Simulation of Metal Electrodeposition Using the Kinetic Monte Carlo and Embedded-Atom Methods

by

Tanyakarn Treeratanaphitak

A thesis presented to the University of Waterloo in fulfillment of the thesis requirement for the degree of Master of Applied Science in Chemical Engineering

Waterloo, Ontario, Canada, 2014

© Tanyakarn Treeratanaphitak 2014
I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.
Abstract

The effects of the microstructure of metal films on electric component performance and longevity have become increasingly important with the recent advances in nanotechnology. Depending on the application of the metal films and interconnects, certain microscopic structures and properties are preferred over others. A common method to produce these films and interconnects is through electrodeposition. As with every process, the ability to control the end product requires a detailed understanding of the system and the effect of operating conditions on the resulting product. To address this problem, a three-dimensional on-lattice kinetic Monte Carlo (KMC) method is developed to conduct atomistic simulations of single crystal and polycrystalline metal electrodeposition. The method utilizes the semi-empirical multi-body embedded-atom method (EAM) potential that accounts for the cohesive forces in a metallic system. The resulting computational method, KMC-EAM, enables highly descriptive simulations of electrodeposition processes to be performed over experimentally relevant scales.

In this work, kinetically controlled copper electrodeposition onto single crystal copper under galvanostatic direct-current conditions and polycrystalline copper under potentiostatic direct-current conditions is modelled using the aforementioned KMC method. Four types of surface processes are considered during electrodeposition: deposition, dissolution, surface diffusion and grain boundary diffusion. The equilibrium microstructures from single crystal experiments were validated using molecular dynamics (MD) simulations through the comparison of energy per atom and average coordination number. The growth mode observed is in agreement with experimental results for the same orientation of copper. MD simulation relaxes constraints and approximations resulting from the use of KMC. Results indicate that collective diffusion mechanisms are essential in order to accurately model the evolution of coating morphology during electrodeposition.

In the polycrystalline simulations, the effect of surface energy is taken into account in the propensities of deposition and dissolution. Sub-surface grain volume measurements were obtained from simulation results and the grain volume evolution with time is in agreement with both qualitative observations based on the deposit morphology and surface
energy calculations. Simulations of polycrystalline deposition agree with findings from experimental studies that the evolution of the root-mean-squared roughness of the deposit during the early stages of deposition follows a power law relationship with respect to time \( \approx t^n \). Furthermore, the power law exponent on time is determined to be \( n \approx 0.5 \), also in agreement with the experimental values reported in the literature.
Acknowledgements

First and foremost, I would like to express my deepest gratitude to my family, for none of this would be possible without their encouragement and support.

I would like to thank my supervisors, Professors Mark Pritzker and Nasser Mohieddin Abukhdeir for their guidance and support throughout the course of my studies.

A special thanks goes to HW and CCC, for your longstanding friendship and unwavering support. I would also like to thank my friends and the Thai Student Association for all of their support throughout my undergraduate and graduate studies. Thank you JVA, YY and members of the ARG for being an invaluable sounding board for the past two years.

Lastly, I would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for their financial support and Shared Hierarchical Academic Research Computing Network (SHARCNET) for the use of their computational facilities.
Table of Contents

List of Tables ix

List of Figures x

Nomenclature xii

1 Introduction 1

1.1 Research Motivation ........................................ 1
1.2 Objectives .................................................. 3
1.3 Structure of Thesis ......................................... 4

2 Background 5

2.1 Electrodeposition ........................................... 5
2.2 Embedded-Atom Method Potential ................................ 10
2.3 On-Lattice Kinetic Monte Carlo Method ........................ 11
2.3.1 Ising and Potts Models ................................ 13
2.3.2 Solid-on-Solid and Solid-by-Solid Methods ............. 15
2.4 Polycrystalline Systems .................................. 16
3 Past Approaches to KMC Simulations of Electrodeposition

3.1 Single Crystal Systems ........................................... 20
  3.1.1 Nucleation Studies ........................................ 21
  3.1.2 Morphological Studies ................................... 23
3.2 Simulations of Electrodeposition ................................ 24
3.3 Multiscale Simulations .......................................... 25

4 Electrodeposition onto a Single Crystal Substrate

4.1 Methodology ...................................................... 29
  4.1.1 Processes ................................................... 30
  4.1.2 Simulation Conditions .................................... 31
4.2 Results and Discussion ......................................... 35
  4.2.1 Kinetics of Diffusion Events ................................. 37
  4.2.2 Effect of Diffusion Mechanisms ............................. 41
  4.2.3 Comparison of Equilibrium Deposits ......................... 43
4.3 Conclusions ..................................................... 46

5 Electrodeposition onto a Polycrystalline Substrate

5.1 Electrochemical Kinetics ..................................... 47
5.2 Methodology ...................................................... 49
  5.2.1 Processes ................................................... 49
  5.2.2 Propensity Scaling ......................................... 51
  5.2.3 Representation of a Polycrystalline System ................ 52
  5.2.4 Substrate Generation ....................................... 54
List of Tables

4.1 Propensity functions for the possible events ..................................... 32
4.2 Parameters used in propensity functions for KMC-EAM .......................... 32
4.3 Average deposit cluster properties .................................................. 42

5.1 Propensity functions for the possible events ..................................... 50
5.2 Parameters used in propensity functions for KMC-EAM .......................... 50
5.3 Parameters used in Eqns (5.5) and (5.4) .......................................... 51
5.4 Comparison of \( \beta \) values for copper deposition .............................. 64
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Schematic diagram of an electrochemical cell</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Schematics of polycrystalline system</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Possible mechanisms for atom attachment</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Simulation system in multiscale SBS method</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Schematics of three possible diffusion mechanisms</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Flowchart of KMC-EAM algorithm</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>Morphology evolution of the configuration</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>Number of diffusion events over time</td>
<td>39</td>
</tr>
<tr>
<td>4.5</td>
<td>Equilibrium morphology at t = 5 s</td>
<td>40</td>
</tr>
<tr>
<td>4.6</td>
<td>Effect of current density on $\delta_E$</td>
<td>44</td>
</tr>
<tr>
<td>4.7</td>
<td>Effect of temperature on $\delta_E$</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Representation of a polycrystalline system</td>
<td>53</td>
</tr>
<tr>
<td>5.2</td>
<td>Flowchart of polycrystalline KMC-EAM algorithm</td>
<td>56</td>
</tr>
<tr>
<td>5.3</td>
<td>Morphology evolution when $\eta = -0.15$ V</td>
<td>58</td>
</tr>
<tr>
<td>5.4</td>
<td>Morphology evolution when $\eta = -0.10$ V</td>
<td>59</td>
</tr>
<tr>
<td>5.5</td>
<td>Morphology evolution when $\eta = -0.05$ V</td>
<td>60</td>
</tr>
</tbody>
</table>
5.6 Side view of final deposit morphology at different overpotentials . . . . . . 61
5.7 Grain volume/(initial volume) of the orientations with time . . . . . . . . 62
5.8 Variation of average roughness exponent with domain size . . . . . . . . . 63
5.9 Variation of the average final roughness with overpotential . . . . . . . . 65
5.10 Variation of the average final roughness of grain orientation with overpotential 65
Nomenclature

$\alpha_a$  Charge transfer coefficient for anodic reaction

$\alpha_c$  Charge transfer coefficient for cathodic reaction

$\beta$  Power law exponent

$\sigma$  State of system

$\Delta E$  Energy difference between final and initial states (eV)

$\delta_C$  Difference between the mean atom coordination number of the equilibrium deposit configurations from KMC-EAM and from MD-EAM (%)

$\delta_E$  Difference between the mean energy of the equilibrium deposit configurations from KMC-EAM and the potential energy component of the same relaxed configurations from MD-EAM (%)

$\delta_{ij}$  Kronecker delta

$\eta$  Overpotential (V)

$\Gamma_{i,j}$  Propensity at site $i$ due to event $j$ (s$^{-1}$)

$\gamma_i$  Activity of species $i$

$\mathbf{R}$  $3 \times 3$ rotation matrix
\( t \) \( 3 \times 1 \) translation vector

\( x_i' \) \( 3 \times 1 \) vector representing the new coordinates of site \( i \)

\( x_i \) \( 3 \times 1 \) vector of reference coordinates for site \( i \)

\( E \) Half-cell voltage (V)

\( E_{appl} \) Applied potential (V)

\( F \) Multi-body embedding energy functional (eV)

\( \nu \) Kinematic viscosity (m\(^2\) s\(^{-1}\))

\( \nu_{ads} \) Adsorption frequency (s\(^{-1}\))

\( \nu_a \) Frequency rate (s\(^{-1}\))

\( \nu_{desorb} \) Desorption frequency (s\(^{-1}\))

\( \nu_d \) Atomic vibrational frequency for diffusion (s\(^{-1}\))

\( \chi \) Average Euler characteristic (nm\(^{-1}\))

\( \overline{A} \) Average cluster area fraction

\( \overline{h} \) Average height (nm)

\( \overline{P} \) Average cluster perimeter (nm)

\( \overline{R}_{RMS} \) Average root-mean-squared roughness (nm)

\( \phi_{binding} \) Binding energy (eV)

\( \phi_{ij} \) Pair-wise repulsion between atoms \( i \) and \( j \) (eV)

\( \phi_{ss} \) Average potential energy between solid atoms (eV)

\( \rho_h \) Function that quantifies the electron density of a neighboring atom
\( \rho_i \)  Total host electron density for atom \( i \)

\( A \)  Substrate surface area (m\(^2\))

\( a \)  Lattice constant (nm)

\( a_{Cu} \)  Lattice constant of copper (0.3615 nm)

\( B \)  Arrhenius constant (mol s\(^{-1}\))

\( C \)  Power law constant

\( c \)  Concentration of the plating bath (mol m\(^{-3}\))

\( Cu \)  Copper

\( D \)  Diffusion coefficient (m\(^2\) s\(^{-1}\))

\( E \)  Total potential energy (Hamiltonian) of the system (eV)

\( e \)  Elementary charge (1.602 \( \times \) \( 10^{-19} \) C)

\( e^- \)  Electron

\( E_A \)  Energy of the system before an event (eV)

\( E_B \)  Energy of the system after an event (eV)

\( E_b \)  Grain boundary diffusion activation energy (eV)

\( E_{dep} \)  Activation energy of deposition (eV)

\( E_{chem_{dep}} \)  Chemical potential contribution to activation energy of deposition (eV)

\( E_{diss} \)  Activation energy of dissolution (eV)

\( E_{chem_{diss}} \)  Chemical potential contribution to activation energy of dissolution (eV)

\( E_{exch} \)  Atom exchange activation energy (eV)
$E_{hop}$  Hopping activation energy (eV)

$E_i$  Interaction energy of atom $i$ (eV)

$E_{step}$  Step-edge atom exchange activation energy (eV)

$F$  Faraday’s constant (96485 C mol$^{-1}$)

$f_s$  Substrate occupancy fraction

$G$  Probability per unit time (s$^{-1}$)

$h_i$  Height of each surface atom (nm)

$h_s$  Substrate layer height (nm)

$i$  Current density (A m$^{-2}$)

$i^0$  Exchange current density (A m$^{-2}$)

$i_{dep}$  Partial deposition current density (A m$^{-2}$)

$i_{diss}$  Partial dissolution current density (A m$^{-2}$)

$i_L$  Limiting current density (A m$^{-2}$)

$J$  Generic interaction potential

$k$  Rate constant (mol s$^{-1}$)

$k_B$  Boltzmann constant (8.617 $\times$ 10$^{-5}$ eV K$^{-1}$)

$L$  Number of possible grain orientations

$m$  Number of moles of metal (mol)

$M_{(aq)}^{z+}$  Metal ions in aqueous solution

$M_{(s)}^0$  Elemental metal
\( n \)  Number of atoms

\( n_c \)  Number of atoms within the potential cutoff (nm)

\( n_{dep} \)  Number of possible deposition sites per unit area (sites \( \text{m}^{-2} \))

\( n_{diss} \)  Number of possible dissolution sites per unit area (sites \( \text{m}^{-2} \))

\( n_{exch} \)  Number of atom exchange diffusion moves

\( n_{hop} \)  Number of hopping diffusion moves

\( n_n \)  Number of occupied first nearest neighbors

\( n_{step} \)  Number of step-edge atom exchange diffusion moves

\( P(\sigma) \)  Probability density

\( R \)  Gas constant (8.314 J mol\(^{-1}\) K\(^{-1}\))

\( r \)  Rate of reaction (mol s\(^{-1}\))

\( r_{ij} \)  The distance between atoms \( i \) and \( j \) (nm)

\( R_{RMS} \)  Root-mean-squared roughness (nm)

\( T \)  Temperature (K)

\( t \)  Time (s)

\( U \)  Uniform random number \( \in (0, 1) \)

\( w \)  Angular rotation speed (rad s\(^{-1}\))

\( Z \)  Partition function

\( z \)  Number of electrons transferred in reduction
Chapter 1

Introduction

1.1 Research Motivation

The study of metal deposition continues to attract attention due to its importance to industry and nanotechnology. One method of forming metal deposits is through electrodeposition which involves reduction of metal ions in an electrolyte solution to their elemental form. This reduction process can occur either electrolytically through the application of a current/potential to the electrode surface or electrolessly through the addition of a chemical reducing agent to the plating solution [1]. Electrodeposition can be used to form protective metal coatings on surfaces [2], electrodes [3], photoelectrodes used in photovoltaic cells [4, 5], catalysts [6], interconnects [7–9] and sensors [6].

When electrodeposition is used to fabricate metal or alloy coatings for devices, specific structures are desired to optimize performance. An example of this is copper interconnects where larger grains and (111) textured films are preferred to improve device longevity [10–12]. Another example is for catalysis where multimodal pore size distributions at the nano- and micron-scale are desired to increase the catalyst surface area [3]. Given the desire for specific deposit structures, it is important to study electrodeposition at the atomistic scale.
Atomistic simulations can provide information as to how process operating conditions can be varied to control the desired deposit structure. In addition to deposit morphology, atomistic simulations provide information regarding the kinetics and different phenomena such as grain growth and nucleation during the deposition process. This is extremely beneficial when aiming to understand the effects of specific phenomena on the final product. Simulations can also be performed to better design experiments to help isolate parameters of interest and validate macroscale observations at the atomic level. Phenomena at the atomic level can contribute to processes measured at the macroscale. Nucleation and growth processes are examples of phenomena that can be modelled at the atomic level [13–38].

Continuum models are appropriate for describing phenomena occurring at the macroscale or microscale but do not provide the same level of detail regarding the system as atomistic simulations. Atomistic simulation methods, such as molecular dynamics (MD), can capture the dynamics of the system down to the level of phonon vibrations. MD can also be used to determine kinetic parameters such as reaction rates. When MD is used in conjunction with a suitable interaction potential, it is a very effective method for simulation of metallic systems over a small time frame (nanoseconds) or simulation of final equilibrium states. The embedded-atom method (EAM) potential has been shown to accurately characterize metal/metal interactions [39] and predict relevant dynamics for systems including hydrogen adsorption onto nickel and segregation in binary alloys [40]. The EAM potential has been extensively validated for metallic systems [39–42] and used in MD simulations of hydrogen dissociation on nickel [42], self-diffusion of metals [40,41,43] and epitaxial growth [44].

A significant limitation of MD is its computational requirement since it explicitly accounts for thermal fluctuations. Thus, even with the use of parallel large-scale MD codes and a large number of parallel processors running over several days, simulations can only resolve time scales on the order of nanoseconds. Even accelerated MD methods such as hyperdynamics [45,46] and temperature-accelerated dynamics [47,48] are limited to small systems. Kinetic Monte Carlo methods, on the other hand, can simulate the dynamics of electrodeposition on time scales of seconds on a single-core computer over hours of computation time. Deposit morphologies and nucleation are still captured in KMC simulations,
but the required coarse-graining does not allow phonon vibrations to be considered. Based on these factors, KMC is an attractive method of simulating electrodeposition at the atomistic level.

1.2 Objectives

The overall objectives of this research are:

1. To develop a KMC method for simulating electrodeposition processes using the EAM potential.

2. To validate the KMC simulation method by comparison with experimental data reported in the literature.

To complete the aforementioned objectives, the following studies have been conducted:

1. KMC simulation of single crystal electrodeposition
   (a) Kinetics of diffusion events and the influence of each diffusion event on deposition.
   (b) Analysis of deposit morphology obtained from the simulations using quantitative morphological measures (Minkowski measures) to determine the effect of diffusion mechanisms.
   (c) Direct comparison of equilibrium morphologies from KMC simulations and MD simulations.

2. KMC simulation of polycrystalline electrodeposition
   (a) Comparison of growth kinetics to experimentally observed behaviour.
   (b) Morphological analysis of the deposit surface and grain volume using morphological measures.
1.3 Structure of Thesis

The thesis is organized into six chapters: Chapter 2 – background, Chapter 3 – literature review, Chapter 4 – KMC simulations of electrodeposition onto a single crystal substrate, Chapter 5 – KMC simulations of electrodeposition onto a polycrystalline substrate and Chapter 6 – conclusions and recommendations.

Chapter 2 will cover the relevant theoretical background to the studies. The on-lattice kinetic Monte Carlo method is described along with the embedded-atom method potential. Fundamentals of the electrodeposition process are also described.

Chapter 3 provides an overview of the current literature on KMC simulations of electrodeposition. Previous studies involving KMC simulations of electrodeposition are summarized. The review is divided into sections based on the nature of the study (morphological studies, nucleation studies, simulations of polycrystalline systems and multiscale simulations).

Chapter 4 presents results of the first part of the study – simulations of single crystal copper electrodeposition. Morphological analysis and kinetics of diffusion events are discussed in this chapter. Validation of the KMC-EAM method with MD simulations using the EAM potential (MD-EAM) is also included in this chapter.

In Chapter 5, the KMC-EAM method presented in Chapter 4 is extended to model polycrystalline systems. Results from simulations of copper electrodeposition onto a randomly generated polycrystalline copper substrate and morphological analysis are included.

Lastly, Chapter 6 summarizes the conclusions from this work and discusses possible extensions to the work.
Chapter 2

Background

The effective use of KMC to simulate electrodeposition requires an understanding of the fundamentals of both electrodeposition and KMC. In this chapter, the basics of electrodeposition, embedded-atom method potential, on-lattice kinetic Monte Carlo and polycrystalline systems are discussed.

2.1 Electrodeposition

Electrodeposition is a process of forming metal deposits electrochemically by reduction of metal ions in an electrolyte to metal atoms. The reduction of metal ions \( M^{z+}_{\text{aq}} \) in an aqueous electrolyte is represented by the following reaction [1]:

\[
M^{z+}_{\text{aq}} + ze^- \rightleftharpoons M^0_{(s)}.
\]  

(2.1)

The reaction proceeds when an external current or potential is applied to an electrochemical cell (Figure 2.1). In the case of the electrochemical cell in Figure 2.1, the reaction described in Eqn (2.1) occurs at the cathode. During operation, an electric potential across the electrochemical cell known as the cell voltage exists. The electrochemical cell such as the one in Figure 2.1 contains two half-cells, the anode and the cathode. The half-cells can
Figure 2.1: Schematic diagram of an electrochemical cell.
undergo different reactions, Eqn (2.1) is an example of a half-cell reaction that occurs at the cathode. Each half-cell reaction has a half-cell voltage ($\mathcal{E}$) associated to it. The half-cell voltage is a function of the equilibrium half-cell voltage and activity of the species involved in the reaction,

$$\mathcal{E} = \mathcal{E}^0 + \frac{RT}{zF} \ln \gamma^z_{M(aq)},$$  \hspace{1cm} (2.2)

where $\mathcal{E}^0$ is the standard (equilibrium) half-cell voltage measured against a reference electrode (V), $\gamma_{M(aq)}$ is the activity of the aqueous species, $z$ is the number of electrons transferred in the reaction and $F$ is Faraday’s constant (C mol$^{-1}$). The voltage calculated in Eqn (2.2) is with respect to the same reference electrode that $\mathcal{E}^0$ is measured against. This equation is commonly known as the Nernst equation [49].

The rate at which metal is deposited on the substrate is determined by the current through Faraday’s law of electrolysis [1]:

$$r = \frac{dn}{dt} = \frac{iA}{zF},$$  \hspace{1cm} (2.3)

where $r$ is the rate of the electrodeposition reaction (mol s$^{-1}$), $m$ is the number of moles of metal deposited, $t$ is time, $i$ is the current density (current per unit area, A m$^{-2}$) and $A$ is substrate surface area (m$^2$). The reaction rate can also be expressed in terms of the activity of the species involved in reaction (2.1) [1]:

$$r = \rightarrow r - \leftarrow r,$$  \hspace{1cm} (2.4)

$$\rightarrow r = k^r \gamma^z_{M(aq)},$$  \hspace{1cm} (2.5)

$$\leftarrow r = k^r \gamma^0_{M(s)} = k^r,$$  \hspace{1cm} (2.6)

where $k$ is the rate constant (mol s$^{-1}$) and the arrows denote the directions of the reaction based on Eqn (2.1).
The rate constants can be expressed using the Arrhenius equation [1]:

\[ \vec{k} = \vec{B} \exp\left( -\frac{E_{dep}}{k_B T} \right), \]  \hspace{1cm} (2.7)

\[ \vec{k} = \vec{B} \exp\left( -\frac{E_{diss}}{k_B T} \right), \]  \hspace{1cm} (2.8)

where \( B \) is a constant (mol s\(^{-1}\)), \( E_{dep} \) is the activation energy of deposition (eV) and \( E_{diss} \) is the activation energy of dissolution (eV). The activation energies can be expressed as the sum of the contribution due to the chemical potential and electric potential [1,50]:

\[ E_{dep} = E_{dep,chem} + \alpha_c z F \mathcal{E}_{appl}/e, \]  \hspace{1cm} (2.9)

\[ E_{diss} = E_{diss,chem} - \alpha_a z F \mathcal{E}_{appl}/e, \]  \hspace{1cm} (2.10)

where \( E_{chem}^{dep} \) and \( E_{chem}^{diss} \) are the chemical potential contributions to the activation energy of deposition and dissolution, respectively, \( \alpha_a \) is the charge transfer coefficient for the reverse direction of reaction (2.1) (i.e. dissolution), \( \alpha_c \) is for the forward direction of reaction (2.1) (i.e. deposition) and \( \mathcal{E}_{appl} \) is the applied electrode potential (V). The charge transfer coefficient is a measure of the effect of the applied potential to each reaction [49]. The elementary charge (e) appears on the right-hand side of Eqns (2.9) and (2.10) to convert the units of the electric potential contribution to eV. Substitution of Eqns (2.9) and (2.10) into Eqns (2.7) and (2.8) yields:

\[ \vec{k} = \vec{B} \exp\left( -\frac{E_{dep,chem}}{k_B T} \right) \exp\left( -\frac{\alpha_c z \mathcal{E}_{appl}}{k_B T} \right), \]  \hspace{1cm} (2.11)

\[ \vec{k} = \vec{B} \exp\left( -\frac{E_{diss,chem}}{k_B T} \right) \exp\left( \frac{\alpha_a z \mathcal{E}_{appl}}{k_B T} \right). \]  \hspace{1cm} (2.12)

The e term cancels if the value of the Boltzmann constant used in units of eV K\(^{-1}\), which is the case in this work. Thus, the partial current densities of the reactions in the forward
and reverse directions can be written as:

\[
\rightarrow i = \frac{zF\hat{B}z\hat{M}_{(aq)}}{A} \exp \left( -\frac{E_{\text{dep,chem}}}{k_BT} \right) \exp \left( -\frac{\alpha_c z\mathcal{E}_{\text{appl}}}{k_BT} \right), \tag{2.13}
\]

\[
\leftarrow i = \frac{zF\hat{B}}{A} \exp \left( -\frac{E_{\text{diss,chem}}}{k_BT} \right) \exp \left( \frac{\alpha_a z\mathcal{E}_{\text{appl}}}{k_BT} \right). \tag{2.14}
\]

At equilibrium, \( \rightarrow i = \leftarrow i = i^0 \) and \( \mathcal{E}_{\text{appl}} = \mathcal{E} \), where \( i^0 \) is the exchange current density (A m\(^{-2}\)). From Eqn (2.1), the overall current density is \( i = \frac{\leftarrow i}{\rightarrow i} \). Thus, the relationship between current density and potential can be written as:

\[
i = i^0 \left[ \exp \left( \frac{\alpha_a z\eta}{k_BT} \right) - \exp \left( -\frac{\alpha_c z\eta}{k_BT} \right) \right], \tag{2.15}
\]

where \( \eta \) is the overpotential (V) [1],

\[
\eta = \mathcal{E}_{\text{appl}} - \mathcal{E} \tag{2.16}
\]

and the exchange current density is given as:

\[
i^0 = \frac{zF\hat{B}z\hat{M}_{(aq)}}{A} \exp \left( -\frac{E_{\text{dep,chem}}}{k_BT} \right) \exp \left( -\frac{\alpha_c z\mathcal{E}}{k_BT} \right) \tag{2.17}
\]

\[
= \frac{zF\hat{B}}{A} \exp \left( -\frac{E_{\text{diss,chem}}}{k_BT} \right) \exp \left( \frac{\alpha_a z\mathcal{E}}{k_BT} \right). \tag{2.18}
\]

Eqn (2.15) has the form of the Butler-Volmer [1, 49, 50] equation that is commonly used to describe electrochemical kinetics.

At the atomic level, the partial current densities are functions of the frequency of deposition/dissolution and the nature of the surface. Assuming that the deposition sites are all of the same type and dissolution sites are also of the same type, the relationship is
given as [51,52]:

\[ \vec{i} = -i_{\text{dep}} = z e \Gamma_{\text{dep}} n_{\text{dep}}, \]  
\[ \vec{i} = i_{\text{diss}} = z e \Gamma_{\text{diss}} n_{\text{diss}}, \]  

where \( i_{\text{dep}} \) and \( i_{\text{diss}} \) are the partial current densities of each reaction (A m\(^{-2}\)), \( \Gamma \) is the frequency or propensity of the event occurring (s\(^{-1}\)), \( n_{\text{dep}} \) is the number of sites that deposition can occur at per unit area (sites m\(^{-2}\)) and \( n_{\text{diss}} \) is the number of sites that dissolution can occur at per unit area (sites m\(^{-2}\)).

### 2.2 Embedded-Atom Method Potential

The embedded-atom method potential is a semi-empirical potential that is based on (quantum) density functional theory (DFT) [39] and is widely used in MD simulations of metallic systems [40]. This potential closely describes the effect of metallic bonding in metal systems to accurately estimate the potential energy of an atom [40]. The potential of each atom (\( E_i \)) is composed of both multi-body and pairwise contributions [39]:

\[ E_i = \mathcal{F}[\rho_i] + \frac{1}{2} \sum_{i \neq j}^n \phi_{ij}(r_{ij}) \]  

where \( r_{ij} \) is the distance between atoms \( i \) and \( j \), \( E_i \) is the interaction energy of atom \( i \), \( \mathcal{F} \) is the multi-body embedding energy functional, \( n_c \) is the number of atoms within the potential cutoff and \( \phi_{ij}(r_{ij}) \) is a pair-wise repulsion between atoms \( i \) and \( j \). The function \( \rho_i \) is the total host electron density for atom \( i \):

\[ \rho_i = \sum_{j \neq i}^{n_c} \rho_h(r_{ij}) \]
where $\rho_h$ is a function that quantifies the electron density of a neighboring atom. The total potential energy of the system (Hamiltonian) is the sum of all of the interaction energies,

$$E = \sum_i^N \sigma_i E_i,$$

where $\sigma_i = 0$ when the site is vacant and $\sigma_i = 1$ when the site is occupied. The significance of the Hamiltonian will be discussed in the next section. The EAM parameters are estimated by fitting the predictions of the DFT calculations to known experimental values of metal properties such as the lattice constant, elastic constants, sublimation energy and vacancy-formation energy [39–42]. Since the EAM potential is based on DFT calculations, the approximations made in the DFT calculations will affect the parameters of the EAM potential.

### 2.3 On-Lattice Kinetic Monte Carlo Method

In MD, the exact locations of the atoms are determined and their motion is solved directly via Newton’s equations of motion [53]. However, this is computationally expensive and so is limited to evolution of the domain over short time scales. For metallic systems, it can be assumed that atoms vibrate about specific locations in quasi-equilibrium over a period of time. Since each of these locations corresponds to a minimum in potential energy of the system, an atom must overcome an energy barrier to move from one minimum to another [54]. Thus using a consistent fine-grained method, such as molecular dynamics or quantum mechanical density functional theory [54], the ground state lattice type (FCC, BCC, etc) and lattice spacing of a specific atomic system [41] are used as inputs for on-lattice KMC simulations. This is the basis of the on-lattice approximation for conducting KMC simulations of metal deposition via KMC [54], whereby the metal atoms’ positions are limited only to sites on this crystal lattice.

Utilizing the on-lattice approximation, the discretized microscopic state $\sigma$ of the system is a function of only lattice site occupancy and time, where $\sigma_i = 0$ for a vacant site and
σ_i = 1 for an occupied site. In order to utilize the KMC methodology, an additional coarse-graining approximation must be used which assumes that the domain evolves through a discrete set of independent dynamic mechanisms. Furthermore, these dynamic mechanisms are assumed to be Poisson processes \[55\]. Given these approximations, the KMC method enables numerical solution of the master equation of the system where the probability density \( P(\sigma) \) of observing state \( \sigma \) is given as \[54,55\]:

\[
\frac{dP(\sigma)}{dt} = \sum_{\sigma' \neq \sigma} G(\sigma' \rightarrow \sigma) P(\sigma') - \sum_{\sigma' \neq \sigma} G(\sigma \rightarrow \sigma') P(\sigma),
\]

(2.24)

where \( G(\sigma \rightarrow \sigma') \) is the probability per unit time that the system will undergo a transition from \( \sigma \) to \( \sigma' \). Alternatively, Eqn (2.24) is also known as the chemical master equation and may be reformulated as \[56\]:

\[
d\sigma_i = \sum_j \Gamma_{ij}^+(\sigma)dt - \sum_j \Gamma_{ij}^-(\sigma)dt,
\]

(2.25)

where \( \Gamma_{ij}(\sigma) \) is the transition probability (s\(^{-1}\)) or propensity function for process \( j \) at site \( i \) when the state \( \sigma \) is observed. The term \( \Gamma_{ij}(\sigma)dt \) gives the probability of state \( \sigma \) undergoing a change due to some move \( j \) at site \( i \) within the time increment \( dt \) \[57\]. The ‘+’ and ‘−’ signs denote whether site \( i \) is entering state \( \sigma \) or leaving state \( \sigma \). The propensity defines the frequency at which the events occur and also defines the possible events that can occur in the system.

When the system is in equilibrium (steady-state), the time derivative of \( P(\sigma_{eq}) \) is equal to zero. Eqn (2.24) becomes:

\[
\sum_{\sigma' \neq \sigma_{eq}} G(\sigma' \rightarrow \sigma) P(\sigma_{eq}') = \sum_{\sigma' \neq \sigma_{eq}} G(\sigma \rightarrow \sigma') P(\sigma_{eq}),
\]

(2.26)
where \([55,58]\):

\[
P(\sigma_{eq}) = Z^{-1} \exp \left( - \frac{E(\sigma_{eq})}{k_B T} \right),
\]

\[
Z = \sum_{\sigma} \exp \left( - \frac{E(\sigma)}{k_B T} \right).
\]

The sum of all the probabilities of the system undergoing a transition from state \(\sigma\) to \(\sigma'\) is equal to the sum of probabilities of the reverse transition. Eqn (2.26) is known as the detailed balance and is a fundamental constraint for Monte Carlo methods \([53,55,59]\).

Site \(i\) and event \(j\) are randomly selected through a KMC algorithm such as the Bortz-Kalos-Lebowitz (BKL) algorithm \([60]\). After an event has been selected and the transition has occurred, the simulation time is updated using the following expression \([55]\):

\[
\Delta t = -\frac{1}{\sum_i \sum_j \Gamma_{ij}} \ln(U),
\]

\[
t_{new} = t_{old} + \Delta t,
\]

where \(U\) is a uniform random number \(\in (0, 1)\). This expression relates the events in the KMC simulation to time.

### 2.3.1 Ising and Potts Models

The simplest example of a KMC model is the Ising spin model. The Ising model is based on the property of a ferromagnet where the atoms has two possible states: spin up and spin down \([61]\). This type of system can be adapted to represent the simplest form of crystal growth. In crystal growth, a site \(i\) can have two possible states, occupied \((\sigma_i = 1)\) and unoccupied \((\sigma_i = 0)\). The propensity function that describes the transition between the two states will be based on the kinetics of the system. An example of this is a simple
adsorption/desorption process with propensity functions as follow [54]:

\[
\Gamma_{\text{ads},i} = \nu_{\text{ads}} (1 - \sigma_i), \quad (2.31)
\]

\[
\Gamma_{\text{desorb},i} = \nu_{\text{desorb}} \sigma_i \exp \left( -\frac{\phi_{\text{binding}}(i)}{k_B T} \right). \quad (2.32)
\]

where \( \phi_{\text{binding}} \) is the binding energy (eV) and \( \nu_{\text{ads}} \) and \( \nu_{\text{desorb}} \) are adsorption and desorption frequencies (s\(^{-1}\)), respectively. The Hamiltonian of the system can be described using the following expression [54]:

\[
E = \sum_i^N E_i = \sum_i^N \sum_j^{n_e} J(r_{ij}) \delta_{\sigma_i,\sigma_j} \quad \sigma = 0 \text{ or } 1, \quad (2.33)
\]

where \( J \) can be any interaction potential and \( \delta_{\sigma_i,\sigma_j} \) is the Kronecker delta.

The Ising model is restricted to having two possible states, although it is possible to extend the model to \( Q \) possible states. Site \( i \) can now move between \( Q \) possible states as opposed to only two states. This \( Q \)-state model is referred to as the Potts \( Q \)-state model. In crystal growth, the \( Q \) state can refer to properties such as the misorientation angle of site \( i \) in a polycrystalline system. An example of this particular application of the Potts model is in ref. [37] where the propensity of diffusion from one site to another site with a different misorientation angle is represented by the following:

\[
\Gamma_{d,i} = \nu_d \exp \left( -\frac{E(\sigma_{\text{init}},\sigma_{\text{final}})}{k_B T} \right), \quad (2.34)
\]

where the energy barrier is a function of the states of both initial and destination sites. In the Potts model, the Hamiltonian is given as [62]:

\[
E = \sum_i^N E_i = \sum_i^N \sum_j^{n_e} J(r_{ij}) \delta_{\sigma_i,\sigma_j} \quad \sigma = 0, 1, \ldots, Q. \quad (2.35)
\]

In the case of polycrystalline crystal growth, the Hamiltonian would be modified to include
interactions between sites of different states,

\[ E = \sum_{i}^{N} E_i = \sum_{i}^{N} \sum_{j}^{n_c} J(r_{ij}, \sigma_i, \sigma_j) \quad \sigma = 0, 1, \ldots, Q. \]  \hspace{1cm} (2.36)

### 2.3.2 Solid-on-Solid and Solid-by-Solid Methods

One of the most widely-used methods of simulating crystal growth through KMC is the solid-on-solid (SOS) method developed by Gilmer and Bennema [63]. The term solid-on-solid comes from the fact that particles are treated as blocks that can stack on top of each other during crystal growth [58]. The SOS method is a ‘2+1’ dimensional method where the simulation domain is two-dimensional but at each site there exists a height parameter that represents the surface morphology [54, 63]. The propensity of adsorption/deposition events depends on the kinetics of the system while the propensity of surface diffusion events and desorption/dissolution events are determined by the number of occupied first nearest neighbors [58, 63]:

\[ \Gamma_{dep} = \nu_a \exp \left( \frac{E_{dep}}{k_BT} \right), \]  \hspace{1cm} (2.37)

\[ \Gamma_{diss} = \nu_a \exp \left( -\frac{n_n \phi_{ss}}{k_BT} \right), \]  \hspace{1cm} (2.38)

\[ \Gamma_{diff} = \nu_d \exp \left( \frac{E_A - E_B}{k_BT} \right), \]  \hspace{1cm} (2.39)

where \( \nu_a \) is the frequency rate of the event (s\(^{-1}\)), \( \phi_{ss} \) is the average potential energy between solid atoms (eV), \( n_n \) is the number of occupied first nearest neighbors, \( \nu_d \) is the atomic vibrational frequency for diffusion (s\(^{-1}\)) and \( E_A \) and \( E_B \) are energies of the system before and after the event (eV). \( E_A \) and \( E_B \) are also functions of \( n_n \), the solid-solid, fluid-fluid and solid-fluid interaction energies [63]. The Hamiltonian also depends on \( n_n \), the solid-solid, fluid-fluid and solid-fluid interaction energies and the state of the system [58].

While the SOS method requires less computational time than other KMC methods, this advantage comes at the expense of accuracy. The approximation that the interaction energy
depends only on the number of occupied first nearest neighbors is insufficient in metallic systems where the nature of metallic bonding implies a multi-body interaction. Another deficiency in the SOS method is the lack of vacancies in the deposit due the restriction that the particles must stack on top of another particle and not on a vacant block [27, 64]. Kaneko and coworkers [23–27] extended the SOS model to account for vacancy formation in the deposit. This solid-by-solid (SBS) method uses propensity functions that are based on Eqns (2.37), (2.38) and (2.39).

### 2.4 Polycrystalline Systems

Electrochemical experiments are commonly conducted on polycrystalline substrates so that many metal grains form and interact with each other during electrodeposition. In order to represent polycrystalline systems, more than one grain must be included in the simulation domain. Phenomena such as grain boundary migration (diffusion of atoms across grains) [65] and grain boundary diffusion (diffusion of atoms along grain boundaries) [66] must be taken into account along with deposition, nucleation and surface diffusion. In addition, the deposit will also contain grains that are at different orientation angles from each other. This orientation must also be taken into account in the simulation. These factors are some of the complexities in simulating polycrystalline systems that have been addressed in some form or another by researchers modeling electrodeposition and other modes of deposition. In this section, the different factors in simulations of polycrystalline systems are discussed with respect to a generic deposition process.

In polycrystalline systems, unaligned grains typically nucleate on a deposit surface [67]. Each grain can be described in terms of a misorientation angle, defined as the angle that it is rotated with respect to some reference lattice. This grain misorientation can be taken into account by either assigning a misorientation angle to each site relative to one grain that is arbitrarily chosen to serve as a reference [37, 68–72] or having different lattices with coordinates that are based on the grain misorientation angle [34, 35, 73–76]. The first approach is based on the Potts model, where the $Q$-state is now the misorientation angle.
Although it only requires one lattice to be generated, the morphology obtained from Potts-type simulations will not accurately reflect that of actual polycrystalline systems due to the system only having one physical lattice. The alternative approach is to define a separate lattice for each grain. The coordinates of the atoms in each grain now have a misorientation with the substrate. The sites in the lattices can now overlap and impinge on each other. This approach clearly gives a more realistic description of the system than that based on Potts-type models, but of course has the downside of being much more complex and requiring additional routines to track the growth of each lattice.

In metallic systems, an atom can diffuse from its location in one grain to another grain, causing the grain boundary to migrate. The rate of migration is temperature-dependent that can be modelled in terms of an Arrhenius-type relationship [65]. In some KMC models of deposition, surface diffusion is restricted to occur within a single grain [77, 78] and thus diffusion across grains is not taken into account. This approach simplifies the model and
reduces the computational cost. However, given that grain boundary migration can become very important to the deposit morphology especially as the temperature increases, this particular mechanism should not be ignored.

Grain boundary migration can be taken into account in two ways by either ‘switching’ the grain in which the atom is contained [37, 68–72] or permitting atoms to diffuse to a new site in a different grain [73–76]. Grain switching is the simpler approach since the only change is in the grain ‘number’ attributed to the atom. The transition probability of grain switching is given by an Arrhenius-type relationship where the activation energy is often taken to be the same as for diffusion but with an added grain boundary energy contribution [68–71]. This method is typically used in simulations where only one lattice is being considered and the atom coordinates are related to a reference coordinate. In this case, movement between grains occurs over a distance similar to that involved during surface diffusion within one grain and grain boundary migration involves essentially one atom switching its grain assignment.

As stated previously, the alternative method allows an atom to diffuse to a new site in a different grain [73–76]. This approach is used when the simulation takes into account the misorientation of the grains by having the atom coordinates reflect the actual location of the atoms as opposed to some reference coordinate. During grain boundary migration, atoms will diffuse across the grain boundary to a site within the lattice of the destination grain with some transition probability. The coordinates of the destination site are based on the misorientation of the grain. As a result of this misorientation of the lattices, the distance between the initial and final sites may not be equal to that between the two sites in a reference lattice. This approach is a more accurate way of representing grain boundary migration in polycrystalline systems than the grain switching approach.

The activation energy of grain boundary migration is obviously different from the activation energy of surface diffusion. This activation energy can be taken as the activation energy of surface diffusion with an added fixed energy barrier contribution from the boundary [37, 71, 73]. The more frequent approach is to include both a migration barrier and a difference in energy of the initial and final states [34, 35, 68–70, 72, 76]. This energy difference will account for the increase in activation energy when the event in question increases
the energy of the system. The contribution of the energy difference is only considered if the diffusion event results in an increase in the total energy of the system. This approach is based on transition state theory [79] and is discussed in detail in refs [80,81].
Chapter 3

Past Approaches to KMC Simulations of Electrodeposition

In the previous chapter, the theoretical background of KMC simulation of electrodeposition is discussed. In this chapter, the past approaches to KMC simulation of electrodeposition are discussed in a comprehensive literature review. The review is separated into three main sections: Section 3.1 – single crystal systems, Section 3.2 – polycrystalline systems and Section 3.3 – multiscale simulations.

3.1 Single Crystal Systems

The majority of the KMC studies of electrodeposition have considered single crystal systems. Since phenomena associated with grain boundaries, such as dislocations and energy effects, are neglected, single crystal systems are simpler to model than polycrystalline systems. However, in industrial applications, single crystals are not as commonly used as polycrystalline systems. Also, the formation of single crystal deposits requires different operating conditions from that of polycrystalline deposits [82]. The following section provides an overview of recent studies of single crystal systems including phenomena such as nucleation and morphological evolution of deposits.
3.1.1 Nucleation Studies

Two mechanisms describe the early stages of nucleation – instantaneous nucleation and progressive nucleation. Instantaneous nucleation occurs when a discrete set of possible nucleation sites are instantaneously assumed to become viable nuclei [1]. Given that the possible deposition sites are the same as possible nucleation sites at the onset of deposition, all of the possible deposition sites on the surface of the substrate will become occupied. Progressive nucleation occurs when the possible nucleation sites continuously form nuclei over time. In this case, the rate of formation of nucleation sites is a function of time, a rate constant, and total number of possible nucleation sites [1].

The study of nucleation of metals usually involves the deposition of less than one monolayer of atoms over a short period of time. Simulations of nucleation can be used to examine the effect of process parameters on the nucleation type (progressive versus instantaneous) and cluster formation. Additionally, the simulations have been compared with known theories of nucleation for specific conditions.

Simulations of nucleation have been conducted using the SOS approach, where the simulation time is restricted such that only sub-monolayer growth occurs. Stephens and Alkire [13] developed a method based on the SOS method to study the formation of step-edges and nucleation on a clean face-centred cubic (FCC) metal surface. The morphology of the simulated deposit was found to depend on the activation energy of surface diffusion. The SOS method was also used by Drews et al. [14] to investigate nucleation and the behaviour of the system at low overpotentials when deposition had not extended past the first monolayer. The energy barrier was determined based on the number of metal and substrate atoms that formed the nearest neighbours of the metal atom undergoing surface diffusion. From the study, the average number of clusters per unit surface area was found to increase when the ratio of the metal-(foreign) substrate surface diffusion energy barrier to metal-metal surface diffusion energy barrier was low.

Alternatively, descriptive potentials such as the EAM potential have also been used in place of the SOS method to study nucleation. In two studies by Gimenez et al. [16,17], two-dimensional KMC simulations with the EAM potential as the interaction potential were
used to study deposition of silver onto a gold substrate. Deposition was carried out at an arbitrary rate and the activation energy for surface diffusion was previously set for different atomic configurations. The first study focused on phase formation for a Frank-van der Merwe system (layer-by-layer growth) on two different surface orientations, Au(100) and Au(111) [16]. The model predicted that growth would occur on the Au(100) surface, but not on Au(111), which contradicts the thermodynamics of the system. The second study focused on simulation of potentiostatic electrodeposition where three different conditions for adatom deposition were considered (Figure 3.1). The conditions are the following:

1. deposition rate is uniform across the domain,
2. deposition only occurs at sites with vacant nearest neighbours,
3. deposition rate at sites with occupied nearest neighbours is higher than at sites with no occupied nearest neighbours [17].

The results from the three cases were found to fit an exponential expression of the surface coverage versus time relationship for each of the three modes.

The EAM potential is not the only descriptive interaction potential that can be used. Frank et al. [18] utilized a different multi-body interaction potential based on DFT. In this study, both KMC and grand-canonical Monte Carlo methods were used to simulate the early stages of Co-Ni alloy electrodeposition. The energies of the different atomic
configurations (i.e. the number of occupied neighbours within a cutoff distance) were stored in a look-up energy table used to calculate system energy during the simulation. Kinetic parameters used were estimated to ensure that the simulations matched the experimental current-potential data for Co-Ni alloy electrodeposition on a glassy carbon substrate.

In many cases, a simulation method for nucleation was presented and validated with existing theories. Guo et al. [21] introduced an alternative KMC model that is not based on the SOS method and accounts for ion diffusion in the bulk to the electrode via Brownian motion coupled to metal deposition determined using KMC. Arbitrary deposition frequencies were used and the simulated results were compared to known mathematical models for nucleation. Frank et al. [19] used a lattice-gas model to simulate nucleation during metal deposition to study the influence of nearest neighbour diffusion on surface coverage. The results were compared to that obtained using the Kolmogorov-Johnson-Mehl-Avrami (KJMA) theory of nucleation. The quasi-equilibrium distribution of clusters obtained from KJMA theory was found to be in agreement with simulation results. In a subsequent work, Frank and Rivkold [20] performed KMC simulations of nucleation using a two-dimensional Ising lattice-gas model to study the influence of surface adsorbate diffusion on phase change. The results were compared to KJMA theory for progressive and instantaneous nucleation. Classical nucleation theory has also been used to relate the KMJA theory to a KMC model [22]. The theory was extended to include kinks and cluster configurations, resulting in the extended classical nucleation theory. Predictions from the extended classical nucleation theory were found to be in good agreement with KMC results.

3.1.2 Morphological Studies

Simulations of the evolution of the morphology of coatings during electrodeposition are complicated by the fact that they involve longer time scales than that those needed for simulation of nucleation. Thus, considerable focus of KMC studies on deposit morphology has been to develop a method that accurately describes the behaviour of the system over longer time scales. The common properties used to characterize the deposit morphology
are surface roughness, cluster density and average cluster size. These parameters are
dependent on operating conditions and kinetic parameters which are adjustable in KMC
simulations.

The group of Braatz and Alkire has performed several morphological studies of copper
electrodeposition using KMC with the SOS method [15, 30–32]. The method used by the
group is a multiscale approach, as discussed in Section 3.3. Another morphological study
using a method very similar to the SOS method was conducted by Liu et al. [36]. They
carried out two-dimensional KMC simulations of the cross-section of single crystal copper
electro-deposits to study their morphological properties. They estimated the activation
energy for surface diffusion from the energy of the metal-metal bond, energy of metal-
substrate bond and number of occupied nearest neighbours, an approach similar to that
used in the SOS method. The operating conditions – electroplating bath concentration,
temperature and applied electrode potential – were found to affect the evolution of deposit
cluster density and cluster size over time. As the plating bath concentration and applied
electrode potential are decreased, the cluster density profile and variance of cluster size
profile change more gradually with respect to time. Kaneko et al. [23–27] used the SBS
method to account for the possible formation of vacancies in the deposit. Simulations were
carried out for different metallic systems and structures, including superfilling of copper
in sub-micron features involved in the fabrication of electronic devices and interconnects
[23–25].

3.2 Simulations of Electrodeposition

Due to their complexity, polycrystalline systems have not been modelled as frequently as
single crystal systems. As of this review, the only reported KMC simulations of the elec-
trodeposition of polycrystalline systems have been restricted to two-dimensional studies.
Liu et al. [34, 35] developed a cross-sectional two-dimensional KMC method for polycrys-
talline systems based on their previous work on single crystal systems [36]. Figure 2.2
describes the system modelled including how data are stored in this method. Correction
factors were introduced to account for the grain boundary energy, while grain orientations were determined randomly. Simulations of copper electrodeposition onto gold and copper substrates were performed under potentiostatic conditions. The morphologies were found to qualitatively agree with those observed in experimental studies of the same systems.

The EAM potential was also used to describe the interactions within a polycrystalline coating produced by kinetically controlled nickel electrodeposition under the presence of hydrogen impurities incorporated in the deposit in a study by Huang et al [37]. The effects of operating conditions (electrolyte temperature and deposition rate) on surface roughness, deposit grain size and relative grain density were examined. Relative grain density is a measure of the ratio of occupied sites in a grain to the maximum number of occupied sites in the same grain. As the electrolyte temperature increases, the relative grain density and average grain size increase. The opposite trend is observed when the deposition rate is increased, with the two measures decreasing almost linearly with deposition rate. Surface roughness is found to increase with an increase in deposition rate and decrease with an increase in electrolyte temperature.

### 3.3 Multiscale Simulations

Similar to polycrystalline systems, multiscale studies do not feature as prominently in the area of KMC simulations of electrodeposition. The majority of the work done on this topic was reported by Braatz and Alkire’s group. Their approach involved coupling ‘2+1’ dimensional simulations with the SOS method for the electrodeposition process to a continuum model for transport and aqueous chemistry in the bulk solution [15, 30]. In the earlier versions of the model, the KMC portion was coupled with a one-dimensional continuum model for the solution. The studies focused on copper electrodeposition on a flat copper surface. The presence of dissolved additives (polyethylene glycol and 3-mercapto-1-propane sulfonic acid) that are commonly incorporated in plating baths to help control deposit composition and morphology was not considered in the first study [30], but was included in the subsequent one [15]. The results of both studies did not agree with experimental observations for electrodeposition of copper onto a copper substrate [15, 30]. In the
case of ref. [15], the roughness evolution of the deposit did not agree with experimental observations while in ref. [30], the morphology obtained did not agree with those observed experimentally. This approach was later expanded [31–33] to simulate copper electrodeposition with additives. This involved coupling a KMC model for electrodeposition with a three-dimensional finite volume model for transport and solution chemistry in the bulk electrolyte [31]. Results from ref. [31] were used to perform parameter estimation of the rate constants by comparison of simulated and experimental data of roughness evolution and current-time transients. The use of the estimated parameters improved the agreement between experimental results and simulation results [32].

Additionally, Kaneko et al. [28] were successful in coupling MD with their SBS KMC method to model silver electrodeposition from a silver nitrate bath in the presence of arbitrary spherical polymeric additives. In their approach, transport of ions in the solution was modelled using MD, while the surface reaction was modelled using KMC. The KMC model described the deposit growth with kinetics affected by ion transport within the solution determined by MD. The influence of additives was considered in a similar approach used by others [31, 32] where an additive has an ‘action’ range within which sites are affected by the additive. Due to their complexity, the additives are excluded from the molecular dynamics simulation. The conditions in refs. [31, 32] were also simulated using a three-dimensional SBS method coupled with the coarse-grained random walk method [23]. The random walk method accounts for ion and additive transport in the solution. The simulation system is described in Figure 3.2 and the model was able to replicate the bottom-up filling required in trenches during damascene Cu electroplating [7].

Based on the literature reviewed in this chapter, only a few three-dimensional KMC methods have been reported. The existing SOS- and SBS-based models are inadequate in their ability to represent a metallic system. The short-comings of the two methods were discussed in detail in Section 2.3.2. Current approaches in representing polycrystalline systems do not involve three-dimensional KMC simulations with a highly descriptive interaction potential like the one developed in this research.
The application of treated coatings, which are useful as solid-electrolyte interfaces, has been studied extensively in the literature. Most studies on the growth of the electroplating deposit have been treated numerically by using a crystal growth model. In this study, we present a multi-scale simulation model for the solution–electrode interface. The simulation is performed by using the Kinetic Monte Carlo (KMC) approach, which allows the simulation to be performed for a 3D system (3D-KMC) without any approximations or simplifications. The simulation system is shown in Figure 3.2: Simulation system in ref. [23].

Figure 3.2: Simulation system in ref. [23].
Chapter 4

KMC-EAM Simulations of Electrodeposition onto a Single Crystal Substrate

The first step in developing a KMC method for simulating electrodeposition using the EAM potential (KMC-EAM) is to develop a method for single crystal systems. Single crystal systems have just one grain and thus grain boundary diffusion and grain boundary migration can be neglected. In this chapter, the single crystal KMC-EAM method is presented and applied to three-dimensional galvanostatic electrodeposition of a copper single crystal. The method is then validated by comparison with the equilibrium microstructures obtained by MD simulations with the EAM potential as the interaction potential (MD-EAM). The MD-EAM method relaxes a number of the constraints and assumptions of the KMC-EAM method: the on-lattice approximation, finite diffusion mechanisms and temporal coarse-graining. The simulations are conducted over a range of current densities and temperatures that match common experimental conditions. Simulations are then performed within these parameter ranges to predict the effect of current density and temp-

perature on surface morphology.

4.1 Methodology

The example chosen to apply and assess KMC-EAM in this work is copper electrodeposition onto a copper substrate (working electrode) from an acidic sulfate solution. The overall reaction for the cathodic reduction of $Cu^{2+}$ is:

$$Cu^{2+} (aq) + 2e^- \rightarrow Cu^0 (s).$$

(4.1)

$Cu^{2+}$ ion reduction proceeds through consecutive single-electron transfer steps and involves the formation of an intermediate in which Cu has oxidation state +1 [83, 84]. However, numerous studies have shown that the first of these steps has much slower kinetics than the second when copper deposition is carried out in acidic sulfate solutions [84]. Thus the first step is rate-determining [83, 84] and the two steps effectively occur almost simultaneously under these conditions. In this study, the deposition mechanism is assumed to be kinetically controlled. Thus, transport of $Cu^{2+}$ within the solution to the electrode surface has no influence on the deposition rate and so only phenomena occurring on the copper surface are considered in the model and simulations.

The EAM interaction potential parameters for copper are taken from Adams et al [41]. This potential is expressed as a function of the atom separation distance in the form of cubic splines, one for the embedding term and one for the pair-wise repulsion term. The energy of each atom is obtained by interpolating these splines according to the separation distance between the atom and each of its neighbours for both embedding energy and pair-wise repulsion contributions to the EAM potential. The neighbour contribution is limited to atoms within a cutoff distance of 0.495 nm, as is consistent with EAM parameters obtained from Adams et al [41]. The lattice used for KMC-EAM simulations is consistent with the EAM parameters for copper. This lattice type is FCC with a lattice spacing of 0.3615 nm which was determined experimentally and was one of the properties to which the EAM potential was fitted [41].
4.1.1 Processes

In this work, two dynamic processes are considered in modelling copper electrodeposition: i) reduction of metal ions and deposition onto the surface as adsorbed atoms (adatoms) and ii) diffusion of these adatoms on the surface. Diffusion in the bulk of the electrode is not considered since simulations are performed under conditions in which very few vacancies form [85]. Lattice relaxation mechanisms are not considered because the on-lattice approximation is used. Diffusion of adatoms on the deposit surface is complex and involves several collective mechanisms (concerted mechanisms) [43], in addition to nearest neighbour hopping.

Three possible adatom surface diffusion mechanisms (shown in Figure 4.1) are included in the model: hopping (single), atom exchange (collective) and step-edge atom exchange (collective). Hopping (Figure 4.1a) involves the diffusion of single adatoms and kink atoms, identified by coordination number \( \leq 6 \) [51], to unoccupied nearest neighbour sites. Most previous simulations include only this mechanism [13, 32, 34–36, 72, 76].
Atom exchange (Figure 4.1b) involves the simultaneous i) displacement of a sub-surface crystalline atom by a nearest neighbour adatom and ii) the hopping of the sub-surface crystalline atom to an unoccupied nearest neighbour site at the surface. Thus, an adatom and sub-surface crystalline atom exchange states so that the sub-surface atom becomes an adatom, while the adatom becomes part of the bulk [43].

A special case of atom exchange occurs when the exchange occurs at the edge of a terrace/step in the surface; this atom exchange process is called step-edge atom exchange (Figure 4.1c). Unlike the previously described atom exchange mechanism, the sub-surface atom hops horizontally within the same layer. The adatom becomes part of the surface crystal and sub-surface atom becomes either an adatom or a kink site depending on the coordination number of its new site.

The propensity functions for each type of diffusion event, used in the KMC-EAM method, are given in Table 4.1 [72]. The numerical values of the parameters contained in the propensity functions used in this work are given in Table 4.2. The activation energies are assumed to be constant regardless of the atomic configuration. The $\Delta E$ term in the propensity functions is evaluated using the EAM potential. The deposition propensity (Eqn (4.2)) is obtained from the relationship between the partial current density ($i_{dep}$) and deposition frequency given by Budevski et al [51]. The projected surface area of the domain in the $x−y$ plane is used to calculate the values of $n_{dep}$ and $n_{diss}$. This is an approximation of the surface area of the deposit which continually changes during the deposition process. Furthermore, simulations are restricted to copper deposition occurring at low enough currents that transport of Cu$^{2+}$ from the electrolyte to the cathode has no influence on the process.

4.1.2 Simulation Conditions

KMC-EAM simulations are carried out for a slab geometry that is infinite in the $x−y$ plane on which deposition occurs and semi-infinite in the $z$ direction normal to this plane. Periodic boundary conditions are assumed in the $x−y$ plane to approximate an infinite plane. The copper substrate surface is of the (100) orientation. In addition to the process
Table 4.1: Propensity functions for the possible events

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Propensity Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deposition</td>
<td>( \Gamma_{i,dep} = \frac{i_{dep}}{-ze^2n_{dep}} ) \hspace{1cm} (4.2)</td>
</tr>
</tbody>
</table>
| Hopping                          | \( \Gamma_{i,hop} = \begin{cases} 
\nu_d \exp\left(-\frac{E_{hop}}{k_B T}\right) & \Delta E \leq 0 \\
\nu_d \exp\left(-\frac{E_{hop} + \Delta E}{k_B T}\right) & \Delta E > 0 
\end{cases} \) \hspace{1cm} (4.3) |
| Atom exchange                    | \( \Gamma_{i,exch} = \begin{cases} 
\nu_d \exp\left(-\frac{E_{exch}}{k_B T}\right) & \Delta E \leq 0 \\
\nu_d \exp\left(-\frac{E_{exch} + \Delta E}{k_B T}\right) & \Delta E > 0 
\end{cases} \) \hspace{1cm} (4.4) |
| Step-edge atom exchange          | \( \Gamma_{i,step} = \begin{cases} 
\nu_d \exp\left(-\frac{E_{step}}{k_B T}\right) & \Delta E \leq 0 \\
\nu_d \exp\left(-\frac{E_{step} + \Delta E}{k_B T}\right) & \Delta E > 0 
\end{cases} \) \hspace{1cm} (4.5) |

Table 4.2: Parameters used in propensity functions for KMC-EAM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{dep} )</td>
<td>number of possible deposition sites per unit area</td>
<td>varies ( [=] ) sites m(^{-2} )</td>
</tr>
<tr>
<td>( e )</td>
<td>elementary charge</td>
<td>( 1.602 \times 10^{-19} ) C</td>
</tr>
<tr>
<td>( z )</td>
<td>number of electrons transferred in reduction reaction</td>
<td>( 2 )</td>
</tr>
<tr>
<td>( \nu_d )</td>
<td>atomic vibrational frequency</td>
<td>( 2 \times 10^{13} ) s(^{-1} )</td>
</tr>
<tr>
<td>( E_{hop} )</td>
<td>hopping activation energy</td>
<td>( 0.5 ) eV ([43])</td>
</tr>
<tr>
<td>( E_{exch} )</td>
<td>atom exchange activation energy</td>
<td>( 0.7 ) eV ([43])</td>
</tr>
<tr>
<td>( E_{step} )</td>
<td>step-edge atom exchange activation energy</td>
<td>( 0.2 ) eV ([43])</td>
</tr>
</tbody>
</table>
and material parameters presented above, input parameters for the simulations include the initial copper substrate seed layer height $h_s$ and the occupancy fraction $f_s$. The simulation domain sizes used range from $25a \times 25a \times 15a$ to $50a \times 50a \times 15a$ ($a_{Cu} = 0.3615$ nm is the lattice constant of copper [41,86]).

During the first stage of the simulation, $2.5 \times 10^4$ atoms are deposited at different deposition rates and allowed to diffuse. Following deposition of all the atoms, simulation continues (in the absence of further deposition) until the system reaches equilibrium. Equilibrium is identified when the change of the mean energy of the system with respect to time approaches zero with a tolerance of 1%.

The equilibrium configuration predicted by KMC-EAM in each case is evaluated by comparing it to the configuration obtained from a simulation using an established MD-EAM method. This is done to validate the equilibrium state obtained from KMC-EAM and not the dynamics predicted by KMC-EAM. This MD-EAM simulation uses the equilibrium configuration predicted by KMC-EAM as its initial condition and involves no further deposition to relax the constraints imposed by on-lattice KMC as described in Section 2.3. The MD-EAM simulations are carried out using the canonical ensemble (constant number of atoms, volume and temperature) at the same temperature as the corresponding KMC-EAM simulation over a period of 6 nanoseconds, which is sufficient for the relaxation of KMC constraints. The resulting configuration is then compared to that from KMC-EAM on the basis of the i) equilibrium energy per atom and ii) average coordination number.

The KMC simulation package that is the basis of the method is the Stochastic Parallel Particle Kinetic Simulator (SPPARKS:spparks.sandia.gov) [87]. The Gibson-Bruck [88] implementation of the direct Gillespie method is used to evolve the system. The KMC-EAM algorithm is illustrated in the flowchart in Figure 4.2. The MD simulation package used for comparisons of equilibrium deposits is the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS:lammps.sandia.gov) [89]. A MD-EAM run making use of 24 CPU cores typically requires a duration of several days to simulate 6 nanoseconds of relaxation, while KMC-EAM requires only 1 CPU and $\sim 12$ hours to complete a simulation in which 5 seconds of electrodeposition are modelled.
Start

Read input parameters for Eqns (4.2) – (4.5), lattice configurations and simulation run time \((t_{\text{max}})\)

Generate data structures for lattice

Compute the EAM potential at each site

Compute propensity (Eqns (4.2) – (4.5)) and energy (Eqn (2.21))

Select and perform an event based on the propensity distribution

Update the coordination number, energy and propensity of the site up to its second nearest neighbours

Advance the simulation clock by Eqn (2.29)

\(t \geq t_{\text{max}}?\)

no

yes

Stop

Figure 4.2: Flowchart of KMC-EAM algorithm
4.2 Results and Discussion

KMC-EAM simulations are performed to model electrodeposition of a fixed number \(2.5 \times 10^4\) of copper atoms for different sets of initial conditions which govern deposition rates. These initial conditions include domain size, thickness of the substrate layer and occupancy fraction of the substrate layer. Once deposited, the atoms are allowed to diffuse over the surface via the three mechanisms described in Figure 4.1.

Simulations are conducted over a range of deposition current densities and operating temperatures. Temperatures between 300 – 330 K are considered to span typical operating conditions used in industry and experimental studies. Current densities ranging from \(-10\) A m\(^{-2}\) to \(-1000\) A m\(^{-2}\) are chosen to span conditions from low to high deposition rates. This study is restricted to conditions where the deposition rate is kinetically controlled and unaffected by mass transfer. The \(\text{Cu}^{2+}\) concentration in the bulk is assumed to be 1 mol dm\(^{-3}\) to ensure that deposition remains in the kinetically controlled regime for all current densities applied in the simulations. At this bulk concentration, the highest current density of \(-1000\) A m\(^{-2}\) considered is less than 20\% of the limiting current density for copper deposition onto a disk electrode rotating at 1000 RPM, as estimated using the Levich equation [49,90]:

\[
i_L = 0.620 z FD^{2/3} w^{1/2} \nu^{-1/6} c.
\] (4.6)

In this expression, \(i_L\) is the limiting current density (A m\(^{-2}\)), \(D\) is the diffusion coefficient (m\(^2\) s\(^{-1}\)), \(w\) is the angular rotation speed (rad s\(^{-1}\)), \(\nu\) is the kinematic viscosity (m\(^2\) s\(^{-1}\)) and \(c\) is the concentration of the plating bath (mol m\(^{-3}\)). The initial occupancy fraction \(f_s\) in the substrate layer is taken to be 1.0 in every simulation, while the initial copper substrate layer height \(h_s\) is set to 1.1 nm. Sample electrodeposition deposit morphology evolution from a KMC-EAM simulation is shown in Figure 4.3.

The first set of results focuses on the influence of the different surface diffusion mechanisms considered in the KMC-EAM method on the resulting deposit roughness and nanoscale morphology. In particular, a comparison is made between the coatings obtained when surface diffusion occurs by hopping alone to those obtained when all three surface diffusion mechanisms operate. Equilibrium deposit morphologies were characterized using
Figure 4.3: Morphology evolution of the configuration from KMC-EAM simulation at a) 0 s, b) 0.1 s, c) 0.2 s, and d) 0.3 s. The current density is $-1000 \text{ A m}^{-2}$ and the operating temperature is 300 K. Colours denote coordination number (blue to red in ascending order). The substrate surface area is $40a_{Cu} \times 40a_{Cu}$ ($\approx 210 \text{ nm}^2$).
root-mean-squared roughness and local morphological measures – area, perimeter and average curvature. More detail on the evaluation of these morphological quantities and their meaning for deposit surfaces is provided in Section 4.2.2.

The second set of results involves the use of equilibrium deposit configurations from KMC-EAM, which correspond to electrodeposition over experimentally relevant timescales (seconds), as initial conditions for MD-EAM simulations. These MD-EAM simulations were used to determine the approximation error associated with the assumptions required for KMC-EAM – the on-lattice approximation, limitation of diffusion mechanisms and time coarse-graining – since these assumptions are not made in MD-EAM. The effect of the variation of the deposition rate and temperature on the accuracy of KMC-EAM was then determined in this way.

In order to characterize the kinetics of the deposition process, the mean energy and average coordination number of the configurations are used. A consideration in comparing KMC-EAM and MD-EAM results is that KMC does not explicitly account for the average kinetic energy of the atoms. Thus, the potential energy contribution to the total energy from MD-EAM is compared to the mean energy from KMC. The average absolute relative energy difference per atom ($\delta E$) and average absolute relative coordination number difference ($\delta C$) between KMC-EAM and MD-EAM are used as a measure of how equilibrium configurations from the KMC-EAM method compare to equilibrium configurations from MD-EAM. The average root-mean-squared displacement per atom (RMS displacement) in MD-EAM simulations is utilized as a means of tracking the distance atoms travel from their starting configuration, which corresponds to the equilibrium configuration from KMC-EAM.

### 4.2.1 Kinetics of Diffusion Events

Figure 4.4 shows the cumulative number of diffusion moves for each diffusion mechanism versus time for the first 1 s of simulation time for two different current densities ($-1000$ A m$^{-2}$ and $-100$ A m$^{-2}$). In both simulations, all diffusion mechanisms are active during the electrodeposition phase. Following the cessation of deposition (denoted with
the vertical line in Figure 4.4), the step-edge atom exchange diffusion (Figure 4.1c) ceases in both simulations while both the hopping (Figure 4.1a) and atom exchange (Figure 4.1b) surface diffusion mechanisms persist. During both the initial electrodeposition and relaxation regimes, diffusion events are observed to have a power law relationship with respect to time as indicated by the linear trends in Figures 4.4a-b. This implies that growth of the deposit surface occurs in a self-similar way where deposit morphology remains qualitatively unchanged as film thickness increases. Following deposition growth, the step-edge atom exchange ceases which indicates that only hopping and atom exchange diffusion mechanisms are important in the relaxation regime.

The hopping surface diffusion mechanism is found to be dominant both in the growth and equilibrium regimes. Any adatom can undergo hopping on the surface, while only atoms that satisfy the restrictions outlined in Section 4.1.1 can undergo atom exchange and step-edge atom exchange. Given that restrictions exist on the sites where atom exchange and step-edge atom exchange surface diffusion can occur, the observation that hopping is the most frequent event is expected.

The step-edge atom exchange is found to occur only during the growth regime, which is reasonable given that the mechanism results in a new configuration that precludes the possibility of the event happening again in that locality with the atoms undergoing the exchange. Given the conditions for the mechanism (Figure 4.1), diffusion via this mechanism ceases when deposition has stopped because no additional step-edges are being created. The duration of time during which the step-edge atom exchange mechanism is most active depends on the current density which determines the rate of deposition. As the deposition rate is increased, the interval over which the step-edge exchange mechanism is most active decreases. This is supported by the increase in the rate of change in $n_{\text{step}}$ during the deposition stage at a current density of $-1000$ A m$^{-2}$ versus that at $-100$ A m$^{-2}$ (Figure 4.4). Alternatively, the maximum value of $n_{\text{step}}$ is independent of deposition rate, comparing Figures 4.4a and 4.4b. Instead, the value of $n_{\text{step}}$ at any time is related primarily to the total number of atoms deposited up to that point.
Figure 4.4: Number of diffusion events ($n$) over time at a) 300 K and $-1000$ A $m^{-2}$ and b) 300 K and $-100$ A $m^{-2}$. Vertical line denotes end of deposition.
Figure 4.5: Equilibrium morphology at $t = 5 \text{s}$ of simulations with a) hopping as the only diffusion mechanism deposited at $-1000 \text{ A m}^{-2}$, b) hopping as the only diffusion mechanism deposited at $-100 \text{ A m}^{-2}$, c) all 3 diffusion mechanisms deposited at $-1000 \text{ A m}^{-2}$ and d) all 3 diffusion mechanisms deposited at $-100 \text{ A m}^{-2}$. Operating temperature is 300 K. Colours denote coordination number (blue to red in ascending order). The surface area of the substrates are $30a_{Cu} \times 30a_{Cu} \approx 120 \text{ nm}^2$. 
4.2.2 Effect of Diffusion Mechanisms

In order to study the role of the surface diffusion mechanisms considered in KMC-EAM (Figure 4.1) on deposit morphology, two sets of simulations were performed assuming that i) hopping alone and ii) all three modes operate. Past KMC simulation studies typically include only the hopping mechanism [23, 25, 34, 36, 76]. Restricting surface diffusion to only hopping precludes the possibility of adatoms diffusing from terraces in the deposit. KMC-EAM simulations were carried out under these two conditions at current densities of $-100 \text{ A m}^{-2}$ and $-1000 \text{ A m}^{-2}$. Equilibrium deposit configurations are shown in Figure 4.5 and a distinct difference in deposit morphology is observed independent of current density.

Deposit morphologies predicted by KMC-EAM simulations with hopping-only show a significant increase in roughness and cluster mean curvature. Deposits simulated when all three diffusion mechanisms are included are less rough and distinct terraces are formed that are large compared to the previous case. The root-mean-squared surface roughness ($R_{RMS}$) is calculated using [91]:

$$R_{RMS} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (h_i - \overline{h})^2},$$ (4.7)

where $h_i$ is the height of each surface atom, $\overline{h}$ is the average height, and $n$ is the number of surface atoms. As shown in Table 4.3, the average $R_{RMS}$ of the deposit when only hopping operates is significantly greater than when all surface diffusion modes are considered regardless of the current density. The relative difference between the two cases is significant since the absolute difference in the roughness will only increase as more layers are being deposited. The fact that the roughness of deposits formed when hopping is the only diffusion mechanism is twice that of when all three diffusion mechanisms are taken into account is an indicator of how the roughness will vary if more sites are to be deposited.

In addition to surface roughness, the morphology of the deposit surface was quantified using the Minkowski measures [92]. Three Minkowski measures are defined for a two-dimensional surface: surface area, perimeter and Euler characteristic. The Euler char-
Table 4.3: Average deposit cluster properties from Figure 4.5 – root-mean-squared roughness ($R_{RMS}$), cluster area fraction ($\bar{A}$), average cluster perimeter ($\bar{P}$) and Euler characteristic ($\bar{\chi}$).

<table>
<thead>
<tr>
<th>$i_{dep}$ (A m$^{-2}$)</th>
<th>Diffusion Mechanisms</th>
<th>$R_{RMS}$ (nm)</th>
<th>$\bar{A}$</th>
<th>$\bar{P}$ (nm)</th>
<th>$\bar{\chi}$ (nm$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1000 Hopping</td>
<td>0.279 ± 0.004</td>
<td>0.14 ± 0.00</td>
<td>8.4 ± 0.3</td>
<td>1651.2 ± 17.4</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>0.156 ± 0.004</td>
<td>0.74 ± 0.00</td>
<td>101.6 ± 43.7</td>
<td>1620.8 ± 21.7</td>
<td></td>
</tr>
<tr>
<td>-100 Hopping</td>
<td>0.287 ± 0.000</td>
<td>0.16 ± 0.00</td>
<td>9.8 ± 0.7</td>
<td>1607.7 ± 34.8</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>0.138 ± 0.004</td>
<td>0.80 ± 0.02</td>
<td>100.1 ± 29.3</td>
<td>1525.2 ± 265.1</td>
<td></td>
</tr>
</tbody>
</table>

The Euler characteristic is an integral measure of curvature over the cluster boundary. To compute these morphological measures from a given deposit surface, they are converted to binary images using surface depth as image intensity. Thus these morphological measures characterize the cluster morphology of the deposit.

Table 4.3 shows the morphological measures from the two sets of simulations. The average cluster area fraction ($\bar{A}$) is the average fraction of the total cluster surface area relative to the total surface area. The average cluster perimeter ($\bar{P}$) is the average perimeter of the clusters in the domain. The average Euler characteristic ($\bar{\chi}$) is related to the total curvature of the cluster boundaries within the simulation domain.

At $-1000$ A m$^{-2}$, the average cluster perimeter is lower when only hopping is involved than when three diffusion mechanisms are involved. This corresponds to smaller clusters which is supported by a reduction in the average total cluster area. Since step-edge atom exchange and atom exchange which tend to level the surface and coalesce the clusters do no occur, this result is expected. The measures obtained for deposition at $-100$ A m$^{-2}$ are consistent with those obtained at the higher current. When the three diffusion mechanisms are considered, $\bar{A}$ is an order of magnitude greater than that obtained when only hopping is considered.

The average perimeters for the two cases also agree with this trend by indicating smaller clusters when hopping is the only diffusion mechanism. The Euler characteristic and thus average curvature of the domains are similar, indicating that the curvature of the cluster
boundaries is determined by minimization of the cluster/bulk interfacial energy and not specific diffusion mechanisms.

The deposit surface features support the qualitative observation made based on Figure 4.5. When hopping is the only diffusion mechanism, the deposit has greater roughness and the individual clusters are smaller. The growth mode observed when three surface diffusion mechanisms are included is similar to that of Cu/Cu(100) homoepitaxial growth observed experimentally [93,94].

4.2.3 Comparison of Equilibrium Deposits

The final set of simulations was performed over a range of initial conditions, current densities and temperatures using KMC-EAM. Equilibrium deposit configurations from these KMC-EAM simulations were then used as initial conditions for MD-EAM simulations under commensurate conditions (temperature and ensemble). Through relaxation of the approximations required to perform KMC, the MD-EAM simulations results were used to determine the validity of the equilibrium structure predicted by the KMC-EAM method for simulations of the electrodeposition process. In all KMC-EAM simulations, the occupancy fraction \( f_s \) is set to be 1.0, corresponding to electrodeposition on an atomically smooth copper crystal.

Figure 4.6 shows the difference \( \delta_E \) between the mean energy of the equilibrium deposit configurations from KMC-EAM and the potential energy component of the same relaxed configurations from MD-EAM. The simulation results span current densities ranging from \(-10\) to \(-1000\) A m\(^{-2}\) at 300 K. It is observed that \( \delta_E \) is non-negligible but reasonable over the full range of applied current densities. The increase of \( \delta_E \) with respect to current density is expected in that an increased deposition rate results in the formation of vacancies which result in lattice relaxations that are not considered in KMC-EAM. Furthermore, lattice relaxation at the deposit surface is also not considered, which contributes to \( \delta_E \).

The difference in atom coordination number \( \delta_C \) was also determined in order to compare the KMC-EAM equilibrium configurations to those of MD-EAM. These plots are not shown.
Figure 4.6: Effect of current density on $\delta_E$ obtained by MD-EAM and KMC-EAM simulations at 300 K, $h_s = 1.1$ nm and $f_s = 1.0$.

since the values of $\delta_C$ were all negligible, less than 0.04%. This implies that deposit morphology from KMC-EAM is almost identical to the average morphology from MD-EAM. Furthermore, current density was not found to have a statistically significant effect on $\delta_C$. Thus, the difference in energy $\delta_E$ is primarily a consequence of the on-lattice approximation of KMC-EAM and not to any significant difference in the deposit morphology.

Figure 4.7 shows the difference in energy $\delta_E$ between KMC-EAM and MD-EAM equilibrium deposit configurations for applied current density of $-10$ A m$^{-2}$ over a range of temperatures $300 - 330$ K. A similar magnitude and trend of $\delta_E$ is observed as in the previous case with $\delta_E$ being non-negligible (ideally, $\delta_E \approx 0$) but small over the full range of operating temperatures. The values of $\delta_C$ are again negligible and thus not shown. The results can be interpreted in the same way as before, but now increasing temperature results in the increased formation of vacancies and also increased lattice strain in the MD simulations. The trend is slightly steeper than that observed from the increase of current density, which implies that the KMC-EAM method will monotonically decrease in accuracy as temperature is increased. Since the range of operating temperatures used in this work
Figure 4.7: Effect of temperature on $\delta_E$ obtained by MD-EAM and KMC-EAM simulations at $-10 \text{ A m}^{-2}$, $h_s = 1.1 \text{ nm}$ and $f_s = 1.0$.

is typical for electrodeposition processes, KMC-EAM performs adequately in comparison to MD-EAM within this range.

The final metric used to evaluate the deposit configuration predicted by KMC-EAM is the RMS displacement of atoms from their starting positions obtained from the metastable configuration of KMC-EAM to reach their final positions as computed by MD-EAM. The RMS displacement value for equilibrium single crystal copper deposits is reported to be 0.0113 nm at 300 K [95]. The RMS displacement values for the KMC-EAM simulations with current density varied at 300 K was found to range between 0.019 – 0.021 nm. For the set of simulations in which temperature was varied, the RMS displacement values range between 0.019 – 0.023 nm. These results indicate that the equilibrium configuration predicted by KMC-EAM simulations is essentially equivalent to that of MD-EAM. Furthermore, the RMS displacement values appear to be only slightly affected by the operating conditions, which supports the interpretation of $\delta_E$ and $\delta_C$ trends discussed previously.
4.3 Conclusions

A kinetic Monte Carlo methodology which uses the embedded-atom method potential and includes collective diffusion mechanisms (KMC-EAM) for single crystal systems has been developed. This methodology was applied to the simulation of galvanostatic electrodeposition of metals onto a single crystal substrate of the same species. The average energy per atom and coordination number of equilibrium configurations from KMC-EAM were validated using MD simulation. KMC-EAM was found to be accurate for deposition current density and temperature values relevant to experimental conditions. Furthermore, the KMC-EAM accurately describes the nanoscale structure of the metal deposit through direct representation of the constituent atoms, unlike the SOS and SBS methods.

In addition to analysis of equilibrium configurations, the effects of surface diffusion mechanisms (hopping, atom exchange and step-edge exchange) and diffusion kinetics were also studied. Results show that the inclusion of collective diffusion mechanisms (atom exchange and step-edge exchange), in addition to nearest neighbour hopping, were required to predict deposit configurations in agreement with both MD-EAM simulations and experimental results for Cu/Cu(100) homoepitaxy. The inclusion of the three surface diffusion mechanisms resulted in quantitatively smoother deposits, as reflected by surface morphology measures – roughness, cluster perimeter and cluster area.

The diffusion kinetics observed indicated that the step-edge exchange mechanism was active predominantly during the deposition process, while hopping and atom-exchange continued following the cessation of electrodeposition. In summary, the presented KMC-EAM method is shown to provide an accurate representation of the electrodeposition process over experimentally relevant length (microns) and time (seconds) scales.
Chapter 5

KMC-EAM Simulations of Electrodeposition onto a Polycrystalline Substrate

In the previous chapter, the KMC-EAM method was developed for single crystal systems. This chapter extends the KMC-EAM method to polycrystalline systems and applies it to three-dimensional electrodeposition of copper onto a smooth polished polycrystalline substrate. In addition to the polycrystalline deposition, the potential dependence of deposition is also included in the simulations. Simulations are performed over a range of overpotentials to predict the effect on deposit texture evolution. Results from the simulations are compared to experimental observations and known surface energies of different grain orientations of copper.

5.1 Electrochemical Kinetics

In this study, the KMC-EAM method is extended to polycrystalline electrodeposition and applied to study potentiostatic deposition. As was the case in the previous chapter, metal
deposition is assumed to proceed by a one-step reaction. Thus, the reduction of copper ions (\(Cu^{2+}_{(aq)}\)) to form copper atoms (\(Cu^0_{(s)}\)) occurs as follows:

\[
Cu^{2+}_{(aq)} + 2e^- \rightleftharpoons Cu^0_{(s)}. \tag{5.1}
\]

The rate of deposition is controlled by the applied current through Faraday’s law of electrolysis [49]. In the case of potentiostatic electrodeposition, the applied current depends on the applied potential. The current-potential relationship used in this study is based on the Butler-Volmer equation used by Cabán and Chapman [50] assuming that electrodeposition is kinetically controlled and the plating bath concentration is 1 mol dm\(^{-3}\):

\[
i_{Cu} = i_{Cu}^0 \left[ \exp \left( \frac{\alpha_a \eta}{k_B T} \right) - \exp \left( \frac{-\alpha_c \eta}{k_B T} \right) \right], \tag{5.2}
\]

where the parameters were previously described in Section 2.1. In terms of the deposition partial current density (\(i_{dep}\)) and dissolution partial current density (\(i_{diss}\)), Eqn (5.2) can be rewritten as:

\[
i_{Cu} = i_{diss} + i_{dep}, \tag{5.3}
\]

where:

\[
i_{diss} = i_{Cu}^0 \exp \left( \frac{\alpha_a \eta}{k_B T} \right), \tag{5.4}
\]

\[
i_{dep} = -i_{Cu}^0 \exp \left( \frac{-\alpha_c \eta}{k_B T} \right). \tag{5.5}
\]

Eqns (5.4) and (5.5) are the basis for the propensity functions of electrochemical kinetics in KMC-EAM.
5.2 Methodology

5.2.1 Processes

Four main dynamic processes are considered for copper electrodeposition: i) reduction of copper ions and deposition onto the substrate surface, ii) oxidation of copper atoms and dissolution into the electroplating bath, iii) diffusion of adatoms on the surface within a single grain and iv) diffusion of atoms along the grain boundaries. Aside from grain boundary diffusion (i.e. diffusion along a grain boundary), which has been observed at near ambient temperatures [66], diffusion within the bulk of the copper deposit is neglected since the operating conditions are in the range where few vacancies are formed [85]. Grain boundary migration (i.e. diffusion across a grain boundary) in the bulk is also assumed negligible at the operating temperatures considered here, which is supported by past work [96]. This restriction is not applied to adatoms, which are allowed to diffuse between grains.

The three surface diffusion mechanisms – hopping, atom exchange and step-edge atom exchange – are described in detail in Section 4.1.1. In addition to the assumptions made in Chapter 4, the activation energies of the events are also assumed to be constant regardless of the grain orientation. Grain boundary diffusion follows the same Arrhenius relationship as other diffusion mechanisms [97]. The activation energy for grain boundary diffusion (Table 5.2) is assumed to be 0.5 eV regardless of the type of grain boundary. This value is based on results obtained from simulation studies in ref. [97].

The propensity functions of each diffusion event are given in Table 5.1 [72] with the parameters given in Table 5.2. The assumptions made regarding the surface area used in the calculation of \( n_{dep} \) in Chapter 4 also applies to \( n_{diss} \). Eqns (5.6) and (5.7) are obtained from the relationship between current density and frequency given in ref. [51]. The current density of deposition and dissolution are defined in Eqns (5.6) and (5.7), respectively. Parameters for the Butler-Volmer equation applicable when the electroplating bath concentration is 1 mol dm\(^{-3}\) are given in Table 5.3 [50].
Table 5.1: Propensity functions for the possible events

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Propensity Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deposition</td>
<td>$\Gamma_{i,\text{dep}} = \frac{i_{\text{dep}}}{z e n_{\text{dep}}} (5.6)$</td>
</tr>
<tr>
<td>Dissolution</td>
<td>$\Gamma_{i,\text{diss}} = \frac{i_{\text{diss}}}{z e n_{\text{diss}}} (5.7)$</td>
</tr>
<tr>
<td>Hopping</td>
<td>$\Gamma_{i,\text{hop}} = \begin{cases} \nu_d \exp \left( \frac{-E_{\text{hop}}}{k_B T} \right) &amp; \Delta E \leq 0 \ \nu_d \exp \left( \frac{-E_{\text{hop}} + \Delta E}{k_B T} \right) &amp; \Delta E &gt; 0 \end{cases} (5.8)$</td>
</tr>
<tr>
<td>Atom exchange</td>
<td>$\Gamma_{i,\text{exch}} = \begin{cases} \nu_d \exp \left( \frac{-E_{\text{exch}}}{k_B T} \right) &amp; \Delta E \leq 0 \ \nu_d \exp \left( \frac{-E_{\text{exch}} + \Delta E}{k_B T} \right) &amp; \Delta E &gt; 0 \end{cases} (5.9)$</td>
</tr>
<tr>
<td>Step-edge atom exchange</td>
<td>$\Gamma_{i,\text{step}} = \begin{cases} \nu_d \exp \left( \frac{-E_{\text{step}}}{k_B T} \right) &amp; \Delta E \leq 0 \ \nu_d \exp \left( \frac{-E_{\text{step}} + \Delta E}{k_B T} \right) &amp; \Delta E &gt; 0 \end{cases} (5.10)$</td>
</tr>
<tr>
<td>Grain boundary diffusion</td>
<td>$\Gamma_{i,b} = \begin{cases} \nu_d \exp \left( \frac{-E_b}{k_B T} \right) &amp; \Delta E \leq 0 \ \nu_d \exp \left( \frac{-E_b + \Delta E}{k_B T} \right) &amp; \Delta E &gt; 0 \end{cases} (5.11)$</td>
</tr>
</tbody>
</table>

Table 5.2: Parameters used in propensity functions for KMC-EAM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{dep}}$</td>
<td>number of possible deposition sites per unit area</td>
<td>varies [sites m$^{-2}$]</td>
</tr>
<tr>
<td>$n_{\text{diss}}$</td>
<td>number of possible dissolution sites per unit area</td>
<td>varies [sites m$^{-2}$]</td>
</tr>
<tr>
<td>$e$</td>
<td>elementary charge</td>
<td>$1.602 \times 10^{-19}$ C</td>
</tr>
<tr>
<td>$z$</td>
<td>number of electrons transferred in reduction reaction</td>
<td>2</td>
</tr>
<tr>
<td>$\nu_d$</td>
<td>atomic vibrational frequency</td>
<td>$2 \times 10^{13}$ s$^{-1}$</td>
</tr>
<tr>
<td>$E_{\text{hop}}$</td>
<td>hopping activation energy</td>
<td>0.5 eV [43]</td>
</tr>
<tr>
<td>$E_{\text{exch}}$</td>
<td>atom exchange activation energy</td>
<td>0.7 eV [43]</td>
</tr>
<tr>
<td>$E_{\text{step}}$</td>
<td>step-edge atom exchange activation energy</td>
<td>0.2 eV [43]</td>
</tr>
<tr>
<td>$E_b$</td>
<td>grain boundary diffusion activation energy</td>
<td>0.5 eV [97]</td>
</tr>
</tbody>
</table>
Table 5.3: Parameters used in Eqns (5.5) and (5.4) [50].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{Cu}^{0}$</td>
<td>107.75 A m$^{-2}$</td>
</tr>
<tr>
<td>$\alpha_a$</td>
<td>1.08</td>
</tr>
<tr>
<td>$\alpha_c$</td>
<td>0.39</td>
</tr>
</tbody>
</table>

5.2.2 Propensity Scaling

In this study, the surface energies may vary between grains since the substrate is polycrystalline. Consequently, the propensities of deposition and dissolution are not uniform across the surface. To account for the difference in surface energy of the different grain orientations of copper, an energy contribution term ($\Delta E/E_{sub}$) is applied to the propensity of deposition:

$$\Gamma'_{i,dep} = \Gamma_{i,dep} \frac{\Delta E}{E_{sub}},$$

(5.12)

where $\Delta E$ is the change in energy of site $i$ after the site becomes occupied and $E_{sub}$ is the sublimation energy of copper ($E_{sub,Cu} = -3.54$ eV [41]). Sites with lower surface energies have higher $\Delta E$ values and thus will preferentially undergo deposition. However, this energy contribution will alter the average propensity and thus the deposition partial current density. A scaling factor is applied to $\Gamma'_{i,dep}$ at every site at which deposition can occur. Equation (5.12) becomes:

$$\Gamma'_{i,dep} = \Gamma_{i,dep} \frac{\Delta E}{E_{sub}} s_{dep},$$

(5.13)

where $s_{dep}$ is the scaling factor which is initially set to 1. The scaling factor is calculated by making use of the average of the deposition propensities:

$$\Gamma_{dep} = \frac{1}{n_{dep}A} \sum_i \Gamma'_{i,dep},$$

(5.14)
where \( A \) is the surface area (\( m^2 \)). Thus,

\[
s_{\text{dep}} = \frac{\Gamma_{\text{dep}}}{\Gamma_{i,\text{dep}}}.
\]  

(5.15)

Previously computed propensities are also updated, but by a factor of \( s_{\text{dep}}/s_{\text{dep,old}} \), where \( s_{\text{old}} \) is the scaling factor from the previous update, bringing the average propensity to \( \Gamma_{i,\text{dep}} \).

In the case of dissolution, the energy contribution is taken into account in a slightly different manner from deposition:

\[
\Gamma'_{i,\text{diss}} = \Gamma_{i,\text{diss}} \frac{E_{\text{sub}}}{-\Delta E}.
\]  

(5.16)

The difference stems from the fact that higher energy sites are more likely to be removed than lower energy sites. Higher energy sites will result in lower \( |\Delta E| \) values and thus higher \( E_{\text{sub}}/(-\Delta E) \). The negative sign in front of \( \Delta E \) is due to the value of \( \Delta E \) being positive in dissolution processes. The scaling factor is determined based on the same procedure as deposition, but now by making use of \( \Gamma'_{i,\text{diss}} \) and \( n_{\text{diss}} \) (defined in Table 5.2):

\[
\Gamma_{\text{diss}} = \frac{1}{n_{\text{diss}} A} \sum_i \Gamma'_{i,\text{diss}}.
\]  

(5.17)

### 5.2.3 Representation of a Polycrystalline System

The KMC-EAM method is extended to represent polycrystalline systems for this study. Information on the number of crystal orientations in the simulation and details about the orientation are now required by the KMC-EAM method. A set of reference coordinates is generated based on the (100) crystal orientation of the metal. The reference coordinates will form the basis on which the coordinates in each orientation are based. The grain orientation is related to a rotation matrix and a translation vector as follows:

\[
x'_i = Rx_i + t.
\]  

(5.18)
where $\mathbf{x}'_i$ is a $3 \times 1$ vector representing the new coordinates of site $i$, $\mathbf{R}$ is a $3 \times 3$ rotation matrix, $\mathbf{x}_i$ is a $3 \times 1$ vector of reference coordinates for site $i$ and $\mathbf{t}$ is a $3 \times 1$ translation vector. Coordinates are rotated to provide the desired orientation normal to the $z$ direction (i.e. the direction exposed to the electroplating solution at $t = 0$) and translated so that the majority of the sites remain in the simulation domain. Any sites that are outside of the simulation domain after translation and rotation are removed but remain in the reference coordinates matrix. While the sites can overlap with each other, occupied sites cannot. Vacant sites with a neighbour from a different orientation in the $a\sqrt{2}/2$ radius that is occupied are excluded from the list of possible deposition sites. Figure 5.1 shows the representation of the sites in the KMC-EAM method.

A list of active sites in each grain is maintained to track the size of each grain. This list consists of sites with coordination numbers between 1 and 11, i.e. sites that are not
completely part of the bulk. When an active site reaches the edge of the generated grain, the grain will grow on the face at which the active site is located. The growth will double the size of the grain in that particular direction. Sites outside the simulation domain are removed from the simulation.

5.2.4 Substrate Generation

In the simulations, copper atoms deposit onto a polycrystalline copper substrate. This polycrystalline substrate is randomly generated by KMC-EAM based on a specified number of possible grain orientations \( L \) and a specified seed layer height \( h_s \). The substrate generation procedure is as follows:

1. The \( L \) possible grain orientations are defined such that they span the entire \( x-y \) plane up to at least \( h_s \).
2. Two seed sites are randomly placed in each of the \( L \) grain orientations in the \( z=0 \) plane.
3. The first nearest neighbours of the seed sites are allowed to ‘grow’ (become occupied).
4. The nearest neighbours of the newly occupied sites are then allowed to grow until the grain comes into contact with another grain, as overlap with other grains is not allowed. This is repeated until no more sites that can become occupied in the seed layer exist.
5. The seed layer is then ‘polished’ down to 0.75\( h_s \). All sites above 0.75\( h_s \) become unoccupied. This is to imitate the effects of polishing the substrate prior to deposition in experiments.

5.2.5 Simulation Conditions

KMC-EAM simulations are carried out for a slab geometry that is periodic in the \( x \) and \( y \) directions to approximate an infinite plane. The algorithm for the polycrystalline KMC-
EAM method is illustrated in Figure 5.2. The copper substrate contains grains with (100), (110) and (111) planes normal to the $z$ direction. The three planes are frequently found in copper [98–105] and their surface energies have been reported in literature [106, 107]. The substrate seed layer height is $h_s = 1.4$ nm. The simulation domain size used is $40a \times 40a \times 40a$ where $a_{Cu} = 0.3615$ nm is the lattice constant of copper [41,86].

During the deposition stage of the simulation, dissolution can also occur. The propensity of dissolution is given in Eqn (5.7). Since Eqn (5.2) is based on the assumption that the electroplating bath concentration is $1 \text{ mol dm}^{-3}$, the highest overpotential in DC mode is $\eta = -0.15$ V in order to remain in the kinetically controlled regime (see Section 4.2 for details). The deposition overpotentials applied in this study are $-0.05$ V, $-0.10$ V and $-0.15$ V. The computational requirement to deposit 70000 sites onto $40a \times 40a$ surface is three days using one CPU.

5.3 Results and Discussion

Simulations using KMC-EAM are performed to model electrodeposition of a fixed number of atoms by the application of direct different overpotentials at 300 K. In this study, the simulation is stopped after all of the atoms have been deposited. The first set of results focuses on the effect of the size of the domain on the accuracy of the results. If the simulation domain is too small, there exists a finite size effect that will affect the kinetics predicted by the simulation. The size of the simulation domain should not have an effect on the kinetics of the system, thus it is important to determine the minimum domain size that is unaffected by finite size effect. Roughness evolution over time is chosen as a morphological measure to determine whether there is a finite size effect at a particular domain size. This measure is chosen because the roughness-time relationship follows a power law and the parameters in the power law are easily quantifiable. The exponent of the power law should be independent of domain size if finite size effect does not exist. The exponent in the roughness-time power law relationship obtained at different domain sizes is compared to determine the minimum domain size required. The second set of results
Read input parameters for Eqns (5.6) – (5.11), lattice configurations and simulation run time ($t_{\text{max}}$)

Generate data structures for each lattice. Apply Eqn (5.19) to coordinates in each grain and create initial substrate

Compute the EAM potential at each site

Determine active and overlapping sites in each grain

Compute propensity (Eqns (5.6) – (5.11)) and energy (Eqn (2.21))

Calculate and apply scaling factors (Section 5.2.2)

Select and perform an event based on the propensity distribution

Update the coordination numbers, active sites list and check status of overlapping sites

Generate new sites?

yes

Create new sites and update data structure

no

Compute energy and propensity of $i$ up to second nearest neighbours

Calculate and apply scaling factors (Section 5.2.2)

Advance the simulation clock by Eqn (2.29)

$t \geq t_{\text{max}}$?

no

yes

Stop

Figure 5.2: Flowchart of polycrystalline KMC-EAM algorithm
focuses on the effect of overpotential on the morphology of the deposit, specifically the roughness.

Figure 5.3 shows an example of the morphology evolution when 70000 atoms are being deposited at an overpotential of $-0.15 \text{ V}$. In Figure 5.3a, the (100) grain have the largest surface area compared to the other two grains at the start of deposition in the randomly generated substrate (Section 5.2.4). However, as atoms are deposited, the (111) grain grows at a noticeably faster rate, covering the (100) grain to merge with the other (111) grain (Figure 5.3d). The same behaviour is observed when the 70000 atoms are being deposited at $\eta = -0.10 \text{ V}$ and $\eta = -0.05 \text{ V}$ (Figures 5.4 and 5.5, respectively). The height difference of each orientation is clear in the side view of the deposit (Figure 5.6). From the figure, the growth of the (111) plane is three-dimensional while growth of the (100) plane is two-dimensional and that of (110) is between the two extremes. This qualitative observation is in agreement with experimental results for copper homoepitaxy on (100) and (111) planes [94,108,109].

Figure 5.7 shows how the volume of each orientation increases with respect to its initial volume for the first 0.1 s of deposition. The grain volume (111) orientation is found to be increasing at a faster rate than the other two orientations at the three overpotentials. This supports the qualitative observation made based on the deposit morphology in Figures 5.3, 5.4 and 5.5. This behaviour stems from the differences in the surface energies of the different faces of copper [106,107]. Since the (111) orientation has the lowest surface energy, atoms will preferentially deposit on this surface.

KMC-EAM accounts for the non-uniform deposition and dissolution propensities through differences in energy contributions (Section 5.2.2). Experimental results indicate that the (111) plane is the most dominant orientation in copper deposits [98–105]. Based on this and the result observed in Figures 5.3 – 5.7, the method discussed in Section 5.2.2 accurately captures the preferential growth of (111)-oriented grains in copper deposition. However, it is important to note that while (111) grows preferentially in all cases, the growth of the (110) and (100) planes observed experimentally appears to be dependent on both the substrate and deposition conditions.
Figure 5.3: Morphology evolution when a) 0 % (the randomly generated substrate), b) 33.3 %, c) 66.7 % and d) 100 % of the 70000 atoms have been deposited at $\eta = -0.15$ V. Colours denote grain orientation. The surface area of the substrates are $40\alpha_{Cu} \times 40\alpha_{Cu}$ ($\approx 210$ nm$^2$).
Figure 5.4: Morphology evolution when a) 0 % (the randomly generated substrate), b) 33.3 %, c) 66.7 % and d) 100 % of the 70000 atoms have been deposited at $\eta = -0.10$ V. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu}$ ($\approx 210$ nm$^2$).
Figure 5.5: Morphology evolution when a) 0 % (the randomly generated substrate), b) 33.3 %, c) 66.7 % and d) 100 % of the 70000 atoms have been deposited at $\eta = -0.05 \text{ V}$. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu}$ ($\approx 210 \text{ nm}^2$).
Figure 5.6: Side view of final deposit morphologies at a) $\eta = -0.05$ V, b) $\eta = -0.10$ V and c) $\eta = -0.15$ V. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu} (\approx 210 \, \text{nm}^2)$.

5.3.1 Effect of Domain Size

The size of the domain can have an effect on the simulation results even when periodic boundary conditions are used. If the domain size is too small, the behaviour of the system captured will be affected by the apparent proximity of grains to each other. Thus, it is important to determine the minimum domain size for which the results are not affected.

In order to determine this critical (minimum) size, simulations of varying domain sizes were performed. The domain sizes chosen are $20a_{Cu} \times 20a_{Cu} \times 40a_{Cu}$, $25a_{Cu} \times 25a_{Cu} \times 40a_{Cu}$, $30a_{Cu} \times 30a_{Cu} \times 40a_{Cu}$, $35a_{Cu} \times 35a_{Cu} \times 40a_{Cu}$ and $40a_{Cu} \times 40a_{Cu} \times 40a_{Cu}$. The parameter used to track the kinetics of the system is the evolution of surface roughness over time. Surface roughness is affected by the kinetics of the system, if the kinetics predicted by the simulation is affected by finite size effect, the roughness evolution will also be affected. At each domain size, the equivalent of 20 monolayers for that domain was deposited and the roughness evolution of the deposit over time was calculated. Depositing a fixed number of atoms for all domain sizes will result in significant variations in the morphology evolution. The equivalent of one monolayer in one domain size could be equivalent to many monolayers in another domain size, this will render the roughness-time data incomparable. The RMS roughness of the surface is calculated using Eqn (4.7). The roughness evolution is assumed
Figure 5.7: Grain volume/(initial volume) of the (100), (111) and (110) orientations with time at a) $\eta = -0.05$ V, b) $\eta = -0.10$ V and c) $\eta = -0.15$ V.
to follow a power law relationship with time:

\[ R_{RMS} = Ct^\beta, \]
\[ \log R_{RMS} = \log C + \beta \log t, \]

where \( C \) is a constant and \( \beta \) is the power law exponent.

The average value of \( \beta \) obtained at each domain size is shown in Figure 5.8. Once the domain size is increased to \( 30a_{Cu} \times 30a_{Cu} \times 40a_{Cu} \) the roughness exponent converges to \( \beta \approx 0.5 \) and becomes independent of domain size. Thus, it appears that the size of the domain no longer affects the roughness of the deposit when it becomes larger than \( 30a_{Cu} \times 30a_{Cu} \times 40a_{Cu} \). Thus the critical domain size is \( 30a_{Cu} \times 30a_{Cu} \) (\( \approx 120 \text{ nm}^2 \)) in the \( x - y \) direction. The value \( \beta \approx 0.5 \) also agrees very well with the roughness exponents obtained from additive-free electrodeposition experiments [15, 110] and copper sputtering experiments [111] with atomic force microscopy measurements (Table 5.4). Previous atomistic simulations have failed to obtain values remotely close to those obtained experi-

63
mentally without the help of adjustment parameters [15,32].

### 5.3.2 Effect of Overpotential on Roughness

In this study, the overpotential is varied to determine its effect on the surface roughness. Simulations were carried out with a domain size of $40a_{Cu} \times 40a_{Cu} \times 40a_{Cu}$ and 70000 atoms being deposited. When substituted into Eqn (5.2), the overpotentials yield current densities in the range that was studied in Chapter 4. Figure 5.9 shows the variation of the average RMS roughness with overpotential. The RMS roughness appears to be unaffected by the increase in overpotential. This trend is in agreement with the results obtained for single crystal deposition (Table 4.3) where the difference between RMS roughness at $-1000 \text{ A m}^{-2}$ and $-100 \text{ A m}^{-2}$ is in the order of $10^{-2} \text{ nm}$.

From the results shown in Figures 5.3 – 5.5, the three orientations appear to exhibit different growth modes and the roughness of each plane varies. The (111) grain is observed to undergo three-dimensional growth, which is supported by the orientation-specific roughness measurements in Figure 5.10. The RMS roughness of the (111) grain is twice that of the (100) grain since the adatoms in the (111) grain undergo fewer step-edge atom exchange events than the adatoms in the other two orientations. The RMS roughness of the (110) grain is slightly higher than that of (100), supporting the observation from Figures 5.3 – 5.5 that the growth mode of (110) is between fully three-dimensional and two-dimensional.

While the effect of deposition rate on roughness is similar for single crystal simulations
Figure 5.9: Variation of the average final roughness with overpotential. Error bars denote standard deviation.

Figure 5.10: Variation of the average final roughness of each grain orientation with overpotential. Error bars denote standard deviation.
and polycrystalline simulations, the actual roughnesses are significantly different. The overall RMS roughness of polycrystalline deposits are four times that of single crystal deposits. In single crystals, no grain boundary effects that could affect the roughness of the deposit obviously occur. The grain-specific roughness of the (100) grain is higher than the roughness reported for single crystal deposition onto the same grain orientation, indicating that the presence of other grains can have an effect on the roughness of a deposit.

5.4 Conclusions

The KMC-EAM methodology has been extended to simulate potentiostatic deposition onto polycrystalline substrates in addition to the galvanostatic deposition onto single crystal substrates previously presented. The atom dissolution mechanism that exists in electrodeposition has also been incorporated into the kinetics captured by KMC-EAM. Grain boundary diffusion and surface diffusion across grains are taken into account in this novel method to describe polycrystalline-specific diffusion events. The method uses Butler-Volmer kinetics to describe the potential dependence of deposition and dissolution rates. The effect of grain orientation on deposition and dissolution kinetics are taken into account through the EAM potential. This results in preferential growth of (111) orientation in agreement with the experimental behaviour that has been previously reported. The growth modes observed are in agreement with experimental results for Cu/Cu(100) and Cu/Cu(111) homoepitaxy. In addition, KMC-EAM was found to accurately predict the exponent in the roughness-time power law relationship with respect to experimental data without the need for any adjustment parameters, which previous atomistic simulations were unable to do.
Chapter 6

Conclusions

6.1 Conclusions

In this work, a novel KMC methodology for simulating electrodeposition under both galvanostatic and potentiostatic conditions using the highly descriptive EAM potential was developed. The KMC-EAM method was found to accurately predict the kinetics of electrodeposition for both single crystal and polycrystalline systems. The polycrystalline KMC methodology is the first three-dimensional polycrystalline method that utilizes the EAM potential and supports both galvanostatic and potentiostatic deposition modes to simulate electrodeposition. The morphology of the deposits follow the same trends observed experimentally. The general conclusions of this work are:

- The KMC-EAM method accurately describes deposit morphology over experimentally relevant deposition rates and temperatures.
- Step-edge atom exchange mechanism is predominantly active during the deposition process.
- Collective diffusion events are required to accurately predict deposit morphology.
• Propensity scaling allows for surface energy to be accurately taken into account in deposition kinetics.

• Roughness-time relationship predicted by KMC-EAM is in agreement with experimental results when the domain size is larger than the critical domain size.

6.2 Recommendations

1. Several simplifications were made with respect to the kinetics of the system. It is recommended that future studies remove the following simplifications:

   (a) **Constant surface area** – In propensity calculations of deposition and dissolution, the surface area term is assumed to be constant and is approximated to be the projected surface area of the domain in the $x-y$ plane. This assumption is valid when the deposit is smooth, which is not always the case. The surface area should be computed at every time step and based on the actual number of surface sites.

   (b) **Activation energy** – The activation energy of each diffusion event is assumed to be constant regardless of grain orientation. In reality, this is not the case, as some diffusion events will preferentially occur in one texture over another. Different activation energies should be used based on the grain orientation.

2. The main limiting factor with KMC-EAM is the size of the domain that can be simulated in a reasonable time frame. It is recommended that the following optimization be made to the KMC-EAM method:

   (a) **Pre-tabulating distances between sites** – The majority of computational time is spent evaluating the EAM potential. This calculation makes use of the distance between sites, which is discarded after the potential calculation is completed. Since the distance between sites does not change with time, it can be stored and would not have to be recalculated after every event.
(b) *Identifying surface diffusion events* – The current approach in identifying surface diffusion events involves a series of *if/else* statements. Instead, an external graph library can be used to identify the configuration of the atom and thus the possible surface diffusion events that can occur.

(c) *Optimizing neighbour solver and neighbour update* – The current neighbour solver is a brute force solver. When the grain data structure grows, the neighbour solver recomputes the neighbour lists using a brute force algorithm. This brute force solver is extremely slow and the neighbour update can be limited to current active sites and new sites.

(d) *Limiting EAM potential evaluation to active sites* – Currently, the EAM potential is evaluated over all sites in the domain. This is unnecessary, as sites in the bulk will not experience any change to their potential energy.

3. While the KMC-EAM method is able to accurately predict the dynamics of electrodeposition of metal onto the same metal substrate, this condition is only one of the many possible modes of electrodeposition. The following types of simulations should also be considered:

(a) *Coarse-grained KMC* – Coarse-grained KMC will help KMC access longer time scales and large domain sizes.

(b) *Multiscale simulations* – Multiscale simulations will enable access to either larger time scales through continuum equations or more detailed kinetics through smaller length scale methods such as DFT or MD. This will also allow for mass transfer effects in the adjacent electrolyte to be studied, which in turn will allow for experimentally relevant simulations of pulsed deposition to be conducted.

(c) *Strained heteroepitaxial growth* – Incorporation of the effect of elastic energy on deposit morphology in metal deposition onto a foreign substrate will significantly extend the applications of the KMC-EAM method.
Appendix – KMC-EAM Documentation

The following documentation is generated using Doxygen\(^2\).

**SPPARKS\_NS::AppEAM Class Reference**

#include <app.eam.h>

Inheritance diagram for SPPARKS\_NS::AppEAM:

```
AppLattice
```

```
SPPARKS\_NS::AppEAM
```

**Public Member Functions**

- `AppEAM` (class SPPARKS *, int, char **)
- virtual `∼AppEAM` ()
- virtual void `grow_app` ()
- virtual void `init_app` ()

\(^2\)URL: [http://www.stack.nl/~dimitri/doxygen/index.html](http://www.stack.nl/~dimitri/doxygen/index.html)
- virtual void setup_app ()
- virtual void setup_end_app ()
- virtual double site_energy (int, int)
- virtual void site_event_rejection (int, class RandomPark *)
- virtual double site_propensity (int)
- virtual void site_event (int, class RandomPark *)
- void stats (char *)
- void stats_header (char *)

Private Member Functions

- double system_energy (int, int, int, int, int)
- int neighbor_check (int, int, int)
- int edge_check (int, int)
- void coord_update (int, int)
- void update_dep_flag (int)
- void gen_seed_layer (class RandomPark *, vec_int)
- void update_status (int, int, int)
- void check_grain_distance (int)
- void check_grain_distance (int, int)
- int determine_direction (int, int)
- double get_local_coordinates (array_coordinates &, int, int)
- int exchange_destination (int, int, int)
- void scale_propensities ()

Private Attributes

- int seed_n
- int n_hop
- int n_atomexch
- int n_step
- int n_grain
- int n_boundary
• vec_int_array_2 diff_sites
• vec_int sites
• double i_dep
• double i_diss
• double alpha_a
• double alpha_c
• double i0
• double eta
• double ox_rate
• int dep_mode
• int diss_sites_count
• vec_int diss_sites_count_grain
• double i_a_dep
• double i_a_diss
• double z_me
• double Ed
• double Ed_exch
• double Ed_step
• double Ed_grain
• double Ed_boundary
• double v_d
• double v_exch
• double v_step
• double v_grain
• double v_boundary
• double seedlayer_size
• double seed_frac
• double charge
• int n_diss
• int n_max
• int dep_sites_count
• vec_coordinates phi_old
• double old_eng
• int phi_old_count
• int count
• vec_int dep_sites_count_grain
• double max_eng
• double polish_height
• vec_int dep_sites
• vec_int diss_sites
• double scale_dep_on
• double scale_dep_off
• double scale_diss_on
• double scale_diss_off
• int pulse
• double on_dt
• double off_dt
• double prev_time
• int pulse_on
• vec_vec_double dep_diss_prob
• PairEAM * pair

Constructor & Destructor Documentation

AppEAM::AppEAM ( class SPPARKS * spk, int narg, char ** arg )

Constructor for AppEAM Arguments for AppEAM:

1. seed layer height in Angstroms
2. fraction of number of occupied sites in seed layer
3. dep - deposition mode (p - potentiostatic; g - galvanostatic)
4. z_me - number of electrons required for metal reduction
5. n_max - max number of atoms adsorbed
6. Ed - hopping diff activation energy (eV)
7. Ed_exch - atom exchange diffusion activation energy (eV)
8. Ed_step - step edge atom exchange activation energy (eV)
9. v_d - vibrational freq of atom for hopping (1/s)
10. v_exch - vibration freq of atom for atom exchange (1/s)
11. v_step - vibration freq of atom for step edge atom exchange (1/s)
12. Ed_grain - activation energy for diffusion across grain boundaries (eV)
13. Ed_boundary - activation energy for diffusion ALONG grain boundaries (eV)
14. v_grain - vibration freq of atom for diffusion across grain boundaries (1/s)
15. v_boundary - vibration freq of atom for diffusion along grain boundaries (1/s)

If galvanostatic deposition:

1. i_dep - deposition current density (pA/nm^2; 1 pA/nm2 = 10^6 A/m2)

If potentiostatic deposition:

1. eta - overpotential (V vs SHE)
2. alpha_a - transfer coefficient for anodic reaction
3. alpha_c - transfer coefficient for cathodic reaction
4. i0 - exchange current density (pA/nm^2; 1 pA/nm2 = 10^6 A/m2)

Plating mode:

1. pulse - pulse-plating type (dc->0, pp->1, or pr->2)

If DC, no parameters required
If pulse-plating:

1. pulse_on - current on time
2. pulse_off - current off time

If pulse-reverse:

1. pulse_on - current on time
2. pulse_off - current off time
3. ox_rate - oxidation rate - overpotential (V)

```c
AppLattice(spk,narg,argc)
{
  ninteger = 4; // 1) site type, 2) site grain id,
  ndouble = 2; // 3) coordination number, 4) deposition site indicator
  delpropensity = 1;
  delevent = 0;
  allow_kmc = 1;
  allow_rejection = 0;
  allow_masking = 0;
  numrandom = 1;
  dt_sweep = 1;
  seed_n = 0;
  count = 0;
  pulse = 0;
  on_dt = off_dt = prev_time = 0.0;
  i0 = 0.0;
  eta = 0.0;
  alpha_a = alpha_c = 0.0;
  ox_rate = 0.0;
  i_diss = i_dep = 0.0;
  n_hop = n_atomexch = n_step = n_grain = n_boundary = n_diss = 0;
  scale_dep_on = 1.0;
  scale_dep_off = 1.0;
  scale_diss_on = 1.0;
  scale_diss_off = 1.0;
  dep_sites_count = 0;
  diss_sites_count = 0;
  charge = 1.602176565e-19; // charge of an electron (C)
  if ((narg < 16) || (strcmp(name,"eam") != 0))
    error->all(FRERR,"Illegal app_style command");
```
seedlayer_size = atof(arg[1]);
seed_frac = atof(arg[2]);
size = atof(arg[3]);
seed_frac = atof(arg[4]);
n_max = atoi(arg[5]);
Ed = atof(arg[6]);
Ed_exch = atof(arg[7]);
Ed_step = atof(arg[8]);
v_d = atof(arg[9]);
v_exch = atof(arg[10]);
v_step = atof(arg[11]);
Ed_grain = atof(arg[12]);
Ed_boundary = atof(arg[13]);
v_grain = atof(arg[14]);
v_boundary = atof(arg[15]);

// set up plating conditions
if (strcmp(arg[3], "p") == 0)
{
if (narg < 21)
    error->all(FLERR, "Deposition parameters not specified");
eta = atof(arg[16]);
alpha_a = atof(arg[17]);
alpha_c = atof(arg[18]);
i0 = atof(arg[19]);
dep_mode = 1;
if (strcmp(arg[20], "dc") == 0)
{
pulse = 0;
pulse_on = 1;
}
if (strcmp(arg[20], "pp") == 0)
{
if (narg != 23)
    error->all(FLERR, "Pulse-plating conditions required");
pulse = 1;
pulse_on = 1;
on_dt = atof(arg[21]);
off_dt = atof(arg[22]);
}
if (strcmp(arg[20], "pr") == 0)
{
if (narg != 24)
    error->all(FLERR, "Pulse-plating conditions required");
pulse = 2;
pulse_on = 1;
on_dt = atof(arg[21]);
off_dt = atof(arg[22]);
ox_rate = atof(arg[23]);
}
else if (strcmp(arg[3], "g") == 0) {
  if (narg < 18)
    error->all(FLERR, "Deposition current density not specified");
  i_dep = atof(arg[16]);
  dep_mode = 0;
  if (strcmp(arg[17], "dc") == 0)
    pulse = 0;
  pulse_on = 1;
}

if (strcmp(arg[17], "pp") == 0) {
  if (narg != 20)
    error->all(FLERR, "Pulse-plating conditions required");
  pulse = 1;
  pulse_on = 1;
  on_dt = atof(arg[18]);
  off_dt = atof(arg[19]);
}

if (strcmp(arg[17], "pr") == 0)
  error->all(FLERR, "Pulse-reverse is unsupported in galvanostatic mode");

if ((seed_frac <= 0.) || (seed_frac > 1.0))
  error->all(FLERR, "Fraction must be between 0 and 1.");

AppEAM::~AppEAM ( ) [virtual]
{
}

Member Function Documentation

void AppEAM::check_grain_distance ( int grain ) [private]
Checks the distance between active sites in grain and active site of other grains to see if
the grains have met

77
**Parameters**

| grain | Grain ID that needs to be checked |

Checks the distance between active sites in grain and neighbors from other grains to see if the grains are in close proximity of each other. If they are, change the dep_flag of those sites in proximity to either -1 or -2 depending on the type. This will inhibit deposition at those sites and will help identify sites that can undergo grain boundary diffusion and grain boundary migration.

Computationally, it’s faster to flip dep_flag[i] to -1 than to compare the energy difference if deposition is to occur at site i. Sites that are less than the cutoff distance away from each other would have a repulsive energy contribution to the Hamiltonian and the event would have been unlikely (as in impossible) anyway.
if (!active
sites
check(j
local.first,j
local.second))
continue;
array_coordinates& coordinates_j = lattice[j
local.second]->coordinates[j
local.first];
x_j = coordinates_j[0];
y_j = coordinates_j[1];
z_j = coordinates_j[2];
dx = x_i - x_j;
dy = y_i - y_j;
dz = z_i - z_j;
if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
dx = fabs(dx) - 2.*xmid;
if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
dy = fabs(dy) - 2.*ymid;
if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
dz = fabs(dz) - 2.*zmid;
rsq = dx*dx + dy*dy + dz*dz;
if (rsq <= cutoff)
{
int& type_j = (lattice[j
local.second]->iarray[0])[j
local.first];
int& dep_flag_j = (lattice[j
local.second]->iarray[3])[j
local.first];
if (type_i[i] != type_j)
{
if (type_i[i] == 0)
{
dep_flag_i[i] = -1;
dep_flag_j = -2;
flag = 1; // if site i still is inhibited, flag = 1
}
else
{
dep_flag_j = -1;
dep_flag_i[i] = -2;
flag = 1;
}
else if ((type_i[i] == 1) && (type_j == 1))
{
dep_flag_i[i] = -2;
dep_flag_j = -2;
flag = 1; // if site i still is inhibited, flag = 1
}
}
else if (flag == 0)
dep_flag_i[i] = 0;

// if initially the site has reached the other grain but now has changed
// revert dep_flag back to 0
if (flag == 0)
dep_flag_i[i] = 0;
void AppEAM::check_grain_distance ( int site, int grain ) [private]

Checks the distance between active sites in grain and active site of other grains to see if
the grains have met

Parameters

<table>
<thead>
<tr>
<th>site</th>
<th>Local index of site that needs checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>grain</td>
<td>Grain ID that needs to be checked</td>
</tr>
</tbody>
</table>

Checks the distance between active sites in grain and neighbors from other grains to see
if the grains are in close proximity of each other. If they are, change the dep_flag of those
sites in proximity to either -1 or -2 depending on the type. This will inhibit deposition at
those sites and will help identify sites that can undergo grain boundary diffusion and grain
boundary migration.

if (!active_sites_check(site, grain))
  return;
int& type_i = (lattice[grain]->iarray[0])[site];
int& dep_flag_i = (lattice[grain]->iarray[3])[site];
vec_int& global_neighbors = lattice[grain]->global_neighbors[site];
int num_global_neighbors = lattice[grain]->num_global_neighbors[site];
double x_i, y_i, z_i, x_j, y_j, z_j, dx, dy, dz, rsq;
double cutoff = ((lattice[grain]->latconst)/1.01)*((lattice[grain]->latconst)/1.01);
double xmid = domain->midpoint(0);
double ymid = domain->midpoint(1);
double zmid = domain->midpoint(2);
array_coordinates& coordinates_i = lattice[grain]->coordinates[site];
x_i = coordinates_i[0];
y_i = coordinates_i[1];
z_i = coordinates_i[2];
int flag = 0; // flag indicating whether dep_flag needs to be reverted back from -1
// this distance is chosen to avoid large gaps between grains
for (int jj = 0; jj < num_global_neighbors; jj++)
  {
    int j = global_neighbors[jj];
pair_int j_local = global_to_grain(j);
if global neighbor is in the same grain, ignore
if (j_local.second == grain)
    continue;

if global neighbor not an active site of another grain, ignore
if (!active_sites_check(j_local.first, j_local.second))
    continue;

array_coordinates& coordinates_j = lattice[j_local.second]->coordinates[j_local.first];
x_j = coordinates_j[0];
y_j = coordinates_j[1];
z_j = coordinates_j[2];
dx = x_i - x_j;
dy = y_i - y_j;
dz = z_i - z_j;

if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
dx = fabs(dx) - 2.*xmid;
if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
    dy = fabs(dy) - 2.*ymid;
if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
    dz = fabs(dz) - 2.*zmid;
rsq = dx*dx + dy*dy + dz*dz;
if (rsq <= cutoff)
{
    int& type_j = (lattice[j_local.second]->iarray[0])[j_local.first];
    int& dep_flag_j = (lattice[j_local.second]->iarray[3])[j_local.first];
    // if sites are less than the cutoff distance away and one of them is
    // unoccupied, the unoccupied site is inhibited
    if (type_i != type_j)
    {
        if (type_i == 0)
        {
            dep_flag_i = -1;
            dep_flag_j = -2;
            flag = 1; // if site i still is inhibited, flag = 1
        }
        else
        {
            dep_flag_j = -1;
            dep_flag_i = -2;
            flag = 1; // if site i still is inhibited, flag = 1
        }
    }
    else if ((type_i == 1) && (type_j == 1))
    {
        dep_flag_i = -2;
        dep_flag_j = -2;
        flag = 1; // if site i still is inhibited, flag = 1
    }
void AppEAM::coord_update ( int i, int grain ) [private]

Updates the coordination number of site i

Parameters

<table>
<thead>
<tr>
<th>i</th>
<th>Local index of site i</th>
</tr>
</thead>
<tbody>
<tr>
<td>grain</td>
<td>Grain ID of site i</td>
</tr>
</tbody>
</table>

Updates coordination number for i’s neighbors, where the index i is a local (within the grain) index.

int AppEAM::determine_direction ( int i, int grain ) [private]

Determines the face that site i is closest to and which direction should be considered when determining whether step-edge and atom exchange moves for site i
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Local index of site $i$</td>
</tr>
<tr>
<td>$grain$</td>
<td>Grain ID of site $i$</td>
</tr>
</tbody>
</table>

Determines the face that site $i$ is closest to and which direction should be considered when determining whether step-edge and atom exchange moves for site $i$.

This is a hack that would not be necessary if we use graph theory to map the diffusion sites.

0 is $x-$, 1 is $x+$, 2 is $y-$, 3 is $y+$, 4 is $z-$, and 5 is $z+$ Implementation: Loop over all first nearest neighbors of $i$. If the neighbor is occupied, boxed in (coordnum > 6) or dep_flag = -1, ignore the site. Find $dx$, $dy$, and $dz$ from site $i$. For every time $dx$, $dy$, $dz$ != 0, add it to the corresponding element in the direction array.

Function will find the max in the array to determine which direction has the highest number of vacant neighbors - that direction is the direction that step-edge/atom exchange will move in.

Example: if loc = 5, this means that site $i$ may undergo step-edge/atom exchange in the $z$ direction on the plane that is normal to the $z+$ direction.

```cpp
vec_int& dep_flag = lattice[grain]->iarray[3];
double xmid = domain->midpoint(0);
double ymid = domain->midpoint(1);
double zmid = domain->midpoint(2);
vec_int direction(6,0); // create 6x1 vector and initialize all values to zero
double x_loc, y_loc, z_loc;
double jx_loc, jy_loc, jz_loc;
double dx, dy, dz;
vec_coordinates& coordinates = lattice[grain]->coordinates;
vec_int& neighbors = lattice[grain]->neighbors;
vec_int& num_neighbors = lattice[grain]->num_neighbors;
vec_int& coordnum = lattice[grain]->iarray[2];
vec_int& type = lattice[grain]->iarray[0];

// get local coordinates of site i
x_loc = get_local_coordinates(coordinates[i], grain, 0);
y_loc = get_local_coordinates(coordinates[i], grain, 1);
z_loc = get_local_coordinates(coordinates[i], grain, 2);
```

83
int j = neighbors[i][jj];

// if site is not vacant, skip
if (type[j] != 0)
    continue;

// if dep_flag = -1 (vacant, on a grain boundary) - skip since we don't want
// sites in the grain boundary to undergo atom exchange
if (dep_flag[j] == -1)
    continue;

// if bulk, ignore
if (coordnum[j] > 6)
    continue;

// get j's local coordinates
j_x_loc = get_local_coordinates(coordinates[j], grain, 0);
j_y_loc = get_local_coordinates(coordinates[j], grain, 1);
j_z_loc = get_local_coordinates(coordinates[j], grain, 2);

// get the distance away from site i in terms of local coordinates
dx = x_loc - j_x_loc;
dy = y_loc - j_y_loc;
dz = z_loc - j_z_loc;

// if periodic boundary condition is used, make sure dx, dy, and dz are correct
if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
    dx = fabs(dx) - 2.*xmid;
if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
    dy = fabs(dy) - 2.*ymid;
if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
    dz = fabs(dz) - 2.*zmid;

// every time dx, dy, dz is not zero, add it to the count in direction array
// store them in the reverse order so that when we use max_element and there's
// a case where two elements are equal, it will favor the z-direction
if ((dx > 0.) && (fabs(dx) > 0.001))
    direction[5]++;
if ((dx < 0.) && (fabs(dx) > 0.001))
    direction[4]++;
if ((dy > 0.) && (fabs(dy) > 0.001))
    direction[3]++;
if ((dy < 0.) && (fabs(dy) > 0.001))
    direction[2]++;
if ((dz > 0.) && (fabs(dz) > 0.001))
    direction[1]++;
if ((dz < 0.) && (fabs(dz) > 0.001))
    direction[0]++;
int sum = std::accumulate(direction.begin(), direction.end(), 0);

// find where the max element in direction vector is (max_element returns an iterator)
int loc = std::distance(direction.begin(), std::max_element(direction.begin(), direction.end()));

loc = 5 - loc; // converting back to x->z ordering

// if there are no sites that satisfy the previous criteria, site i is at the grain boundary
if (sum == 0)
    loc = -1;

return loc;

---

int AppEAM::edge_check ( int i, int grain ) [private]

Checks whether site i is at the edge of the simulation domain

Parameters

<table>
<thead>
<tr>
<th>i</th>
<th>Local index of site i</th>
</tr>
</thead>
<tbody>
<tr>
<td>grain</td>
<td>Grain ID of site i</td>
</tr>
</tbody>
</table>

Checks if site is at the edge of the box. This is based on the coordination number of the edge site being lower in non-periodic BCs.

vec_int& coordnum = lattice[grain]->iarray[2];
vec_coordinates& coordinates = lattice[grain]->coordinates;
double x = coordinates[i][0];
double y = coordinates[i][1];
double x_max = domain->boxhi[0];
double y_max = domain->boxhi[1];

// max distance away from the box boundary to be considered an edge site
double max_dist = (lattice[grain]->next_site)*(lattice[grain]->latconst);

// if the site is at the edge of the box at the x-axis and is non-periodic in x
if (x <= max_dist) || (fabs(x - x_max) <= max_dist) && (domain->periodicity[0] == 0)
    { if (coordnum[i] >= 5) // coordination number of edge sites in the bulk >= 4
        return 1;
        else if (fabs(y - y_max) <= max_dist) && (coordnum[i] >= 3))
        // unless it is at the corner, in that case coordnum >= 3
int AppEAM::exchange_destination ( int n, int grain, int direction )
[private]

Check if site n is a possible destination for step-edge atom exchange and atom exchange in some direction.

Parameters

<table>
<thead>
<tr>
<th>n</th>
<th>Local index of site n</th>
</tr>
</thead>
<tbody>
<tr>
<td>grain</td>
<td>Grain ID of site n</td>
</tr>
<tr>
<td>direction</td>
<td>Direction that step-edge is occurring</td>
</tr>
</tbody>
</table>

Check if site n is a possible destination for step-edge atom exchange and atom exchange in some direction. Function returns 0 or 1 depending on whether it is a possible exchange site or not. Criteria is that site n’s 4 nearest neighbors in the layer below it in the plane in which step-edge occurs have to be occupied. This is because step-edge is the movement from one terrace down to the layer below it. The layer below is a fully occupied layer because otherwise the atom will be hanging out from the terrace.
// if coordination number is not greater than 4 and not at the edge, the site cannot
// be a potential destination for step-edge atom exchange

if ((coordnum[n] < 5) && (coordinates[n][floor(direction*0.5)]) >=
    (lattice[grain]->next_site)*((lattice[grain]->latconst)/1.01))
    return 0;

vec<int> neighbors = lattice[grain]->neighbors[n];
int num_neighbors = lattice[grain]->num_neighbors[n];

double dx, dy, dz;
int neigh_count = 0;

double xmid = domain->midpoint(0);
double ymid = domain->midpoint(1);
double zmid = domain->midpoint(2);

for (int jj = 0; jj < num_neighbors; jj++)
{
    int j = neighbors[jj];
    if (direction == 0)
    {
        dx = coordinates[n][0] - coordinates[j][0];
        if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
            dx = fabs(dx) - 2.*xmid;
        if ((dx < 0.) && (fabs(dx) > 0.001) && (type[j] == 1))
            neigh_count++;
    }
    else if (direction == 1)
    {
        dx = coordinates[n][0] - coordinates[j][0];
        if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
            dx = fabs(dx) - 2.*xmid;
        if ((dx > 0.) && (fabs(dx) > 0.001) && (type[j] == 1))
            neigh_count++;
    }
    else if (direction == 2)
    {
        dy = coordinates[n][1] - coordinates[j][1];
        if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
            dy = fabs(dy) - 2.*ymid;
        if ((dy < 0.) && (fabs(dy) > 0.001) && (type[j] == 1))
            neigh_count++;
    }
    else if (direction == 3)
    {
        dy = coordinates[n][1] - coordinates[j][1];
        if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
            dy = fabs(dy) - 2.*ymid;
        if ((dy > 0.) && (fabs(dy) > 0.001) && (type[j] == 1))
            neigh_count++;
    }
}
if ((dy > 0.) && (fabs(dy) > 0.001) && (type[j] == 1))
neigh_count++;

else if (direction == 4)
{
dz = coordinates[n][2] - coordinates[j][2];

if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
dz = fabs(dz) - 2.*zmid;

if ((dz < 0.) && (fabs(dz) > 0.001) && (type[j] == 1))
neigh_count++;

else if (direction == 5)
{
dz = coordinates[n][2] - coordinates[j][2];

if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
dz = fabs(dz) - 2.*zmid;

if ((dz > 0.) && (fabs(dz) > 0.001) && (type[j] == 1))
neigh_count++;

} // if the 4 nearest neighbors above site n are vacant, return 1
if (neigh_count == 4)
return 1;

return 0;

} // if ((dy > 0.) && (fabs(dy) > 0.001) && (type[j] == 1))


void AppEAM::gen_seed_layer ( class RandomPark ∗ random, vec_int seedlayer ) [private]

Generates a seed layer to initialize the simulation

Parameters

| random | Random number generator |
| seedlayer | Vector of sites in the seed layer |

Generates the seed layer for deposition to occur. App will not work without a seed layer since the propensity will be zero for all sites.

Two modes are possible: i) seed layer occupancy fraction is not 1 - sites are randomly assigned or deposition is on a foreign substrate and ii) seed layer occupancy fraction is 1 - polycrystalline substrate will be randomly generated.

00564 {
pair_int site_local;

int seed_count = 0; // counter for number of sites occupied in seed layer
int ind = 0;
double einitial;
double efinal;
int flag;

if (seed_frac != 1.0)
// if the desired fraction is not 1, randomly assign atoms to sites in the seed layer
{
    double xmid = domain->midpoint(0);
double ymid = domain->midpoint(1);
double zmid = domain->midpoint(2);

double dx, dy, dz, rsq;
double cutoff = ((lattice[0]->latconst)/1.01)*((lattice[0]->latconst)/1.01);
while (seed_count <= seed_n)
{
    // select site using random number
    ind = int (random->uniform()*seedlayer.size());
    if (ind == seedlayer.size())
        ind--;
    flag = 0;
    site_local = global_to_grain(seedlayer[ind]);

    int& type_i = (lattice[site_local.second]->iarray[0])[site_local.first];
    vec_int& num_global_neighbors = lattice[site_local.second]->num_global_neighbors;
    vec_vec_int& global_neighbors = lattice[site_local.second]->global_neighbors;

    array_coordinates& coordinates_site = lattice[site_local.second]->coordinates[site_local.first];

    // check global neighbors within cutoff to prevent overlap
    for (int i = 0; i < num_global_neighbors[site_local.first]; i++)
    {
        int j = global_neighbors[site_local.first][i];
        pair_int j_local = global_to_grain(j);

        array_coordinates coordinates_j = lattice[j_local.second]->coordinates[j_local.first];

        dx = coordinates_site[0] - coordinates_j[0];
dy = coordinates_site[1] - coordinates_j[1];
dz = coordinates_site[2] - coordinates_j[2];

        if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
dx = fabs(dx) - 2.*xmid;
if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
dy = fabs(dy) - 2.*ymid;
if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
dz = fabs(dz) - 2.*zmid;

    }
}

}
\(r^2 = dx^2 + dy^2 + dz^2;\)

```cpp
if (r^2 >= cutoff)
    continue;
```

```cpp
int& typej = (lattice[j_local.second]->iarray[0])[j_local.first];
```

```cpp
// if one of the site's neighbors from a different grain is occupied
// this site has to be unoccupied (same idea as check_grain_distance)
if ((typej == 1) && (site_local.second != j_local.second))
    flag = 1; // flip the flag to prevent deposition
}
```

```cpp
// if no issues, try depositing a site here
if (flag == 0)
{
    e_initial = system_energy(site_local.first,-1,0,site_local.second,site_local.second);
    // change type to occupied
    typei = 1;
    e_final = system_energy(site_local.first,-1,1,site_local.second,site_local.second);
    // check energy
    if (e_final <= e_initial)
    {
        seed_count++;
        e_final = system_energy(site_local.first,-1,2,site_local.second,site_local.second);
    }
    else // else, revert type back
    {
        typei = 0;
    }
}
```

```cpp
else // 'grow' the seed up to seedlayer.size then polish it back down to 0.75*height
```

```cpp
int seed_sites = lattice_count*2; // get the number of seed sites
int seed_sites_count = 0;
```

```cpp
while (seed_sites_count < seed_sites)
```

```cpp
    ind = int (random->uniform()*seedlayer.size()); // pick a site
    if (ind == seedlayer.size())
        // making sure that the index will not exceed the dim. of array
    ind--;
```

```cpp
    site_local = global_to_grain(seedlayer[ind]);
```
array_coordinates i = coordinates[site_local.second] - > coordinates[site_local.first];

type i = (lattice[site_local.second] -> iarray[0])[site_local.first];

// if the site is at the bottom z limit of box, plant the seed
if ((coordinates[i][2] <= (lattice[site_local.second] -> next_site) * (lattice[site_local.second] -> latconst)/1.01)) &&

(type i == next_grain)){

type i = 1;

seed_sites.push_back(seedlayer[ind]); // add to seed sites
seedlayer.erase(seedlayer.begin() + ind); // remove from seedlayer
coord_update(site_local.first, site_local.second); // update coordnum

// add to active sites
active_sites[site_local.second].push_back(site_local.first);

vec_int num_neighbors = lattice[site_local.second] -> num_neighbors;
vec_vec_int neighbors = lattice[site_local.second] -> neighbors;

check_grain_distance(site_local.first, site_local.second);

// add the nearest neighbors to active_sites as well
for (int jj = 0; jj < num_neighbors[site_local.first]; jj++)
{
    int j = neighbors[site_local.first][jj];
    active_sites[site_local.second].push_back(j);
    check_grain_distance(j, site_local.second);
}

// update dep_flag
update_dep_flag(site_local.second);

next_grain ++; // move on to next grain

// if looped over all grains, start over
if (next_grain == lattice_count)

next_grain = 0;

} // now fill in the rest based on the seed sites

// let it loop over occ_sites.size() - this will grow as the number of occ_sites
// increases

int k = 0;
while (k < occ_sites.size())
{
    int site = occ_sites[k];

    site_local = global_to_grain(site);

    num_neighbors = lattice[site_local.second] -> num_neighbors;
    neighbors = lattice[site_local.second] -> neighbors;

    array_coordinates i = coordinates[site_local.second] - > coordinates[site_local.first];
// loop over all first nearest neighbors
for (int j = 0; j < num_neighbors[site_local.first]; j++)
{
    // get global index to find site in seedlayer
    int neigh_global = grain_to_global(std::make_pair(neighbors[site_local.first][j], site_local.second));
    auto ind = std::find(seedlayer.begin(), seedlayer.end(), neigh_global);
    // if the neighbor is in the seedlayer vector, it’s a seed site
    if (ind != seedlayer.end())
    {
        site_local = global_to_grain(*ind);
        int dep_flag_i = lattice[site_local.second]->iarray[3][site_local.first];
        // check dep_flag to prevent overlaps and to avoid computing the energy
        // (sites that will cause efinal > einitial are excluded by dep_flag)
        if (dep_flag_i != 1)
            continue;
        einitial = system_energy(site_local.first,-1,0,site_local.second,site_local.second);
        // change type to 1
        lattice[site_local.second]->iarray[0][site_local.first] = 1;
        efinal = system_energy(site_local.first,-1,1,site_local.second,site_local.second);
        if (efinal > einitial)
            { lattice[site_local.second]->iarray[0][site_local.first] = 0;
              seedlayer.erase(ind);
              continue; }
    }
    occ_sites.push_back(*ind);
    seedlayer.erase(ind);
    coord_update(site_local.first,site_local.second);
    // if it was not in active_sites, add it
    if (!active_sites_check(site_local.first, site_local.second))
        active_sites[site_local.second].push_back(site_local.first);
    check_grain_distance(site_local.first,site_local.second);
    for (int jj = 0; jj < num_neighbors[site_local.first]; jj++)
    { int j = neighbors[site_local.first][jj];
      if (!active_sites_check(j, site_local.second))
      { active_sites[site_local.second].push_back(j);
        check_grain_distance(j,site_local.second);
      }
    }
    update_dep_flag(site_local.second);
// now that we have the substrate, we 'polish' it
for (int i = 0; i < occ_sites.size(); i++)
{
    site_local = global_to_grain(occ_sites[i]);
    array_coordinates& coordinates_i = lattice[site_local.second]->coordinates[site_local.first];
    if (coordinates_i[2] > polish_height)
        lattice[site_local.second]->array[0][site_local.first] = 0;
    coord_update(site_local.first,site_local.second);
}
seed_n = occ_sites.size();

double AppEAM::get_local_coordinates( array_coordinates & coord, int grain, int direction ) [private]

Calculates the local coordinates of a site
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>Coordinates of site</td>
</tr>
<tr>
<td>grain</td>
<td>Grain ID of site</td>
</tr>
<tr>
<td>direction</td>
<td>Direction (x, y, or z) of the local coordinates to return</td>
</tr>
</tbody>
</table>

Calculates the local 'direction'-coordinate of site.

matrix& inv_rot_mat = lattice[grain]->inv_rot_mat;
array_coordinates& trans_mat = lattice[grain]->trans_mat;
double local_coord;

if (direction == 0)
    local_coord = inv_rot_mat[0][0]*coord[0] - trans_mat[0] + inv_rot_mat[0][1]*coord[1] - trans_mat[1] + inv_rot_mat[0][2]*coord[2] - trans_mat[2];
else if (direction == 1)
else if (direction == 2)
else
    error->all(FLERR, "Invalid direction");
return local_coord;
virtual void SPPARKS_NS::AppEAM::grow_app ( ) [inline], [virtual]

void AppEAM::init_app ( ) [virtual]

Initialize the application before each run.

The function will initialize the EAM potential, calculate the current densities of deposition and dissolution, generate seed layer, and determine coordination number, dep_flag, etc. seed_n = specified fraction * number of sites in seed layer, rounded up to nearest integer

00247 {  
00248 // initialize potential  
00249 potential->init();  
00250 pair = (PairEAM*)potential->pair;  
00251 // if potentiostatic, apply Butler-Volmer to get the current densities  
00252 if (dep_mode == 1)  
00253 {  
00254     l dep = l0*exp(-1. * alpha_c * eta * t_inverse);  
00255     l diss = l0*exp(alpha_a * eta * t_inverse);  
00256  
00257     if (pulse == 2)  
00258         {  
00259             la dep = l0*exp(-1. * alpha_c * ox_rate * t_inverse);  
00260             la diss = l0*exp(alpha_a * ox_rate * t_inverse);  
00261         }  
00262     }  
00263     // vector of vector of doubles containing the propensity contribution of each  
00264     // dep/diss event for each site  
00265     dep_diss_prob.resize(N_total, vec_double(4, 0.));  
00266     // dep/diss event for each site  
00267     dep_diss_prob.resize(N_total, vec_double(4, 0.));  
00268     // sum of dep/diss flags of each grain  
00269     dep_sites_count_grain.resize(lattice_count, 0);  
00270     diss_sites_count_grain.resize(lattice_count, 0);  

94
vec_int seedlayer;

// find sites in seed layer
for (int i = 0; i < lattice_count; i++)
{
    int& N = lattice[i]->N;
    vec_coordinates& coordinates = lattice[i]->coordinates;
    for (int j = 0; j < N; j++)
    {
        if (coordinates[j][2] < seedlayer.size)
            i_local = std::make_pair(j, i);
        seedlayer.push_back(grain_to_global(i_local));
    }
}

seedn = static_cast<int> (ceil(seedfrac*seedlayer.size())); // number of occupied sites in the seed layer
polish_height = 0.75*seedlayer.size;
if (seedn > Ntotal)
    error->all(FPERR,"Specified density exceeds number of sites");
if (seedn > 0)
gen_seedlayer(ranapp, seedlayer);
std::cout << "Seeded " << seedn << " sites." << std::endl;

// initializes coordination number, dep_flags, and active_sites
for (int k = 0; k < lattice_count; k++)
{
    int& N = lattice[k]->N;
    vec_int& num_neighbors = lattice[k]->num_neighbors;
    vec_vec_int& neighbors = lattice[k]->neighbors;
    vec_int& coordnum = lattice[k]->iarray[2];
    vec_int& dep_flag = lattice[k]->iarray[3];
    vec_coordinates& coordinates = lattice[k]->coordinates;
    vec_int& grainid = lattice[k]->iarray[1];
    for (int i = 0; i < N; i++)
    {
        coord_update(i, k);
        // identify the edges of the active site
        if ((coordnum[i] != 12) || (coordnum[i] != 0))
            active_sites[k].push_back(i);
    }
if ((seed == 0) && (coordinates[i][2] < (lattice[k]->next site)*((lattice[k]->latconst)/1.01)))
    active_sites[k].push_back(i);
    grain_id[i] = k; // assign grain id
    dep_flag[i] = 0; // clear out dep_flag
}

// check whether grain needs growing
for (int k = 0; k < lattice_count; k++)
{
    vec<int> type = lattice[k]->iarray[0];
    // check the grain (only check occupied active sites)
    for (int i = 0; i < active_sites[k].size(); i++)
    {
        if (type[active_sites[k][i]] == 1)
            update_status(i,k,0);
    }
}

// check the distance between grains and flag any sites that are in close proximity
for (int k = 0; k < lattice_count; k++)
    check_grain_distance(k);

dep_sites_count = 0; // reset dep_sites_count - it might have been changed by gen_seed_layer
diss_sites_count = 0;

// update dep_flag
for (int k = 0; k < lattice_count; k++)
{
    dep_sites_count_grain[k] = 0; // clear out grain count
    diss_sites_count_grain[k] = 0; // clear out grain count
    update_dep_flag(k); // now update dep_flag
}

int AppEAM::neighbor_check ( int i, int j, int grain ) [private]

Checks whether site j is a 2nd nearest neighbor of site i

Parameters

| i | Local index of site i |
Check if \( j \) is already part of \( \text{neighbors2}[i] \)

```cpp
00813 {
00814    vec_int num_neighbors2 = lattice[grain]->num_neighbors2;
00815    vec_vec_int neighbors2 = lattice[grain]->neighbors2;
00816    if (i == j)
00817        return 1;
00818    if (j == -1)
00819        return 1;
00820    if (num_neighbors2[i] == 0)
00821        return 0;
00822    // using stl to do the search instead of for loop
00823    auto loc = std::find(neighbors2[i].begin(), neighbors2[i].end(), j);
00824    if (loc != neighbors2[i].end())
00825        return 1;
00826    // lattice site not in 2nd nearest neighbors list
00827    return 0;
00828 }
00829
```

**void AppEAM::scale_propensities() [private]**

Scales the deposition propensities such that the average = propensity with \( i \text{dep} \)

Calculates the average propensity for each event based on the propensities stored in \( i \text{dep}, i \text{diss}, i \text{prob} \). Divide that average by the desired average propensity based on the current density. The quotient of that is the scaling factor. Update the scaling factor and apply the change to all sites in question.

```cpp
00410 {
00411    diss_sites.clear(); // clear old data
00412    dep_sites.clear();
00413    double sum = 0.0;
00414    double scale = 1.0;
00415    double area = (domain->length(0))*(domain->length(1));
00416    // find the deposition and dissolution sites whose propensity is > 1e-10
00417    // loop over active sites in each grain
00418    for (int j = 0; j < lattice_count; j++)
00419    {
00420        for (int ii = 0; ii < active_sites[j].size(); ii++)
00421```
{  
    int i = active_sites[j][ii];  
    int k = grain_to_global(std::make_pair(i,j));  
    // find all sites that can undergo deposition  
    if ((lattice[j]->iarray[3][i] == 1) && (lattice[j]->iarray[0][i] == 0)  
        && (propensity[k] > 1e-10))  
        dep_sites.push_back(k);  
    // now all sites that can undergo dissolution  
    else if ((lattice[j]->iarray[3][i] == 2) && (lattice[j]->iarray[0][i] == 1)  
        && (propensity[k] > 1e-10))  
        diss_sites.push_back(k);  
}

// start with deposition  
for (int i = 0; i < dep_sites.size(); i++)  
    {  
        // remove the contribution from total propensity for now, will add scaled value back to it  
        propensity[dep_sites[i]] -= dep_diss_prob[dep_sites[i]][0];  
        sum += dep_diss_prob[dep_sites[i]][0]; // add it to sum  
    }

// find numx - area is in Angstroms^2, so we need to convert to nm^2  
double numx = dep_sites_count*100./area;  
// i dep is in pA/nm^2, 1e-12 is to convert it to A/nm^2  
double avg = i.dep * 1.e-12/ (zme*charge*numx); // desired average propensity  
sum = sum / dep_sites.size(); // divide by dep_sites.size() to get 'average' propensity  
if (sum < 1e-15) // to prevent divide by zero error  
    scale = 1.;  
else  
    scale = avg/sum; // get the new scaling parameter  
scale_dep_on = scale; // apply that to existing scaling parameter  
for (int i = 0; i < dep_sites.size(); i++)  
    {  
        dep_diss_prob[dep_sites[i]][0] *= scale; // apply the scale to old probs  
        propensity[dep_sites[i]] = dep_diss_prob[dep_sites[i]][0]; // add back to propensity  
    }

// now dep rate when turned off - only in pulse-reverse  
if (pulse == 2)  
    {  
        sum = 0.; // zero out sum  
        for (int i = 0; i < dep_sites.size(); i++)  
            {  
                propensity[dep_sites[i]] -= dep_diss_prob[dep_sites[i]][1];  
                sum += dep_diss_prob[dep_sites[i]][1];  
            }  
        avg = i.dep * 1.e-12/ (zme*charge*numx); // desired average propensity  
        sum = sum / dep_sites.size(); // divide by dep_sites.size() to get 'average' propensity  
    }
if (sum < 1e-15) // to prevent divide by zero error
    scale = 1.;
else
    scale = avg/sum; // get the new scaling parameter
scale_dep_off *= scale;
for (int i = 0; i < dep_sites.size(); i++)
    dep_diss_prob[dep_sites[i]][1] *= scale; // apply the scaling factor
    propensity[dep_sites[i]] += dep_diss_prob[dep_sites[i]][1]; // add back to propensity
}
solve->update(dep_sites, propensity); // update the solver

// now for dissolution events - first check if there are any sites that fit the criteria
if (diss_sites.size() != 0)
{
    sum = 0.0; // clear out sum
    for (int i = 0; i < diss_sites.size(); i++)
        propensity[diss_sites[i]] -= dep_diss_prob[diss_sites[i]][2];
    sum += dep_diss_prob[diss_sites[i]][2];
}

numx = diss_sites_count*100./area; // now with diss_sites_count for dissolution
avg = i_diss * 1.e-12/ (z_me*charge*numx); // desired average propensity
sum = sum / diss_sites.size(); // divide by diss_sites.size() to get 'average' propensity
if (sum < 1e-15)
    scale = 1.;
else
    scale = avg/sum; // get the new scaling parameter
scale_diss_on *= scale;
for (int i = 0; i < diss_sites.size(); i++)
    dep_diss_prob[diss_sites[i]][2] *= scale; // apply the scaling factor
    propensity[diss_sites[i]] += dep_diss_prob[diss_sites[i]][2]; // add back to propensity

// diss rate when off -only in PR
if (pulse == 2)
{
    sum = 0.;
    for (int i = 0; i < diss_sites.size(); i++)
        propensity[diss_sites[i]] -= dep_diss_prob[diss_sites[i]][3];
    sum += dep_diss_prob[diss_sites[i]][3];
}

avg = i_s_diss * 1.e-12/ (z_me*charge*numx); // desired average propensity
sum = sum / diss_sites.size(); // divide by diss_sites.size() to get 'average' propensity
if (sum < 1e-15)
    scale = 1.;
else
    scale = avg/sum; // get the new scaling parameter
scale_diss_off *= scale;
for (int i = 0; i < diss_sites.size(); i++)
    dep_diss_prob[diss_sites[i]][3] *= scale; // apply the scaling factor
    propensity[diss_sites[i]] += dep_diss_prob[diss_sites[i]][3]; // add back to propensity
solve->update(diss_sites, propensity); // update solver

void AppEAM::setup_app ( ) [virtual]

Set up the application.

Calculate the potential at each site before the propensities are computed.

max_eng = -3.54;

void AppEAM::setup_end_app ( ) [virtual]

Set up the application after propensity is computed.

Scale the propensities before application starts
double AppEAM::site_energy ( int i, int grain ) [virtual]

Computes the energy of site i

```cpp
vec_int& type = lattice[grain]->iarray[0];
vec_double& rho = lattice[grain]->darray[0];
vec_double& phi = lattice[grain]->darray[1];
vec_coordinates& coordinates = lattice[grain]->coordinates;
vec_int num_global_neighbors = lattice[grain]->num_global_neighbors;
vec_vec_int global_neighbors = lattice[grain]->global_neighbors;
if (type[i] != 0)
    return pair->energy(i, grain, num_global_neighbors, global_neighbors, coordinates, type, rho, phi);
else
    return 0.;
```

void AppEAM::site_event ( int i, class RandomPark * random ) [virtual]

KMC method - choose and perform an event for site. See comments in site_propensity for explanation of how events are identified. energy after proposed move

```cpp
if (propensity[i] == 0.)
    { naccept--; return; }
if ((pulse != 0) && (pulse_on == 1) && ((time - prev_time) > on_dt))
    { pulse_on = 0;
      prev_time = time;
    }
else if ((pulse != 0) && (pulse_on == 0) && ((time - prev_time) > off_dt))
    { pulse_on = 1;
      prev_time = time;
    }
pair_int i_local = global_to_grain(i);
int jj;
vec_int& type = lattice[i_local.second]->iarray[0];
```
vec_int& coordnum = lattice[i_local.second]->iarray[2];
vec_int& dep_flag = lattice[i_local.second]->iarray[3];

vec_coordinates& coordinates = lattice[i_local.second]->coordinates;
vec_int& num_neighbors = lattice[i_local.second]->num_neighbors;
vec_vec_int& neighbors = lattice[i_local.second]->neighbors;
vec_int& num_global_neighbors = lattice[i_local.second]->num_global_neighbors;
vec_vec_int& global_neighbors = lattice[i_local.second]->global_neighbors;
vec_int& num_neighbors2 = lattice[i_local.second]->num_neighbors2;
vec_vec_int& neighbors2 = lattice[i_local.second]->neighbors2;

double xmid = domain->midpoint(0);
double ymid = domain->midpoint(1);
double zmid = domain->midpoint(2);

sites.clear();

sites.clear();

sites.clear();
int diff_count = 0;

int j = i_local.second;
int flag = -1;

int flag = 0; // flag to indicate diffusion across a grain (=1)
int step_flag = 0; // flag to indicate whether step-edge atom exchange is possible or not
double prob = 0.0;
double efinal;

diff_sites.clear();

// zero out diff_count
i = i_local.first;

// we only need to update the status of the site and the system energy
if (type[i] == 0)
{
    if (count >= n_max)
    {
        naccept--;
        return;
    }

    if ((dep_flag[i] == 1) && (pulse_on == 1))
    {
        einitial = system_energy(i, j, 0, i_local.second, i_local.second);
        type[i] = 1;

        // find surface area in (Angstroms)^2
        double area = (domain->length(0)) * (domain->length(1));
02440  // possible dep sites per unit area (sites/nm^2)
02441  double numx = dep_sites_count*100./area;
02442
02443  efinal = system_energy(i,j,1,i_local.second,i_local.second);
02444
02445  // Budevski et al. p.28
02446  prob += i_dep * 1.e-12 *(efinal - einitial) * scale_dep_on/ (z_me*charge*numx*max_eng);
02447  if (prob >= threshold)
02448      { // need to run system_energy to calculate the energy after the move
02449            // otherwise, the energy will not be updated
02450            efinal = system_energy(i,j,2,i_local.second,i_local.second);
02451            count++;
02452            grain_j = i_local.second;
02453            goto update;
02454      }
02455  }  // deposition during anodic part of pulse-reverse
02456  else if ((dep_flag[i] == 1) && (pulse == 2) && (pulse_on == 0))
02457      {
02458        einitial = system_energy(i,j,0,i_local.second,i_local.second);
02459        type[i] = 1;
02460        // find surface area in (Angstroms)^2
02461        double area = (domain->length(0))*(domain->length(1));
02462
02463        // possible dep sites per unit area (sites/nm^2)
02464        double numx = dep_sites_count*100./area;
02465        efinal = system_energy(i,j,1,i_local.second,i_local.second);
02466
02467        // Budevski et al. p.28
02468        prob += i_a_dep * 1.e-12 *(efinal - einitial) * scale_dep_off / (z_me*charge*numx*max_eng);
02469        if (prob >= threshold)
02470            { // need to run system_energy to calculate the energy after the move
02471            // otherwise, the energy will not be updated
02472            efinal = system_energy(i,j,2,i_local.second,i_local.second);
02473            count++;
02474            grain_j = i_local.second;
02475            goto update;
02476      }
02477  }  // deposition during anodic part of pulse-reverse
02478  else
02479      {
02480        type[i] = 0;
02481        naccept--;  // deposition during anodic part of pulse-reverse
02482        return;
02483      }
naccept--; 
return;
} }

// if neither are satisfied 
naccept--; return;

// if the event is diffusion, have to loop over all possible diffusion sites 
else {
// special case of diffusion - grain boundary migration 
// the event will affect sites in 2 different grains 
for (int kk = 0; kk < num_global_neighbors[i]; kk++)

int k = global_neighbors[i][kk];
pair_int k_local = global_to_grain(k);

// if they are in the same grain, ignore it 
if (k.second == i.second)
continue;

// we have already identified possible destination sites using dep_flag 
// criteria of assigning -1 to dep_flag 
vec_int& dep_flag_k = lattice[k.second]->iarray[3];

if (dep_flag_k[k.first] != -1)
continue;

vec_int& type_k = lattice[k.second]->iarray[0];
einitial = system_energy(i, k.first, 0, i.second, k.second); 
type[i] = 0;
type[k.first] = 1;
efinal = system_energy(i, k.first, 1, i.second, k.second); 
if (efinal <= einitial) 
prob += vgrain*exp(-Ed_grain*t_inverse);
else if (efinal > einitial, this event is impossible 
prob += vgrain*exp(-(Ed_grain + efinal - einitial)*t_inverse);

if (prob >= threshold) 
{ 
efinal = system_energy(i, k.first, 2, i.second, k.second); 

j = k.first;
grain_j = k.second; 
n_grain++; 
flag = 1;
// flip the dep_flag of i and k as well since they swap states
dep_flag[i] = -1;
derp_flag[k] = -2;
break;
}
else
{
type_k = 0;
type[i] = 1;
j = -1;
}

if (j != -1)
    goto update;

if (coordinates[i][2] < (lattice[i_local.second]->next_site) *
    ((lattice[i_local.second]->latconst)/1.01))
{
    naccept--;
    return;
}
dissolution when not pulse-reverse
if ((dep_flag[i] == 2) && (pulse_on == 1) && (count < n_max))
{
einitial = system_energy(i,j,0,i_local.second,i_local.second);
type[i] = 0;
// get efinal without permanently changing the energy
efinal = system_energy(i,j,1,i_local.second,i_local.second);
double area = (domain->length(0)) * (domain->length(1)); // find surface area in (Angstroms)^2
numx = diss_sites_count*100./area; // possible dep sites per unit area
 double prob += i_diss * 1.e-12 * max_eng*scale_diss / (zme*charge*numx*(einitial-efinal));
prob += i_diss * 1.e-12 / (zme*charge*numx);
if (prob >= threshold)
{
    // need to run system_energy to calculate the energy after the move
    efinal = system_energy(i,j,2,i_local.second,i_local.second); // energy after proposed move
    n_diss++;
    grain_j = i_local.second;
goto update;
}
else
    type[i] = 1;
if ((dep_flag[i] == 2) && (pulse_on == 0) && (pulse == 2) && (count < n_max)) {
    enitial = system_energy(i, j, 0, i_local.second, i_local.second);
    type[i] = 0;
    // get efinal without permanently changing the energy
    efinal = system_energy(i, j, 1, i_local.second, i_local.second);
    double area = (domain->length(0)) * (domain->length(1)); // find surface area in (Angstroms)^2
    double numx = diss_sites_count*100./area; // possible dep sites per unit area (sites/nm^2)
    prob += i_a_diss * 1.e-12 * max_eng * scale_diss_off / (zme*charge*numx*(enitial-efinal)); // convert i_dep to A/nm^2 and calculate propensity
    if (prob >= threshold) {
        // need to run system_energy to calculate the energy after the move
        // otherwise, the energy will not be updated
        efinal = system_energy(i, j, 2, i_local.second, i_local.second);
        n_diss++;
        grain_j = i_local.second;
        goto update;
    } else
        type[i] = 1;
} else
    if (coordnum[i] > 6) { naccept--; return; }

if (edge_check(i, i_local.second)) { naccept--; return; }

// loop over first nearest neighbors to see if there’s any neighbor that has dep_flag = -1 (close to or overlapping with another grain)
// if there is, change step_flag to -1 to prevent step-edge atom exch
for (int kk = 0; kk < num_neighbors[i]; kk++) { int k = neighbors[i][kk];
    if (dep_flag[k] == -1) step_flag = 1;
// get the direction that the local coordinate of i and j will be compared
int direction = determine_direction(i, i.local.second);
int dir;
double local_coord_i, local_coord_k, local_coord_n;

if (direction != -1)
{
  dir = static_cast<int> (floor(direction*0.5));
  local_coord_i = get_local_coordinates(coordinates[i], i.local.second, dir);
  // if we're dealing with lower limit, multiply the coordinate by -1 to get the
  // right comparison
  if (direction % 2 == 0)
    local_coord_i *= -1;
}

double dx, dy, dz, rsq;
double cutoff = (lattice[i.local.second]->latconst)*(lattice[i.local.second]->latconst);

for (int kk = 0; kk < num_global_neighbors[i]; kk++)
{
  int k = global_neighbors[i][kk];
  pair<int> k_local = global_to_grain(k);
  int& type_k = (lattice[k.local.second]->iarray[0])[k.local.first];
  int& coordnum_k = (lattice[k.local.second]->iarray[2])[k.local.first];
  int& dep_flag_k = (lattice[k.local.second]->iarray[3])[k.local.first];
  array_coordinates& coordinates_k = lattice[k.local.second]->coordinates[k.local.first];
  dx = coordinates[i][0] - coordinates_k[0];
  dy = coordinates[i][1] - coordinates_k[1];
  dz = coordinates[i][2] - coordinates_k[2];

  if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
    dx = fabs(dx) - 2.*xmid;
  if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
    dy = fabs(dy) - 2.*ymid;
  if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
    dz = fabs(dz) - 2.*zmid;
  rsq = dx*dx + dy*dy + dz*dz;
  if (rsq > cutoff)
    continue;

  // hopping mechanism across grains
  // coordnum should be > 1, or else we're diffusing into thin air
  if ((i.local.second != k.local.second) && (type_k == 0)
      && (coordnum_k > 1) && (dep_flag[i] == -2)
      && (exchange_destination(k.local.first, k.local.second, direction)))
for (int kk = 0; kk < num_neighbors[i]; kk++)
{
  int k = neighbors[i][kk];
  if (direction != -1)
  {
    local_coord_k = get_local_coordinates(coordinates[k], i_local.second, dir);
    if (direction % 2 == 0)
      local_coord_k *= -1;
  }
  // regular hopping
  if ((type[k] == 0) && (coordnum[k] > 1) && (dep_flag[k] != -1))
  {
    // special case - grain boundary diffusion
    // assume that it happens only via hopping for now
    // Herzig2005 and Suzuki2005
    diff_sites.push_back(int_array2());
    if (dep_flag[i] == -2)
      diff_sites[diff_count][1] = 0;
    else
      diff_sites[diff_count][1] = 1;
    diff_sites[diff_count][0] = k;
    diff_count++; // # of possible diffusion sites
  }
  if ((type[k] == 1) && (local_coord_k < local_coord_i) && (coordnum[k] >= 6) && (direction != -1) && (dep_flag[i] != -2))
  // exchange
  {
    for (int nn = 0; nn < num_neighbors[k]; nn++)
    {
      int n = neighbors[k][nn]; // destination site
      if (dep_flag[n] == -1)
        continue;
      if (type[n] != 0)
        continue;
      if (exchange_destination(n, i_local.second, direction) == 0)
        continue;
      local_coord_n = get_local_coordinates(coordinates[n], i_local.second, dir);
if (direction % 2 == 0)
    local_coord *= -1;
if ((fabs(local_coord_n - local_coord_i) < 0.001)
    && (coordnum[k] >= 8))
    {
    // atom exchange
    diff_sites.push_back(int_array_2 ());
    diff_sites[diff_count][0] = n;
    diff_sites[diff_count][1] = 2;
    diff_count++;
    }
else if ((fabs(local_coord_n - local_coord_k) < 0.001)
    && (step_flag == 0)
    && ((direction % 2 == 0)
        && (coordinates[i][dir] < coordinates[n][dir])
        && (fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001))
    || ((direction % 2 == 1)
        && (coordinates[i][dir] > coordinates[n][dir])
        && (fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001)))
    {
    // step edge atom exchange
    diff_sites.push_back(int_array_2 ());
    diff_sites[diff_count][0] = n;
    diff_sites[diff_count][1] = 3;
    diff_count++;
    }
}
// end of exchange loop
// end of nearest neighbors loop
// if we are assuming that the diffusion rate is the same for all configurations
// and sites
// if diff is independent of the energy of the system
for (int k = 0; k < diff_count; k++)
    {
    j = diff_sites[k][0]; // find site j from diff_sites
    grain_j = i_local.second;
    pair_int j_local;
    // special case for hopping - hop across grains
    if (diff_sites[k][1] == 4)
        {
    j_local = global_to_grain(j);
    j = j_local.first;
    grain_j = j_local.second;
    einitial = system_energy(i,j_local.first,0,i_local.second,j_local.second);
    // type[j] = 1
    lattice[j_local.second]->iarray[0][j_local.first] = 1;
    type[i] = 0;
    efinal = system_energy(i,j_local.first,1,i_local.second,j_local.second);
        }
    else
        {
    einitial = system_energy(i,j,0,i_local.second,i_local.second);
    efinal = system_energy(i,j,1,i_local.second,i_local.second);
    }
type[j] = 1;
type[i] = 0; // in surface diffusion, i becomes vacant
efinal = system_energy(i, j, 1, i, local.second, i, local.second);
}

if (efinal <= einitial)
{
    if (diff_sites[k][1] == 0)
    {
        prob += v_boundary*exp(-1.0*Ed_boundary*t_inverse);
    }
    else if ((diff_sites[k][1] == 1) || (diff_sites[k][1] == 4))
    {
        prob += v_d*exp(-1.0*Ed*t_inverse);
    }
    else if (diff_sites[k][1] == 2)
    {
        prob += v_exch*exp(-1.0*Ed_exch*t_inverse);
    }
    else if (diff_sites[k][1] == 3)
    {
        prob += v_step*exp(-1.0*Ed_step*t_inverse);
    }
    if (prob >= threshold)
    {
        efinal = system_energy(i, j, 2, i, local.second, grain_j); // do the energy update
        if (diff_sites[k][1] == 0)
            n_boundary++;
        else if (diff_sites[k][1] == 1)
            n_hop++;
        else if (diff_sites[k][1] == 2)
            n_atomexch++;
        else if (diff_sites[k][1] == 3)
            n_step++;
        else if (diff_sites[k][1] == 4)
            n_hop++;
        break; // move accepted, break the for loop
    }
    else
    {
        if (diff_sites[k][1] != 4)
            type[j] = 0;
        else
            lattice[j, local.second]->iarray[0][j, local.first] = 0;
        type[i] = 1;
        j = -1; // set j = -1 since nothing happened
    }
}
else
{
    if (diff_sites[k][1] == 0)
    {
        prob += v_boundary*exp(-1.0*(Ed_boundary + efinal - einitial)*t_inverse);
    }
    else if ((diff_sites[k][1] == 1) || (diff_sites[k][1] == 4))
    {
        prob += v_d*exp(-1.0*(Ed + efinal - einitial)*t_inverse);
    }
    else if (diff_sites[k][1] == 2)
    {
        prob += v_exch*exp(-1.0*(Ed_exch + efinal - einitial)*t_inverse);
    }
    else if (diff_sites[k][1] == 3)
    {
        prob += v_step*exp(-1.0*(Ed_step + efinal - einitial)*t_inverse);
    }
    if (prob >= threshold)
efinal = system_energy(i,j,2,i.local.second,grain.j); // do the energy update
if (diff_sites[k][1] == 0)
    n_boundary++;
else if (diff_sites[k][1] == 1)
    n_hop++;
else if (diff_sites[k][1] == 2)
    n_atomexch++;
else if (diff_sites[k][1] == 3)
    n_step++;
else if (diff_sites[k][1] == 4)
    n_hop++;
break;
}
else // restore the types if i and j
{
    if (diff_sites[k][1] != 4)
        type[j] = 0;
    else
        lattice[j.local.second]->iarray[0][j.local.first] = 0;
    type[i] = 1;
    j = -1;
}
}

// end of diff_count loop

if (j == -1) // if nothing happened, skip the rest and return
{
    naccept--;
    return;
}
go to update;

update: coord_update(i,i.local.second); // update the coordination numbers
if (j != -1) // if there's a site j, update that too
    coord_update(j,grain.j);

// check the status of i as an active site and check if grain needs growing
update_status(i,i.local.second,1);

// same for j if there is a site j
if (j != -1)
    update_status(j,grain.j,1);

// check distance between grains
check_grain_distance(i,i.local.second);

for (int mm = 0; mm < num_neighbors[i]; mm++)
    check_grain_distance(neighbors[i][mm], i.local.second);

if (j != -1)
02943    check_grain_distance(j, grain_j);
02944    for (int mm = 0; mm < lattice[grain_j]->num_neighbors[j]; mm++)
02945        check_grain_distance(lattice[grain_j]->neighbors[j][mm], grain_j);
02946    }
02947
02948    if (grain_j != i_local.second)
02949        flag = 1; // flip the flag so that the update is done on the right grain
02950    for (int m = 0; m < lattice_count; m++)
02951        update_dep_flag(m);
02952
02953    scale_propensities();
02954
02955    // update the propensities for site i and its 2nd nearest neighbors - use global indices
02956    int ii = grain_to_global(i_local);
02957    sites.push_back(ii);
02958    propensity[ii] = site_propensity(ii);
02959
02960    for (int mm = 0; mm < num_neighbors2[i]; mm++)
02961    {
02962        int m = neighbors2[i][mm];
02963        pair<int, grain_j> = std::make_pair(m, i_local.second);
02964        sites.push_back(m);
02965        propensity[m] = site_propensity(m);
02966    }
02967
02968    if (j != -1) // if diffusion, need to update j and its 2nd nn as well
02969    {
02970        if (flag == 0) // if same grain
02971            if (!neighbor_check(i, j, i_local.second))
02972                jj = grain_to_global(std::make_pair(j, grain_j));
02973                sites.push_back(jj);
02974                propensity[jj] = site_propensity(jj);
02975            }
02976
02977        for (int mm = 0; mm < num_neighbors2[j]; mm++)
02978        {
02979            if (!neighbor_check(i, m, i_local.second))
02980                // if it's a '2nd nn' of i, it would have already been taken care of in
02981                // the first loop; this check is there to optimize the update
02982                if (!neighbor_check(i, m, i_local.second))
02983                    // if its '2nd nn' of j, it would have already been taken care of in
02984                    // the first loop; this check is there to optimize the update
02985                    {   
02986                        pair<int, grain_j> = std::make_pair(m, i_local.second);
02987                        m = grain_to_global(grain_m);
02988                        sites.push_back(m);
02989                        propensity[m] = site_propensity(m);
02990                    }
02991                }
02992
02993        else // if another grain is affected, flag = 1 and the update is done differently
02994            jj = grain_to_global(std::make_pair(j, grain_j));
02995            sites.push_back(jj);
02996            propensity[jj] = site_propensity(jj);
02997        }
02998

virtual void SPPARKS_NS::AppEAM::site_event_rejection ( int , class RandomPark * ) [inline], [virtual]
00055 { }

double AppEAM::site_propensity ( int i ) [virtual]

KMC method – compute total propensity of owned site summed over possible events The integer 'dir' is the direction that is normal to the surface at site i.

General Hopping Mechanism: Atom at site i can hop to a nearest neighbor site in the same grain or to a vacant surface site of another grain. For this to occur, the destination site, referred to as site k, will have to be vacant and its coordination number has to be greater than 1 to avoid diffusing into air.

Hopping to Another Surface Site (Different Grains): Same conditions as the general case, but a check is added to ensure that site k is a surface site of the grain.

General Exchange Mechanism: The intermediate site k will have a lower local 'dir'-coordinate than that of site i. The intermediate site will also have to be part of the crystal, hence coordnum[k] >= 6. The direction has to be a valid direction (!= -1) and the dep_flag of site i is not -2 (occupied and in proximity of an active site from another grain).

Atom Exchange: Sites i and n are in the plane where their local 'dir'-coordinates are equal. The coordination number of the intermediate site k should also be >= 8 to ensure that the site is 'buried' in the layer below.
Step Edge Atom Exchange: For step edge atom exchange, the intermediate site \( k \) and destination site \( n \) have the same ‘dir’ coordinate. The coordination number of \( k \) should be less than 8 such that it is part of the crystal but not completely in the bulk. The mod of direction will determine whether the event is occurring in the + or - direction with respect to ‘dir’-coordinate. If the mod is 0, \( i \) is at the - direction and destination site \( n \) will have a higher ‘dir’-coordinate than site \( i \). The last condition that \( \text{fabs(coordinates}[i][\text{dir}] - coordinates[n][\text{dir}]) > 0.001 \) is to ensure that the two sites are in different layers.

```c
int i_global = i;

pair_int i_local = global_to_grain(i);

double prob = 0.0;

int flag = 0; // flag to indicate whether step-edge atom exchange is possible or not

i = i_local.first; // get the local i and overwrite i

vec_int type = lattice[i_local.second]->iarray[0];
vec_int coordnum = lattice[i_local.second]->iarray[2];
vec_int & dep_flag = lattice[i_local.second]->iarray[3];
vec_int & coordinates = lattice[i_local.second]->coordinates;
vec_int & num_neighbors = lattice[i_local.second]->num_neighbors;
vec_int & neighbors = lattice[i_local.second]->neighbors;
vec_int & num_global_neighbors = lattice[i_local.second]->num_global_neighbors;
vec_int & global_neighbors = lattice[i_local.second]->global_neighbors;
vec_int & num_neighbors2 = lattice[i_local.second]->num_neighbors2;
vec_int & neighbors2 = lattice[i_local.second]->neighbors2;

double xmid = domain->midpoint(0);
double ymid = domain->midpoint(1);
double zmid = domain->midpoint(2);

// SPPARKS will calculate the propensities of all sites before performing site_event
// site_event will only update the propensities of affected sites

// propensity of all events at site i

// possible events: adsorption and diffusion, depending on the type of site i

// if the site is part of the bulk or in the vacuum, return prob = 0

int j = -1; // diffusion destination site index, -1 for adsorption

type[0] = 0; // energy of initial configuration
```

114
if dep_flag[i] = -1 or 0, return prob = 0
if (dep_flag[i] != 1)
    return prob;
if (count >= n_max)
    return prob;
einitial = system_energy(i,j,0,i_local.second,i_local.second);
type[i] = 1;
// get efinal without permanently changing the energy
efinal = system_energy(i,j,1,i_local.second,i_local.second);
type[i] = 0;
double area = (domain->length(0)) *(domain->length(1)); // find surface area in (Angstroms)^2
double numx = dep_sites_count*100./area; // possible dep sites per unit area (sites/nm^2)
// convert i_dep to A/nm^2 and calculate propensity
double prob_tmp = i_a_dep * 1.e-12 * (efinal - einitial)*scale_dep_on/ (z_me*charge*numx*max_eng);
if (prob_tmp < 0.) // if calculated prob is < 0, let it = 0
    { dep_diss_prob[i_global][0] = 0.;
    }
else
    { prob += prob_tmp;
    dep_diss_prob[i_global][0] = prob_tmp;
    }
if (pulse == 2)
{
einitial = system_energy(i,j,0,i_local.second,i_local.second);
type[i] = 1;
// find surface area in (Angstroms)^2
double area = (domain->length(0)) *(domain->length(1));
double numx = dep_sites_count*100./area;
efinal = system_energy(i,j,1,i_local.second,i_local.second);
type[i] = 0;
// convert i_dep to A/nm^2 and calculate propensity
double prob_tmp = i_a_dep * 1.e-12 * (efinal - einitial)*scale_dep_off/ (z_me*charge*numx*max_eng);
if (prob_tmp < 0.) // if calculated prob is < 0, let it = 0
    { ...
    }
else
    { ...
    }
}
if (type[i] != 0)
{
    for (int kk = 0; kk < num_neighbors[i]; kk++)
    {
        int k = neighbors[i][kk];
        if ((coordnum[k] == 1) && (type[k] == 1))
        // if coordnum of one of i’s neighbors is 1, that site will end up
        // floating in space when i diffuses away
        // this cannot happen, so if this is the case, i cannot diffuse
        return prob = 0.0;
    }
    if (coordinates[i][2] < (lattice[i_local.second]->next_site)
*{(lattice[i_local.second]->lattice/1.01))
    return prob;
}
if (dep_flag[i] == 2) && (count < n_max)
{
    einitial = system_energy(i,j,0,i_local.second,i_local.second);
    type[i] = 0;
    // get efinal without permanently changing the energy
    efinal = system_energy(i,j,1,i_local.second,i_local.second);
    type[i] = 1;
    double area = (domain->length(0)) *(domain->length(1)); // find surface area in (Angstroms)^2
    double numx = diss_sites_count*100./area; // possible dep sites per unit area
    scale_diss_on/ (zMe*charge*numx*(einitial-efinal)); // convert i_dep to A/nm^2 and
calculate propensity
}
prob += prob_tmp;

dep_diss_prob[i_global][2] = prob_tmp;
}

// anodic part of pulse-reverse -- remove sites at i_A_diss
if ((dep_flag[i] == 2) && (pulse == 2) && (count < n_max)) {
    einitial = system_energy(i,j,0,i_local.second,i_local.second);
    type[i] = 0;
    // get efinal without permanently changing the energy
    efinal = system_energy(i,j,1,i_local.second,i_local.second);
    type[i] = 1;
    double area = (domain->length(0)) * (domain->length(1)); // find surface area in (Angstroms)^2
    double numx = diss_sites_count*100./area; // possible dep sites per unit area (sites/\text{nm}^2)
    double efinal = (einitial-efinal); // possible dep sites per unit area (sites/\text{nm}^2)
    double prob_tmp = i_A_diss * 1.e-12 * (max_eng)*scale_diss_off / (zme*charge*numx*(einitial-efinal)); // Budevski et al. p.28
    if (prob_tmp < 0.) {
        dep_diss_prob[i_global][3] = 0.;
    } else {
        prob += prob_tmp;
        dep_diss_prob[i_global][3] = prob_tmp;
    }
}

// If site is occupied it can participate in diffusion across grains.
for (int kk = 0; kk < num_global_neighbors[i]; kk++) {
    int k = global_neighbors[i][kk];
    pair_int k_local = global_to_grain(k);
    // if they are in the same grain, ignore it
    if (k_local.second == i_local.second) {
        continue;
    }
    // we have already identified possible destination sites using dep_flag
    // when dep_flag[k] = -1, atom i can diffuse to site k (based on the
    // criteria of assigning -1 to dep_flag)
    vec_int& dep_flag_k = lattice[k_local.second]->iarray[3];
    // if dep_flag[k] is not -1, the two sites are not close enough
    if (dep_flag_k[k_local.first] != -1) {
        continue;
    }
vec_int & type_k = lattice[k_local.second]->iarray[0];
einitial = system_energy(i, k_local.first, 0, i_local.second, k_local.second);
type[i] = 0;
type[k_local.first] = 1;
efinal = system_energy(i, k_local.first, 1, i_local.second, k_local.second);
type[k_local.first] = 0;
type[i] = 1;
if (efinal <= einitial)
prob += v_grain*exp(-Ed_grain*t_inverse);
else
if (edge_check(i, i_local.second))
if it is the edge and is part of the crystal, no diffusion (for non PBC)
return prob;

// if site is part of the bulk, it cannot undergo diffusion
if (coordnum[i] > 6)
return prob;

// loop over first nearest neighbors to see if there’s any neighbor that has
dep_flag = -1 (close to or overlapping with another grain)
// if there is, change flag to -1 to prevent step-edge atom exch
for (int kk = 0; kk < num_neighbors[i]; kk++)
{
int k = neighbors[i][kk];
if (dep_flag[k] == -1)
flag = 1;
}

// surface diffusion, site i becomes vacant while destination j becomes occupied
// determine the direction that is normal to site i; function returns int [0,5]
int direction = determine_direction(i, i_local.second);
double local_coord_i, local_coord_k, local_coord_n;
int dir;

// function returns -1 when site is not exposed to the solution
if (direction != -1)
{
dir = static_cast<int>(floor(direction*0.5));
local_coord_i = get_local_coordinates(coordinates[i], i_local.second, dir);

// if we’re dealing with lower limit, multiply the coordinate by -1 to get the
// right comparison
if (direction % 2 == 0)
local_coord[i] *= -1;

double dx, dy, dz, rsq;

// define cutoff for diffusion involving sites from different grains - use latconst
double cutoff = (lattice[i_local.second]->latconst)*(lattice[i_local.second]->latconst);

// hopping to another grain on the surface - loop over global_neighbors
for (int kk = 0; kk < num_global_neighbors[i]; kk++)
{
    int k = global_neighbors[i][kk];
    pair<int, int> local = global_to_grain(k);

    int& type_k = (lattice[k_local.second]->iarray[0])[k_local.first];
    int& coordnum_k = (lattice[k_local.second]->iarray[2])[k_local.first];
    int& dep_flag_k = (lattice[k_local.second]->iarray[3])[k_local.first];
    array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];

    dx = coordinates[i][0] - coordinates_k[0];
    dy = coordinates[i][1] - coordinates_k[1];
    dz = coordinates[i][2] - coordinates_k[2];

    if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
        dx = fabs(dx) - 2.*xmid;
    if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
        dy = fabs(dy) - 2.*ymid;
    if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
        dz = fabs(dz) - 2.*zmid;

    rsq = dx*dx + dy*dy + dz*dz;

    // if outside of cutoff, ignore that site
    if (rsq > cutoff)
        continue;

    if ((i_local.second != k_local.second) && (type_k == 0)
        && (coordnum_k > 1) && (dep_flag_k == -1) && (dep_flag[i] == -2)
        && (exchange_destination(k_local.first, k_local.second, direction)))
    {
        // hopping to another grain on the surface
        einitial = system_energy(i,k_local.first,0,i_local.second,k_local.second);

        type_k = 1; // update the type of site j
        type[i] = 0; // in surface diffusion, i becomes vacant

        efinal = system_energy(i,k_local.first,1,i_local.second,k_local.second);

        // restore the types
        type[i] = 1;
    }
}

else
    prob += \( v_d \times \exp(-1.0 \times (E_d + e_{\text{final}} - e_{\text{initial}})) \times t^{-1} \);
}
for (int kk = 0; kk < num_neighbors[i]; kk++)
{
    int k = neighbors[i][kk];
    if (direction != -1)
    {
        local\_coord\_k = get\_local\_coordinates(coordinates[k], i\_local.second, dir);
        if (direction % 2 == 0)
            local\_coord\_k *= -1;
    }
    if ((type[k] == 0) && (coordnum[k] > 1) && (dep\_flag[k] != -1))
    {
        // hopping
        // cannot diffuse to a site that has coordnum <= 1, it will be floating
        // if type[nearest neighbors] = 0, find the propensity of the move
        j = k;
        e_{\text{initial}} = \text{system\_energy}(i, j, 0, i\_local.second, i\_local.second);
        type[j] = 1; // update the type of site j
        type[i] = 0; // in surface diffusion, i becomes vacant
        e_{\text{final}} = \text{system\_energy}(i, j, 1, i\_local.second, i\_local.second);
        // restore the types
        type[j] = 0;
        type[i] = 1;
        if (e_{\text{final}} <= e_{\text{initial}})
        {
            if (dep\_flag[i] == -2) // special case, grain boundary diffusion
                prob += \( v_{\text{boundary}} \times \exp(-1.0 \times E_{\text{boundary}}) \times t^{-1} \);
            else // otherwise, regular hopping
                prob += \( v_{d} \times \exp(-1.0 \times E_{d}) \times t^{-1} \);
        }
    }
    else if ((type[k] == 1) && (local\_coord\_k < local\_coord\_i) && (coordnum[k] >= 6) && (direction != -1) && (dep\_flag[i] != -2))
    {
        // exchange mechanism
        // coordnum = 6 is a kink site, can still participate in exchange
        // find intermediate sites, k, whose z[k] < z[i] (lower plane)
        // Antczak and Ehrlich, p. 88-104
for (int nn = 0; nn < num_neighbors[k]; nn++)
{
    int n = neighbors[k][nn]; // destination site
    if (dep_flag[n] == -1)
        continue;
    if (type[n] != 0)
        continue;
    if (exchange_destination(n, i_local.second, direction) == 0)
        continue;

    local_coord_n = get_local_coordinates(coordinates[n], i_local.second, dir);
    if (direction % 2 == 0)
        local_coord_n *= -1;
    if ((fabs(local_coord_n - local_coord_i) < 0.001) && (coordnum[k] >= 8))
    {
        // atom exchange on a plateau
        j = n;
        einitial = system_energy(i, j, 0, i_local.second, i_local.second);
        type[j] = 1;
        type[i] = 0;
        efinal = system_energy(i, j, 1, i_local.second, i_local.second);
        type[j] = 0;
        type[i] = 1;
        if (efinal <= einitial)
            prob += v_exch*exp(-1.0*Ed_exch*t_inverse);
        else
            prob += v_exch*exp(-1.0*(Ed_exch+efinal-einitial)*t_inverse);
    }
    else if ((fabs(local_coord_n - local_coord_k) < 0.001)
        && (coordnum[k] < 8) && (flag == 0)
        && (coordinates[i][dir] < coordinates[n][dir])
        || (coordinates[i][dir] > coordinates[n][dir]))
    {
        // step edge atom exchange at edges
        // atom i will displace atom k, pushing atom k outwards
// coordnum[k] < 8 to ensure that it is at the edge
j = n;
einitial = system_energy(i, j, 0, i_local.second, i_local.second);
type[j] = 1;
type[i] = 0;
efinal = system_energy(i, j, 1, i_local.second, i_local.second);
type[j] = 0;
type[i] = 1;
if (efinal <= einitial)
    prob += v_step*exp(-1.0*Ed_step*t_inverse);
else
    prob += v_step*exp(-1.0*(Ed_step+efinal-einitial)*t_inverse);
}
}
}
}
}

return prob;

void AppEAM::stats ( char ∗ strtmp )
{
char big[8], format[256];
strcpy(big, BIGINT_FORMAT);
if (solve)
{
sprintf(format, "%%10g %%10s %%10d %%10d %%10d %%10d %%10d
%%10d %%10d %%10g %%10g", &big[1]);
sprintf(strtmp, format, time, naccept, n_hop, n_atomexch,
        n_step, n_boundary, n_grain, count, n_diss,
        dep_sites_count, diss_sites_count, scale_dep_on,
        scale_diss_on);
}
else
{
sprintf(format, "%%10g %%10s %%10d %%10d %%10d %%10d %%10d
%%10d %%10d %%10g %%10g", &big[1], &big[1]);
sprintf(strtmp, format, time, naccept, n_hop, n_atomexch,
        n_step, n_boundary, n_grain, count, n_diss,
        dep_sites_count, diss_sites_count, scale_dep_on,
        scale_diss_on);
}
}
void AppEAM::stats_header ( char * strtmp )
03045 { 
03046 sprintf(strtmp,"%10s %10s %10s %10s %10s %10s %10s %10s %10s %10s %10s %10s %10s", "Time", 
03048 "sum_dep", "sum_diss", "scale_dep", "scale_diss");
03049 }

double AppEAM::system_energy ( int i, int j, int flag, int grain_i, int grain_j ) [private]

Calculates the energy of sites affected by a move

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Local index of site in question</td>
</tr>
<tr>
<td>j</td>
<td>If the event is a diffusion event, local index of destination site</td>
</tr>
<tr>
<td>flag</td>
<td>Flag to indicate whether a permanent update of the energies is required or not</td>
</tr>
<tr>
<td>grain_i</td>
<td>Grain ID of the site i</td>
</tr>
<tr>
<td>grain_j</td>
<td>Grain ID of the site j</td>
</tr>
</tbody>
</table>

Computes the energy of system for before or after a proposed change Function will calculate the energy of the system after a proposed change in i calling site_energy for site i and its 2nd nearest neighbors.

This is because when site i is changed, the energy of its 2nd nearest neighbors are also affected. But the change of site i will not affect its neighbors that is outside the cutoff range of site i.

The function is used to determine einitial and efinal to calculate the change in the energy of the system after some change.

The flag is to indicate einitial (flag = 0), efinal without overwrite (flag = 1), efinal with overwrite (flag = 2). When flag = 2, the phis from phi_old are not restored into phi.

When the function is run with flag = 0, as is the case when calculating einitial, the function will also find the indices of sites that will be affected by some change in i and j. This is stored in the n x 3 vector phi_old. The first column in phi_old stores the indices of the sites. The second column stores the phis of the sites before any change is made to the system. The last column stores the new values of phis.

01393 {
01417 double eng = 0.0;
if (flag == 0)
{
  double xmid = domain->midpoint(0);
  double ymid = domain->midpoint(1);
  double zmid = domain->midpoint(2);
  double dx, dy, dz, rsq;
  // use latconst as the cutoff
  double cutoff = (lattice[0]->latconst) * (lattice[0]->latconst);

  // initialize the references to the grain data
  double& phi_i = (lattice[grain_i]->darray[1])[i];

  // create a copy of i's global_neighbors list
  vec<int> global_neighbors_i(lattice[grain_i]->global_neighbors[i]);
  int num_global_neighbors_i = lattice[grain_i]->num_global_neighbors[i];
  array_coordinates& coordinates_i = lattice[grain_i]->coordinates[i];

  old_eng = 0.0; // zero out old_eng

  for (int kk = 0; kk < lattice_count; kk++)
  {
    for (int k = 0; k < lattice[kk]->N; k++)
      old_eng += lattice[kk]->darray[1][k]; // add up energy before change
  }

  old_eng = 0.0; // clear out phi_old
  phi_old.clear();

  phi_old.push_back(array_coordinates());
  phi_old[0][0] = grain_to_global(std::make_pair(i, grain_i));
  phi_old[0][1] = phi_i; // start adding sites that will see their energy changed

  phi_old_count = 1; // number of sites whose phi are stored in phi_old

  // loop over global neighbors (within EAM potential cutoff)
  for (int kk = num_global_neighbors_i-1; kk >= 0; kk--)
  {
    int k = global_neighbors_i[kk];
    int k_local = global_to_grain(k);
    array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];

    dx = coordinates_i[0] - coordinates_k[0];
    dy = coordinates_i[1] - coordinates_k[1];
    dz = coordinates_i[2] - coordinates_k[2];

    if (fabs(dx) > xmid) && (domain->periodicity[0] == 1))
      dx = fabs(dx) - 2. * xmid;
    if (fabs(dy) > ymid) && (domain->periodicity[1] == 1))
      dy = fabs(dy) - 2. * ymid;
    if (fabs(dz) > zmid) && (domain->periodicity[2] == 1))
      dz = fabs(dz) - 2. * zmid;

    rsq = dx*dx + dy*dy + dz*dz;

    // ignore sites outside the cutoff
if (rsq > cutoff) {
    global_neighbors_i.erase(global_neighbors_i.begin() + kk);
    continue;
}

phi_old.push_back(array_coordinates());
phi_old[phi_old_count][0] = k;
phi_old[phi_old_count][1] = lattice[k_local.second]->darray[1][k_local.first];
phi_old_count++;

// if the event is surface diffusion, the energy of site j and its 2nd nearest neighbor are also stored in phi_old
if (j != -1) {
    double& phi_j = lattice[grain_j]->darray[1][j];
    vec_int& global_neighbors_j = lattice[grain_j]->global_neighbors[j];
    array_coordinates& coordinates_j = lattice[grain_j]->coordinates[j];
    phi_old.push_back(array_coordinates());
    phi_old[phi_old_count][0] = grain_to_global(std::make_pair(j, grain_j));
    phi_old[phi_old_count][1] = phi_j;
    phi_old_count++;
}

// no need to create a copy for j's global neighbor list
for (int kk = 0; kk < global_neighbors_j.size(); kk++) {
    int k = global_neighbors_j[kk];
    pair<int, int> k_local = global_to_grain(k);
    array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];
    dx = coordinates_j[0] - coordinates_k[0];
    dy = coordinates_j[1] - coordinates_k[1];
    dz = coordinates_j[2] - coordinates_k[2];
    if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
        dx = fabs(dx) - 2.*xmid;
    if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
        dy = fabs(dy) - 2.*ymid;
    if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
        dz = fabs(dz) - 2.*zmid;
    rsq = dx*dx + dy*dy + dz*dz;
    if (rsq > cutoff) continue;
    // if not part of i's global neighbor2 list - add to phi_old
    if (std::find(global_neighbors_i.begin(), global_neighbors_i.end(), k) ==
        global_neighbors_i.end()) {
        phi_old.push_back(array_coordinates());
        phi_old[phi_old_count][0] = k;
    }
Updates dep_flag after an event is accepted
Parameters

| grain | Grain ID that is being checked |

Updates dep_flag for all active sites in the grain and update the deposition site and dissolution site counts.

Possible values of dep_flag are: 0 (not an active site), 1 (available for deposition), 2 (available for dissolution), -1 (occupied, the site is at the edge of the grain) and -2 (unoccupied, the site is at the edge of the grain)

```cpp
01049 {  
01050  vec_int& type = lattice[grain]->iarray[0];  
01051  vec_int& coordnum = lattice[grain]->iarray[2];  
01052  vec_int& dep_flag = lattice[grain]->iarray[3];  
01053  vec_coordinates& coordinates = lattice[grain]->coordinates;  
01054  int diss_old = diss_sites_count[grain];  
01055  int dep_old = dep_sites_count[grain]; // store old value  
01056  for (int ii = 0; ii < active_sites[grain].size(); ii++)  
01057  {  
01058   int i = active_sites[grain][ii];  
01059   // if coordnum != 12 and 0, vacant, not in proximity of another grain, site is  
01060   // available for deposition  
01061   if ((type[i] == 0) && (coordnum[i] != 12) && (coordnum[i] != 0)  
01062   && (dep_flag[i] >= 0))  
01063   dep_flag[i] = 1; // if possible deposition site, change the flag to 1  
01064   else if ((coordnum[i] < 10) && (type[i] == 1) && (coordinates[i][0] > min_dist)  
01065   && (coordinates[i][0] < max_dist_x) && (coordinates[i][1] > min_dist)  
01066   && (coordinates[i][1] < max_dist_y) && (coordinates[i][2] > min_dist)  
01067   && (determine_direction(i,grain) != -1) || (dep_flag[i] >= 0))  
01068   dep_flag[i] = 2; // possible dissolution site, change flag to 2  
01069   else  
01070   {  
01071   // if coordnum != 12 and 0, vacant, not in proximity of another grain, site is  
01072   // available for deposition  
01073   // if site is occupied, has coordnum < 10 - it can undergo dissolution  
01074   // the condition in coordnum and determine_direction is to ensure that it is  
01075   // at the topmost layer in a particular plane  
01076   }  
01077   else if ((coordnum[i] < 10) && (type[i] == 1) && (coordinates[i][0] > min_dist)  
01078   && (coordinates[i][0] < max_dist_x) && (coordinates[i][1] > min_dist)  
01079   && (coordinates[i][1] < max_dist_y) && (coordinates[i][2] > min_dist)  
01080   && (determine_direction(i,grain) == -1) || (dep_flag[i] == 0))  
01081   dep_flag[i] = 2; // possible dissolution site, change flag to 2  
01082   else  
01083   {  
01084   // edge sites are excluded to prevent weird things from happening to the domain  
01085   // resets dep_sites_count[grain]  
01086   dep_sites_count[grain] = 0;  
01087   diss_sites_count[grain] = 0;  
01088   for (int ii = 0; ii < active_sites[grain].size(); ii++)  
01089   {  
```

void AppEAM::update_status ( int i, int grain, int flag ) [private]

Checks whether the active site i is at the edge of existing lattice sites. Updates the status of site i and its nearest neighbors in the active_sites vector.

Parameters

<table>
<thead>
<tr>
<th>i</th>
<th>Local index of site i</th>
</tr>
</thead>
<tbody>
<tr>
<td>grain</td>
<td>Grain ID of site i</td>
</tr>
<tr>
<td>flag</td>
<td>Flag indicating whether function is called in site_event() or not</td>
</tr>
</tbody>
</table>

Calls check_box and does the necessary update for AppEAM-specific data structures. Updates active_sites with the new status of i and its neighbors.
vec<int> type = lattice[grain]->iarray[0];
vec<int> grainid = lattice[grain]->iarray[1];

// find where i is in the active sites list
auto loc = std::find(active_sites[grain].begin(), active_sites[grain].end(), i);

// max distance away from the box boundary to be considered an edge site
double max_dist = (lattice[grain]->next_site)*(lattice[grain]->latconst);

// if not in the list and not at the edge, update active_sites
if (loc == active_sites[grain].end())
{
    if (((coordnum[i] != 12) && (coordnum[i] != 0))
    {
        active_sites[grain].push_back(i);
    }
    for (int mm = 0; mm < num_neighbors[i]; mm++)
    {
        int m = neighbors[i][mm];
        // condition for an active site
        if (((coordnum[m] != 12) && (coordnum[m] != 0))
        {
            active_sites[grain].push_back(m);
        }
        else if (((coordnum[m] == 12) || (coordnum[m] == 0))
        {
            auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
            active_sites[grain].erase(loc2);
            dep_flag[m] = 0;
        }
    }
}
else
{
    // check if i is no longer an active site
    if (((coordnum[i] == 12) || (coordnum[i] == 0))
    {
        active_sites[grain].erase(loc);
        dep_flag[i] = 0;
    }
    // check its neighbors too
    for (int mm = 0; mm < num_neighbors[i]; mm++)
    {
        int m = neighbors[i][mm];
        if (((coordnum[m] != 12) && (coordnum[m] != 0))
        {
            active_sites[grain].push_back(m);
        }
        else if (((coordnum[m] == 12) || (coordnum[m] == 0))
        {
            active_sites[grain].push_back(m);
        }
    }
}
else if (((coordnum[m] == 12) || (coordnum[m] == 0))
auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
active_sites[grain].erase(loc2);
dep_flag[m] = 0;
}
}
}
}

// if site is not at the edge, then check whether it needs to grow or not
auto loc2 = std::find(neighbors[i].begin(), neighbors[i].end(), -1);

if ((loc2 != neighbors[i].end())
&& (fabs(coordinates[i][0]) > max_dist)
&& (fabs(coordinates[i][1]) > max_dist)
&& (fabs(coordinates[i][2]) > max_dist)
&& (loc != active_sites[grain].end()))
// if one of its neighbors = -1, grow the lattice
{
    int grow = update_grain(i, grain); // calls check_grain in app_lattice to check
    // if we did grow the grain the number of sites would have increased
    if (grow)
    {
        std::cout << "Updating propensity...";
        for (int m = Nold; m < lattice[grain]->N; m++)
        {
            // update all the arrays, no need to update energy
            // and type since the new sites start out as unoccupied
            grainid[m] = grain;
            coord_update(m, grain);
        }
        // check the status of the new sites
        if (((coordnum[m] != 12) && (coordnum[m] != 0))
            && (!active_sites_check(m, grain)))
        {
            active_sites[grain].push_back(m);
        } else if (((coordnum[m] == 12) || (coordnum[m] == 0))
            && (active_sites_check(m, grain)))
        {
            auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
            active_sites[grain].erase(loc2);
der_flag[m] = 0;
        }
    }
    for (int m = 0; m < lattice_count; m++)
    {
        check_grain_distance(m);
    }
    for (int m = 0; m < lattice_count; m++)
    {
        update_dep_flag(m);
    }
    dep_diss_prob.resize(Ntotal, vec_double(4, 0.));
if (flag == 1)
{
    for (int m = N_old; m < lattice[grain]->N; m++)
        propensity.push_back(site_propensity(grain_to_global(std::make_pair(m, grain))));
    for (int n = 0; n < active_sites[grain].size(); n++)
        site_propensity(grain_to_global(std::make_pair(active_sites[grain][n], grain)));
    solve->resize(N_total, propensity); // resize the propensity bin
}

std::cout << " done" << std::endl;

// check if i is no longer an active site
if ((coordnum[i] == 12) || (coordnum[i] == 0))
{
    active_sites[grain].erase(loc);
    dep_flag[i] = 0;
}

// check its neighbors too
for (int mm = 0; mm < num_neighbors[i]; mm++)
{
    int m = neighbors[i][mm];
    if ((coordnum[m] != 12) && (coordnum[m] != 0) && (active_sites_check(m, grain)))
    {
        active_sites[grain].push_back(m);
    }
    else if (((coordnum[m] == 12) || (coordnum[m] == 0)) && (active_sites_check(m, grain)))
    {
        auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
        active_sites[grain].erase(loc2);
        dep_flag[m] = 0;
    }
}

Member Data Documentation

double SPPARKS_NS::AppEAM::alpha_a [private]

Transfer coefficient for anodic reaction
double SPPARKS_NS::AppEAM::alpha_c [private]
Transfer coefficient for cathodic reaction

double SPPARKS_NS::AppEAM::charge [private]
Elementary charge (C)

int SPPARKS_NS::AppEAM::count [private]
Number of sites deposited

vec_vec_double SPPARKS_NS::AppEAM::dep_diss_prob [private]
Stores deposition and dissolution propensity contributions - used in scaling; columns: dep when pulse is on, dep when pulse is off, diss when pulse is on, diss when pulse is off

int SPPARKS_NS::AppEAM::dep_mode [private]
Deposition mode (0 galvanostaic, 1 potentiostatic)

vec_int SPPARKS_NS::AppEAM::dep_sites [private]
List of sites whose propensities need to be scaled (deposition)

int SPPARKS_NS::AppEAM::dep_sites_count [private]
Total number of possible deposition sites

vec_int SPPARKS_NS::AppEAM::dep_sites_count_grain [private]
Sum of dep_flag of active sites in each grain
vec_int_array_2 SPPARKS_NS::AppEAM::diff_sites [private]
A n x 2 vector of possible diffusion sites and the type of diffusion

vec_int SPPARKS_NS::AppEAM::diss_sites [private]
List of sites whose propensities need to be scaled (dissolution)

int SPPARKS_NS::AppEAM::diss_sites_count [private]
Total number of possible dissolution sites

vec_int SPPARKS_NS::AppEAM::diss_sites_count_grain [private]
Number of possible dissolution sites in each grain

double SPPARKS_NS::AppEAM::Ed [private]
Energy barrier for hopping diffusion

double SPPARKS_NS::AppEAM::Ed_boundary [private]
Energy barrier for diffusion along grain boundaries

double SPPARKS_NS::AppEAM::Ed_exch [private]
Energy barrier for atom exchange diffusion

double SPPARKS_NS::AppEAM::Ed_grain [private]
Energy barrier for diffusion across grain boundaries

double SPPARKS_NS::AppEAM::Ed_step [private]
Energy barrier for step-edge atom exchange diffusion
double SPPARKS_NS::AppEAM::eta [private]
Overpotential (V)

double SPPARKS_NS::AppEAM::i0 [private]
Exchange current density (pA/nm^2)

double SPPARKS_NS::AppEAM::i_a_dep [private]
Deposition current density during oxidation (pA/nm^2)

double SPPARKS_NS::AppEAM::i_a_diss [private]
Dissolution current density during oxidation (pA/nm^2)

double SPPARKS_NS::AppEAM::i_dep [private]
Deoposition current density (pA/nm2)

double SPPARKS_NS::AppEAM::i_diss [private]
Dissolution current density (pA/nm2)

double SPPARKS_NS::AppEAM::max_eng [private]
Max value of energy at a site at t = 0

int SPPARKS_NS::AppEAM::n_atomexch [private]
Number of atom exchange diffusion events occurred

int SPPARKS_NS::AppEAM::n_boundary [private]
Number of diffusion events along grain boundaries

134
int SPPARKS_NS::AppEAM::n_diss [private]
Number of sites removed

int SPPARKS_NS::AppEAM::n_grain [private]
Number of diffusion events across a grain boundary

int SPPARKS_NS::AppEAM::n_hop [private]
Number of hopping diffusion events occurred

int SPPARKS_NS::AppEAM::n_max [private]
Maximum number of sites deposited

int SPPARKS_NS::AppEAM::n_step [private]
Number of step-edge atom exchange diffusion events occurred

double SPPARKS_NS::AppEAM::off_dt [private]
Pulse off time

double SPPARKS_NS::AppEAM::old_eng [private]
Total energy of affected sites before event occurs

double SPPARKS_NS::AppEAM::on_dt [private]
Pulse on time

double SPPARKS_NS::AppEAM::ox_rate [private]
Overpotential during ‘off’ time in pulse-reverse (V)
PairEAM* SPPARKS_NS::AppEAM::pair [private]

vec_coordinates SPPARKS_NS::AppEAM::phi_old [private]
A n x 3 vector of site indices, energy, new energy

int SPPARKS_NS::AppEAM::phi_old_count [private]
Number of rows used in phi_old

double SPPARKS_NS::AppEAM::polish_height [private]
Height to polish to (Angstroms)

double SPPARKS_NS::AppEAM::prev_time [private]
Previous time step

int SPPARKS_NS::AppEAM::pulse [private]
Flag to indicate pulse plating

int SPPARKS_NS::AppEAM::pulse_on [private]
Pulse on/off

double SPPARKS_NS::AppEAM::scale_dep_off [private]
Scaling parameter for propensity when deposition is off

double SPPARKS_NS::AppEAM::scale_dep_on [private]
Scaling parameter for propensity when deposition is on
double SPPARKS_NS::AppEAM::scale_diss_off  [private]
Scaling parameter for propensity when dissolution is off

double SPPARKS_NS::AppEAM::scale_diss_on  [private]
Scaling parameter for propensity when dissolution is on

double SPPARKS_NS::AppEAM::seed_frac  [private]
Seed layer occupancy fraction

int SPPARKS_NS::AppEAM::seed_n  [private]
Number of sites in seed layer

double SPPARKS_NS::AppEAM::seedlayer_size  [private]
Height of seed layer (Angstroms)

vec_int SPPARKS_NS::AppEAM::sites [private]
Vector of sites whose propensity needed updating

double SPPARKS_NS::AppEAM::v_boundary [private]
Frequency factor for diffusion along grain boundaries

double SPPARKS_NS::AppEAM::v_d [private]
Frequency factor for hopping diffusion

double SPPARKS_NS::AppEAM::v_exch [private]
Frequency factor for atom exchange diffusion
double SPPARKS_NS::AppEAM::v_grain [private]
Frequency factor for diffusion across grain boundaries

double SPPARKS_NS::AppEAM::v_step [private]
Frequency factor for step-edge atom exchange diffusion

double SPPARKS_NS::AppEAM::z_me [private]
Number of electrons transferred in reduction reaction
References


