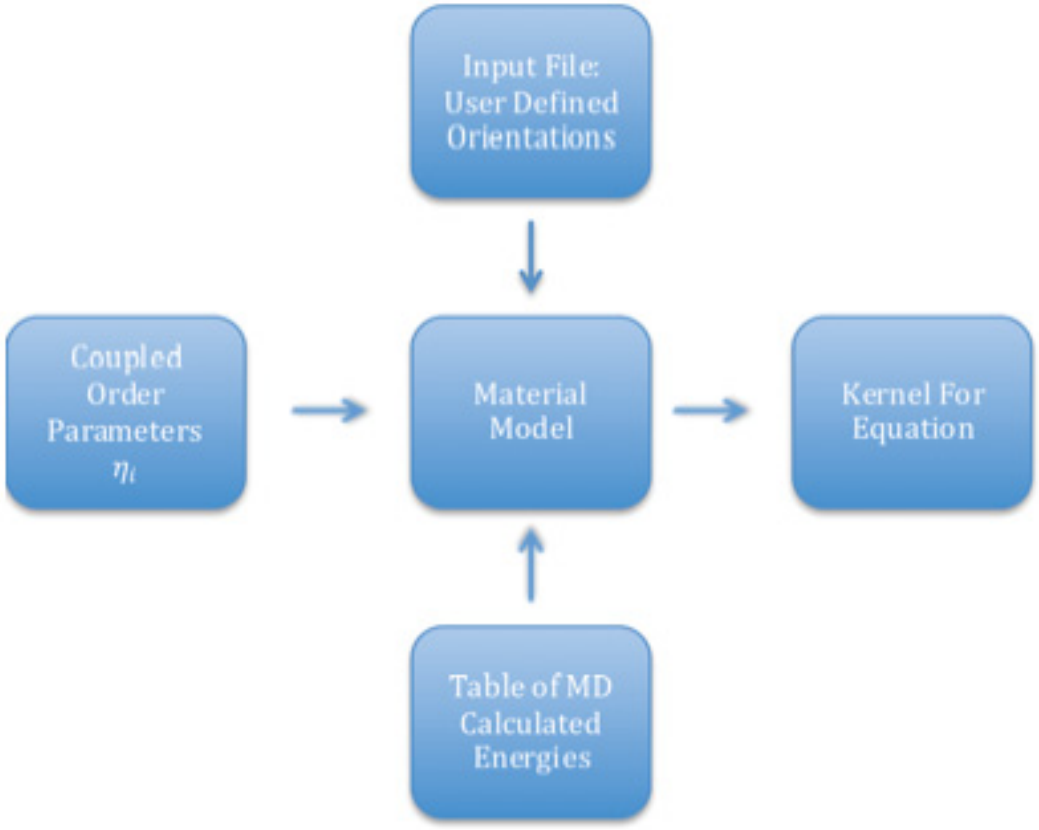


Figure 2.3 Anisotropic Model for MARMOT

material class. It then overloads the computeQpProperties() function. As the object of the class is created at the beginning of the simulation, the MD database of grain boundary energies is loaded into a multidimensional array for easy reference by the constructor of the class. Further, the constructor will couple each of the order parameters into the simulation to reference where the quadrature point is within the domain and how close the point exists to each of the grains. The model will allow the user to define the orientation of the grains in the simulation or let the user assign a random distribution to each grain.

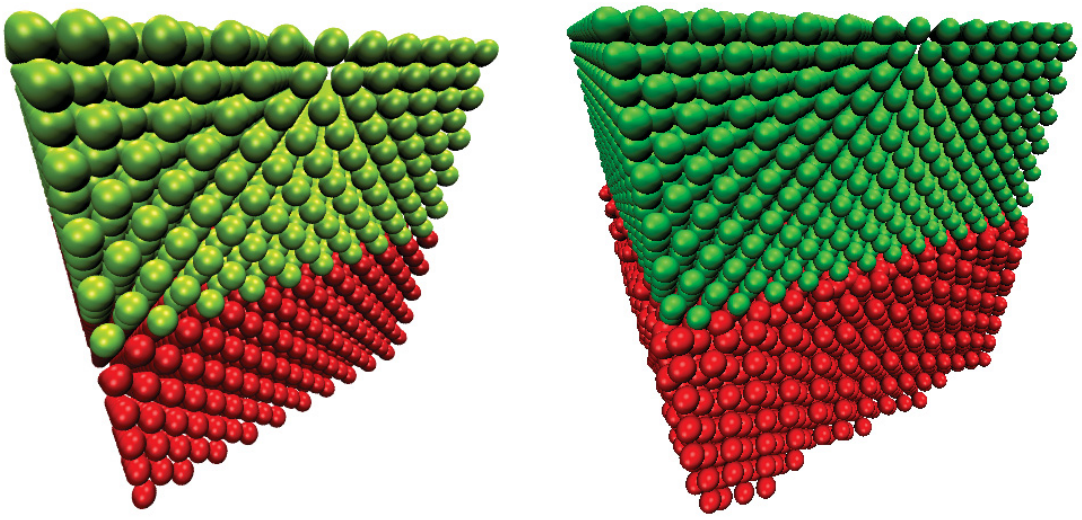
As the simulation starts, quadrature points will be evaluated. As the computeQpProperties function is called, the function checks the quadrature points location to the nearest grains and uses one of the grains as a reference orientation. Then the function calculates the difference between the two orientations according to ϕ and θ. Referencing the container of energies, it finds the closest energy to the calculated ϕ and θ.

Chapter 3

Results

3.1 Grain Boundary Energies

Minimizing the MD lattices took a large portion of time. The simulations used over 4,500 CPU-hours. I used a program to create many folders for each simulation that needed to be run. I started the run, saving all of the trajectory information of the atoms in each simulation. Once it filled up over 800 GB of data, I had to stop collecting that data. After running the simulation, I compared the minimized single crystalline lattice to the other polycrystalline lattices and found that the simulation allowed other lattices to minimize lower in energy than what should be the base energy. I believe this happened because I did not delete as many atoms as I should have, allowing some atoms to be too close together. Also, the atoms were placed in each region separately which resulted in the lattice restarting its pattern rather than just filling the whole simulation box with one lattice command. This would have eliminated the need for deleting atoms. To correct this, I found the lowest energy of the calculated grains and made that the zero reference point, and the grain boundary energies were plotted according to ϕ and θ. The results of the grain boundary energies



(a) Monocrystalline Lattice **(b) Polycrystilline Lattice**

Figure 3.1 Results of Two Molecular Dynamics Simulations

are located in figure 3.2. The energies followed a general trend, increasing in energy as the angle increased further away from a monocrystalline structure. The graph shows grain boundary energies as a function of ϕ and θ. The grain boundaries form an interpolated surface and the grain boundary energies follow a general trend with a few exceptions of spikes in the data.

3.2 Verifying the Data

Though numerical modeling is an estimate of the real world, we must get it as close as we can to the real values for the model to be accurate. MARMOT has used $0.708 J/m^2$ for the grain boundary energy everywhere, as stated in a paper by Schonfelder[6]. It is the value of a 30 degree twist boundary. I compared my 30 degree twist boundary energy with his to see that mine was and found that both were very comparable.

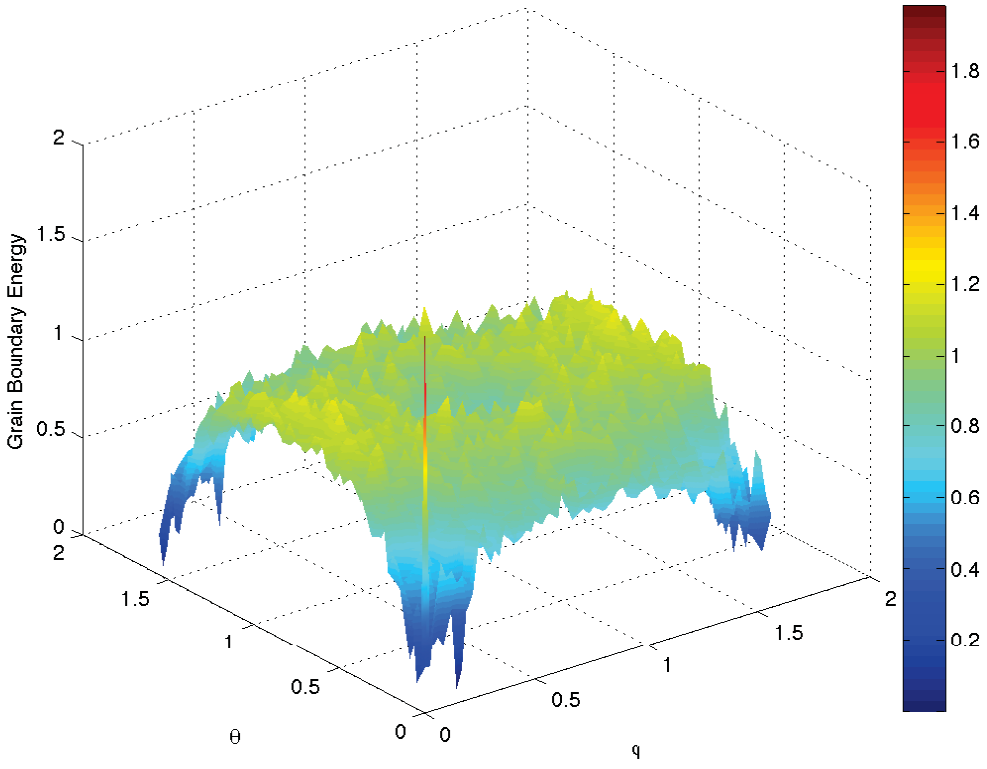


Figure 3.2 Grain Boundary Energies by Orientation

3.3 Using the Data in the Phase Field Simulation

Once I obtained the energies for different orientations, it was time to apply them to the anisotropic model. Using the code that can be found in Appendix A, I used the grain boundary energies to calculate the constants μ and κ in the material model. Running the simulation I was able to watch κ, and κ and μ were doing exactly what I wanted. They were being defined on the grain boundaries and were zero everywhere else. It was fine that it was zero everywhere else; however, I could not get the simulation to converge. This showed that μ and κ could not still be treated as constants and had to be defined as functions of the order parameters. This added a larger complexity to the problem many were hoping was negligible. As you can see by the figure, my material property worked well for the assumption that I was trying to make. In the figure it shows a pixilated nature for κ. This is not an accurate representation of the data. Instead, they were actually defined at each quadrature point; but, the post-processor in elemental so it averages the values of the quadrature points and reports one value for each element.

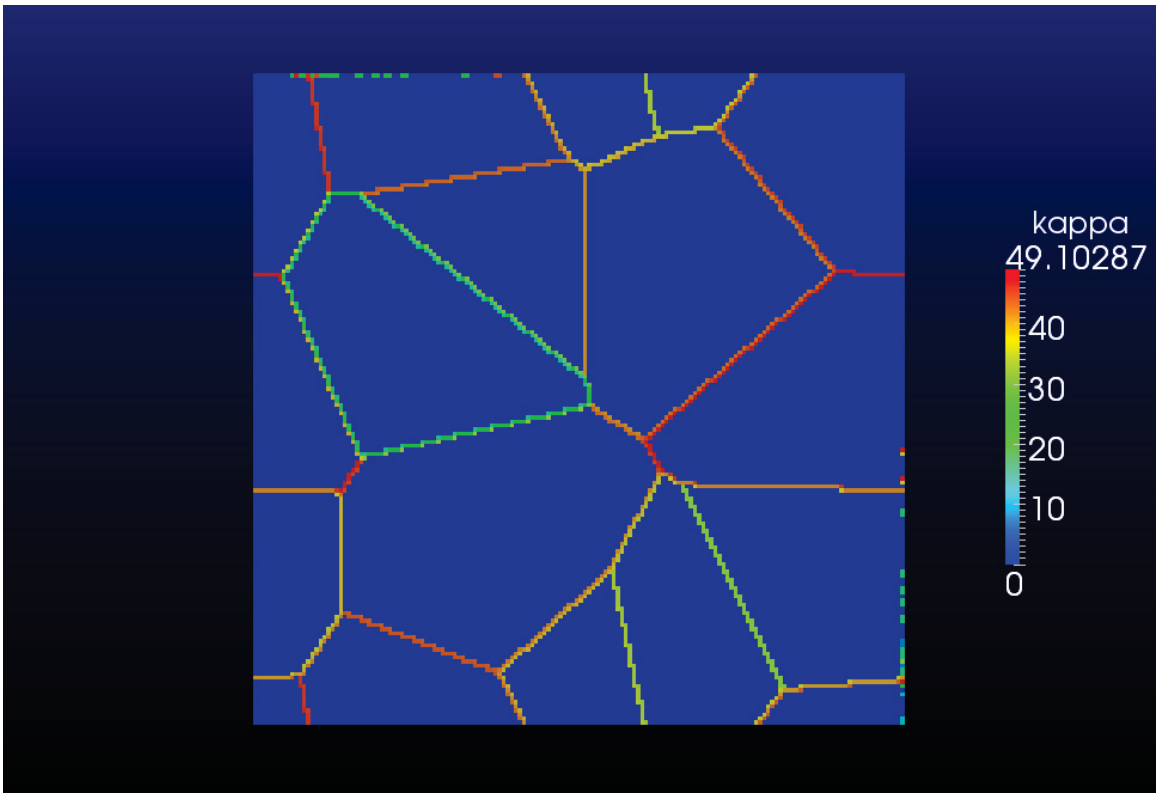


Figure 3.3 Post Processor View of Kappa Across the Domain

Chapter 4

Summary and Conclusions

There are necessary stepping-stones to accomplish goals in research. In setting out to find a new model for Anisotropic Grain Growth within the MARMOT framework, research leads to the right answer. My research showed what valid assumptions could be made and provided necessary data to successfully model FCC copper with anisotropic grain boundaries. I proved that μ and κ need to be represented as some function of order parameters. Then using the chain rule, the material model will again be expanded to include the anisotropic cases.

4.1 Validate Values Obtained from MD

I have calculated over 4,000 grain boundary energies. Each energy represents some twist, tilt or combination of the two. I was able to find one grain boundary energy that was calculated for a 30 degree twist boundary; however, that does not mean I am right. Other MD work should be done to verify. Also, experimental validation is required to verify that my values are correct. Some of the data has unresolved spikes that seem out of the ordinary. These spikes need to be resolved in order for the model

to work correctly. In my anisotropic model, I threw them out because I feel it was a model artifact from setting my deletion radius too short.

Bibliography

[1] Department of Energy. Light water reactor sustainability program integrated program plan. Technical Report INL/EXT-11-23452, 2013.

[2] Derek Gaston, Chris Newman, Glen Hansen, and Damien Lebrun-Grandi. Moose: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design*, 239(10):1768–1778, 10 2009.

[3] Michael R. Tonks, Derek Gaston, Paul C. Millett, David Andrs, and Paul Talbot. An object-oriented finite element framework for multiphysics phase field simulations. *Computational Materials Science*, 51(1):20–29, 1 2012.

[4] Nicholas J. Giordano and Hisao Nakanishi. *Molecular Dynamics*, page 270. Computational Physics. Pearson Education, Inc., Upper Saddle River, NJ, 2nd edition, 2006.

[5] Kenneth H. Hubner, Earl A. Thornton, and Ted G. Byrom. *The Finite Element Method for Engineers*. John Wiley & Sons, New York City, New York, third edition, 1995.

[6] B. Schonfelder, D. Wolf, S. R. Philpot, and M. Futkamp. Molecular-dynamics method for the simulation of grain-boundary migration. 5(4):245–245–262, 1997.

[7] Tae Wook Heo, Lei Zhang, Qiang Du, and Long-Qing Chen. Incorporating diffuse-interface nuclei in phase-field simulations. *Scripta Materialia*, 63(1):8–11, 7 2010.

[8] Pratyush Tiwary, Axel van de Walle, Byoungseon Jeon, and Niels Grönbech-Jensen. Interatomic potentials for mixed oxide and advanced nuclear fuels. *Phys.Rev.B*, 83(9):094104, Mar 2011.

[9] A. Leenaers, L. Sannen, S. Van den Berghe, and M. Verwerft. Oxidation of spent uo2 fuel stored in moist environment. *Journal of Nuclear Materials*, 317(23):226–233, 5/1 2003.

[10] N. Moelans, B. Blanpain, and P. Wollants. Quantitative analysis of grain boundary properties in a generalized phase field model for grain growth in anisotropic systems. *Phys. Rev. B*, 78:024113, Jul 2008.

[11] Department of Energy. Energy consumption by primary fuel. Technical Report DOE/EIA-0383ER, 2013.

[12] Crystal planes in silicon miller index angle between planes wafer flat crystallography. id: 1.

Appendix A

Code

A.1 LAMMPS

```
#Define The Properties of the Simultion
units           metal
atom_style      atomic
boundary  p p p
dimension 3

#Create The Atoms of The Lattice Structure
region     abox block -25.00 25.00 -25.00 25.00 -25.00 25.00 units box
create_box  1 abox
region     tgr block INF INF INF INF  0.0 25 units box
lattice    fcc 3.61070147794558 orient x 1 0 0 orient y 0 1 0 orient z 0
   0 1
create_atoms  1 region tgr
region     bgr block INF INF INF INF -25 0.0 units box
lattice    custom 3.61070147794558 &
    a1 %x &
    a2 %y &
```

```
17      a3 %z &
        basis 0.0 0.0 0.99999999 &
19      basis 0.0 0.99999999 0.0 &
        basis 0.99999999 0.0 0.0 &
21      basis 0.5 0.5 0.0 &
        basis 0.0 0.5 0.5 &
23      basis 0.5 0.0 0.5 &
        basis 0.0 0.0 0.0
25  create_atoms  1 region bgr

27  group   gr1 region tgr
    group   gr2 region bgr
29
    #Define the Potential of the System
31  pair_style      eam/alloy
    pair_coeff      * * Cu01.eam.alloy Cu
33  neighbor        1.0 bin
    neigh_modify    every 1 delay 5 check yes
35
    # Delete Overlaping Atoms
37  delete_atoms  overlap 0.35 all all



39


41  # Run Minimization
    reset_timestep 0
43  thermo          10
    thermo_style custom step pe lx ly lz atoms
45  dump 1 gr1 atom 1  %path/gr0.atom
    dump 2 gr2 atom 1  %path/gr1.atom
47  min_style cg
```

```
minimize 1e-15 1e-15 5000 5000
undump 1
undump 2

# Run Minimization 2
reset_timestep 0
thermo          10
thermo_style custom step pe lx ly lz atoms
fix 1 all box/relax z 0 vmax 0.001
min_style cg
minimize 1e-15 1e-15 5000 5000

# Dump Data into Files
dump 1 gr1 atom 1  %path/gr0.atom
dump 2 gr2 atom 1  %path/gr1.atom
```

template.in

A.2 C++

```
#ifndef AnisotropicCuGrGr_H
#define AnisotropicCugrgr_H

#include "Material.h"
#include "AddV.h"
#include "MooseRandom.h"

//Forward Declarations
class AnisotropicCuGrGr;

template<>
```

```
12 InputParameters validParams<AnisotropicCuGrGr>();

14 class AnisotropicCuGrGr : public Material
   {
16 public:
     AnisotropicCuGrGr(const std::string & name,
18           InputParameters parameters);

20 protected:
     virtual void computeProperties();
22   unsigned int binarySearch( std::vector<Real> &array, Real value);
     bool onGrainBoundary(unsigned int qp);
24   void findClosestGrains(unsigned int &closest_grain,unsigned int &
      next_closest_grain, unsigned int qp);
     void initializeGBEnergies();
26
   private:
28   //VariableValue & _cg;

30   Real _temp;
     Real _f0s;
32   Real _wGB;
     Real _length_scale;
34   Real _time_scale;
     Real _GBmob0;
36   Real _Q;
     Real _GBenergy;
38   bool _has_T;
     unsigned int _crys_num;
40
```

```
42  VariableValue * _T; //pointer rather than reference

44  MaterialProperty<Real> & _sigma;
    MaterialProperty<Real> & _M_GB;
46  MaterialProperty<Real> & _kappa;
    MaterialProperty<Real> & _L;
48  MaterialProperty<Real> & _l_GB;
    MaterialProperty<Real> & _mu;
50  MaterialProperty<Real> & _entropy_diff;
    MaterialProperty<Real> & _molar_volume;
52  MaterialProperty<Real> & _act_wGB;

54  Real kb;
    Real pi;
56
    // These Three Vectors Hold the Values From the MD Simulations
58  std::vector<std::vector<Real> > _eng_lookup;
    std::vector<Real> _possible_phi;
60  std::vector<Real> _possible_theta;

62  // These Vectors hold the Orientaion of the grains in the Simulation
    std::vector<Real> _grn_phi;
64  std::vector<Real> _grn_theta;

66  // Differences in orientaion
    Real _theta;
68  Real _phi;

70  std::vector<VariableValue *> _vals;
    unsigned int _ncrys;
72
```

```
};

#endif //AnisotropicCuGrGr_H
```

AnisotropicCuGrGr.h

```
#include "AnisotropicCuGrGr.h"

template<>
InputParameters validParams<AnisotropicCuGrGr>()
{
  InputParameters params = validParams<Material>();

  params.addParam<Real>("temp", 300,"Constant temperature in Kelvin");
  params.addCoupledVar("T", "Temperature in Kelvin");
  params.addParam<Real>("f0s", 0.125,"The GB energy constant ");
  params.addParam<Real>("wGB", 5.0,"Diffuse GB width in nm ");
  params.addParam<Real>("length_scale",1.0e-9,"Length scale in m, where
    default is nm");
  params.addParam<Real>("time_scale", 1.0e-9,"Time scale in s, where
    default is ns");
  params.addParam<Real>("GBmob0", 2.5e-6,"Grain boundary mobility
    prefactor in m^4/(J*s), defaults to the value for copper from
    Schoenfelder1997");
  params.addParam<Real>("Q", 0.23,"Grain boundary migration activation
    energy in eV, defaults to the value for copper from Schoenfelder1997
    ");
  //params.addParam<Real>("GBenergy", 0.708,"Grain boundary energy in J/
    m^2, defaults to the value for copper from Schoenfelder1997");
  params.addCoupledVar("v","Array of coupled variables");
  params.addRequiredParam<unsigned int>("crys_num","number of grains");
```

```
19    params.addRequiredParam<std::string>("var_name_base","base for
        variable names");


21

    return params;
23  }

25  AnisotropicCuGrGr::AnisotropicCuGrGr(const std::string & name,
        InputParameters parameters):
        Material(name, AddV(parameters) ),
27      _temp(getParam<Real>("temp")),
        //_cg(coupledValue("cg")),
29      _f0s(getParam<Real>("f0s")),
        _wGB(getParam<Real>("wGB")),
31      _length_scale(getParam<Real>("length_scale")),
        _time_scale(getParam<Real>("time_scale")),
33      _GBmob0(getParam<Real>("GBmob0")),
        _Q(getParam<Real>("Q")),
35      //_GBenergy(declareProperty<Real>("GBenergy")),
        //_GBenergy(getParam<Real>("GBenergy")),
37      _has_T(isCoupled("T")),
        _crys_num(getParam<unsigned int>("crys_num")),
39      _T(_has_T ? &coupledValue("T") : NULL),
        _sigma(declareProperty<Real>("sigma")),
41      _M_GB(declareProperty<Real>("M_GB")),
        _kappa(declareProperty<Real>("kappa_op")),
43      _L(declareProperty<Real>("L")),
        _l_GB(declareProperty<Real>("l_GB")),
45      _mu(declareProperty<Real>("mu")),
        _entropy_diff(declareProperty<Real>("entropy_diff")),
47      _molar_volume(declareProperty<Real>("molar_volume")),
```

```
    _act_wGB(declareProperty<Real>("act_wGB"))
{
 kb = 8.617343e-5;//Boltzmann constant in eV/K
 pi = 3.14159265359;

 MooseRandom::seed(11025);

 // Create Pointers to qp in MooseVariables for grains
 _ncrys = coupledComponents("v");

 _vals.resize(_ncrys);
 _grn_theta.resize(_ncrys);
 _grn_phi.resize(_ncrys);

 for (unsigned int i=0; i<_ncrys; ++i)
 {
   _vals[i] = &coupledValue("v", i);

   // Assign Orientations to Grains
   _grn_phi[i] = MooseRandom::rand()*(pi/2);
   _grn_theta[i] = MooseRandom::rand()*(pi/2);
 }

 // Import Grain Boundary Values
 initializeGBEnergies();

}

void
AnisotropicCuGrGr::computeProperties()
{
```

```
79  Real M0 = _GBmob0;

81  Real JtoeV = 6.24150974e18;// joule to eV conversion

83  Real T = 0.0;


85

    M0 *= _time_scale/(JtoeV*(_length_scale*_length_scale*_length_scale*
      _length_scale));//Convert to nm^4/(eV*ns);

87

    for(unsigned int qp=0; qp<_qrule->n_points(); qp++)
89  {
      if (_has_T)
91      T = (*_T)[qp];
      else
93      T = _temp;

95    // Check if it is on grain boundary
      if(onGrainBoundary(qp))
97    {
        // Calculate Closest grains
99      unsigned int closest_grain;
        unsigned int next_closest_grain;
101

        findClosestGrains(closest_grain, next_closest_grain,qp);
103

        //std::cout << "The Closest Grain is " << closest_grain << std::
      endl;
105     //std::cout << "The Next Closest Grain is " << next_closest_grain
    << std::endl;
```

```
107     // Find Angles between two grains
        _phi = std::abs(_grn_phi[closest_grain] - _grn_phi[
    next_closest_grain]);
109     _theta = std::abs(_grn_theta[closest_grain] - _grn_theta[
    next_closest_grain]);

111     // Look-up Closest GrainBoundary Energy
        unsigned int phi_index = binarySearch(_possible_phi,_phi);
113     unsigned int theta_index = binarySearch(_possible_theta,_theta);

115     // Look Up Grain Boundary in table
        _GBenergy = _eng_lookup[phi_index][theta_index];
117
    }
119 else
      _GBenergy = 0.0;
121
    _sigma[qp] = _GBenergy*JtoeV*(_length_scale*_length_scale);// eV/nm
    ^2
123
    _M_GB[qp] = M0*std::exp(-_Q/(kb*T));
125
    _l_GB[qp] = _wGB; //in the length scale of the system
127
    _L[qp] = 4.0/3.0*_M_GB[qp]/_l_GB[qp];
129
    _kappa[qp] = 3.0/4.0*_sigma[qp]*_l_GB[qp];
131
    _mu[qp] = 3.0/4.0*1/_f0s*_sigma[qp]/_l_GB[qp];
133
    _entropy_diff[qp] = 8.0e3*JtoeV; //J/(K mol) converted to eV(K mol)
```

```
135
    _molar_volume[qp] = 7.11e-6/(_length_scale*_length_scale*
    _length_scale); //m^3/mol converted to ls^3/mol
137
    _act_wGB[qp] = 0.5e-9/_length_scale;
139  }
}
141
unsigned int
143 AnisotropicCuGrGr::binarySearch( std::vector<Real> &array, Real
    search_value)
{
145  unsigned int low = 0;
  unsigned int high = array.size() - 1;
147  unsigned int mid;

149  while (low + 1 != high)
  {
151    mid = (low + high)/2;
    if (array[mid] < search_value)
153    {
      low = mid;
155    }
    else
157    {
      high = mid;
159    }
  }
161  if (std::abs(mid - low) < std::abs(high - mid))
    mid = low;
163  else
```

```
    mid = high;
  return mid;
}

bool
AnisotropicCuGrGr::onGrainBoundary(unsigned int qp)
{
  Real value = 0;

  for (unsigned int i=0; i<_ncrys; ++i)
    value += (*_vals[i])[qp]*(*_vals[i])[qp];

  if(value < 1.0)
    return true;
  else
    return false;
}

void
AnisotropicCuGrGr::findClosestGrains(unsigned int &closest_grain,
    unsigned int &next_closest_grain, unsigned int qp)
{
  closest_grain = 0;
  next_closest_grain = 0;

  for (unsigned int i = 1; i < _ncrys; i++)
  {
    if((*_vals[i])[qp] > (*_vals[closest_grain])[qp])
    {
      closest_grain = i;
    }
```

```
  }
195
  for (unsigned int i = 1; i < _ncrys; i++)
197 {
    if(((*_vals[i])[qp] > (*_vals[closest_grain])[qp]) && i !=
   closest_grain)
199   {
      next_closest_grain = i;
201   }
  }
203 }

205 void
AnisotropicCuGrGr::initializeGBEnergies()
207 {
  Real value = 0.0;
209 Real step_size = 0.023898905039773;
  unsigned int array_size = 66;
211
  _possible_phi.resize(array_size);
213 _possible_theta.resize(array_size);

215 _eng_lookup.resize(array_size);
  for(size_t i = 0 ; i < array_size ; i++ )
217   _eng_lookup[i].resize(array_size);

219 for(unsigned int i = 0; i < array_size; i++)
  {
221   _possible_phi[i] = value;
    _possible_theta[i] = value;
223   value += step_size;
```

```
  }
225
  // Grain Boundary Energies
227 _eng_lookup[0][0] = 0.0;
_eng_lookup[0][1] = -0.077777;
229 _eng_lookup[0][2] = -0.169839;
_eng_lookup[0][3] = 0.133453;
231 _eng_lookup[0][4] = 0.087703;
_eng_lookup[0][5] = 0.288228;
233 _eng_lookup[0][6] = 0.257641;
_eng_lookup[0][7] = 0.408668;
235 // Continues On Forever
}
```

AnisotropicCuGrGr.C