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

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

List of Figures

Nomenclature

α_a	Charge transfer coefficient for anodic reaction
α_c	Charge transfer coefficient for cathodic reaction
β	Power law exponent
$\boldsymbol{\sigma}$	State of system
ΔE	Energy difference between final and initial states (eV)
δ_C	Difference between the mean atom coordination number of the equilibrium deposit configurations from KMC-EAM and from MD-EAM (%)
δ_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 (%)
δ_{ij}	Kronecker delta
η	Overpotential (V)
$\Gamma_{i,j}$	Propensity at site i due to event j (s^{-1})
γ_i	Activity of species i
$\mathbf{R}$	3×3 rotation matrix

$\mathbf{t}$ 3×1 translation vector

$\mathbf{x}_i^{'}$ 3×1 vector representing the new coordinates of site i

$\mathbf{x}_i$ 3×1 vector of reference coordinates for site i

$\mathcal{E}$ Half-cell voltage (V)

$\mathcal{E}_{appl}$ Applied potential (V)

$\mathcal{F}$ Multi-body embedding energy functional (eV)

ν Kinematic viscosity ($m^2\ s^{-1}$)

ν_{ads} Adsorption frequency (s^{-1})

ν_a Frequency rate (s^{-1})

ν_{desorb} Desorption frequency (s^{-1})

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

$\overline{\chi}$ 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)

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

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

ρ_h Function that quantifies the electron density of a neighboring atom

ρ_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 (1.602×10^{-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_{dep}^{chem} Chemical potential contribution to activation energy of deposition (eV)

E_{diss} Activation energy of dissolution (eV)

E_{diss}^{chem} 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 (8.617×10^{-5} eV K^{-1})

L Number of possible grain orientations

m Number of moles of metal (mol)

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

$M^0_{(s)}$ 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 m^{-2})

n_{diss} Number of possible dissolution sites per unit area (sites 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

$P(\boldsymbol{\sigma})$ 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_{aq}^{z+}) in an aqueous electrolyte is represented by the following reaction [1]:

$$M_{(aq)}^{z+} + ze^{-} \leftrightharpoons M_{(s)}^{0}. \tag{2.1}$$

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

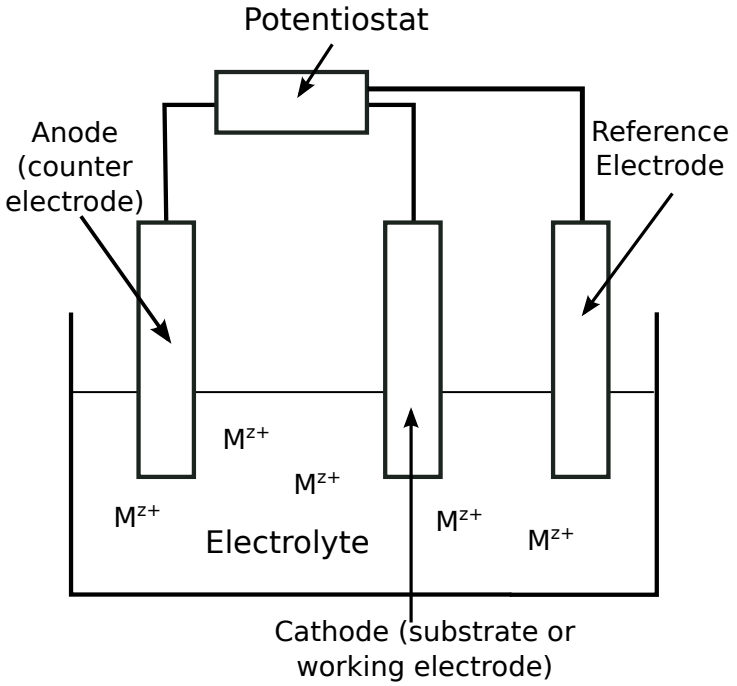


Figure 2.1: Schematic diagram of an electrochemical cell.

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,

$$\mathcal{E} = \mathcal{E}^0 + \frac{RT}{zF} \ln \gamma^z_{M_{(aq)}}, \tag{2.2}$$

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]:

$$r = \frac{\mathrm{d}m}{\mathrm{d}t} = \frac{iA}{zF}, \tag{2.3}$$

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]:

$$r = \overleftarrow{r} - \overrightarrow{r}, \tag{2.4}$$

$$\overrightarrow{r} = \overrightarrow{k}\, \gamma^z_{M_{(aq)}}, \tag{2.5}$$

$$\overleftarrow{r} = \overleftarrow{k}\, \gamma^0_{M_{(s)}} = \overleftarrow{k}\,, \tag{2.6}$$

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]:

$$\overrightarrow{k} = \overrightarrow{B} \exp\left(-\frac{E_{dep}}{k_B T}\right), \tag{2.7}$$

$$\overleftarrow{k} = \overleftarrow{B} \exp\left(-\frac{E_{diss}}{k_B T}\right), \tag{2.8}$$

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]:

$$E_{dep} = E_{dep}^{chem} + \alpha_c z F \mathcal{E}_{appl}/e, \tag{2.9}$$

$$E_{diss} = E_{diss}^{chem} - \alpha_a z F \mathcal{E}_{appl}/e, \tag{2.10}$$

where E_{dep}^{chem} and E_{diss}^{chem} are the chemical potential contributions to the activation energy of deposition and dissolution, respectively, α_a is the charge transfer coefficient for the reverse direction of reaction (2.1) (i.e. dissolution), α_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:

$$\overrightarrow{k} = \overrightarrow{B} \exp\left(-\frac{E_{dep,chem}}{k_B T}\right) \exp\left(-\frac{\alpha_c z \mathcal{E}_{appl}}{k_B T}\right), \tag{2.11}$$

$$\overleftarrow{k} = \overleftarrow{B} \exp\left(-\frac{E_{diss,chem}}{k_B T}\right) \exp\left(\frac{\alpha_a z \mathcal{E}_{appl}}{k_B T}\right). \tag{2.12}$$

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:

$$\overrightarrow{i} = \frac{zF\overrightarrow{B}\gamma^z_{M_{(aq)}}}{A} \exp\left(-\frac{E_{dep,chem}}{k_BT}\right) \exp\left(-\frac{\alpha_c z\mathcal{E}_{appl}}{k_BT}\right), \tag{2.13}$$

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

At equilibrium, $\overrightarrow{i} = \overleftarrow{i} = i^0$ and $\mathcal{E}_{appl} = \mathcal{E}$, where i^0 is the exchange current density (A m^{-2}). From Eqn (2.1), the overall current density is $i = \overleftarrow{i} - \overrightarrow{i}$. 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 η is the overpotential (V) [1],

$$\eta = \mathcal{E}_{appl} - \mathcal{E} \tag{2.16}$$

and the exchange current density is given as:

$$i^0 = \frac{zF\overrightarrow{B}\gamma^z_{M_{(aq)}}}{A} \exp\left(-\frac{E_{dep,chem}}{k_BT}\right) \exp\left(-\frac{\alpha_c z\mathcal{E}}{k_BT}\right) \tag{2.17}$$

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

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]:

$$\overrightarrow{i} = -i_{dep} = ze\Gamma_{dep}n_{dep}, \tag{2.19}$$

$$\overleftarrow{i} = i_{diss} = ze\Gamma_{diss}n_{diss}, \tag{2.20}$$

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

$$E_i = \mathcal{F}[\rho_i] + \frac{1}{2}\sum_{\substack{j \\ i \neq j}}^{n_c} \phi_{ij}(r_{ij}) \tag{2.21}$$

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 ρ_i is the total host electron density for atom i:

$$\rho_i = \sum_{\substack{j \\ i \neq j}}^{n_c} \rho_h(r_{ij}) \tag{2.22}$$

where ρ_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,

$$E = \sum_{i}^{N} \sigma_i E_i, \tag{2.23}$$

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 $\boldsymbol{\sigma}$ of the system is a function of only lattice site occupancy and time, where $\sigma_i = 0$ for a vacant site and

$\sigma_i = 1$ 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 $P(\boldsymbol{\sigma})$ of observing state $\boldsymbol{\sigma}$ is given as [54, 55]:

$$\frac{\mathrm{d}P(\boldsymbol{\sigma})}{\mathrm{d}t} = \sum_{\substack{\boldsymbol{\sigma}' \\ \boldsymbol{\sigma}' \neq \boldsymbol{\sigma}}} G(\boldsymbol{\sigma}' \rightarrow \boldsymbol{\sigma}) P(\boldsymbol{\sigma}') - \sum_{\substack{\boldsymbol{\sigma}' \\ \boldsymbol{\sigma}' \neq \boldsymbol{\sigma}}} G(\boldsymbol{\sigma} \rightarrow \boldsymbol{\sigma}') P(\boldsymbol{\sigma}), \tag{2.24}$$

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

$$\mathrm{d}\sigma_i = \sum_j \Gamma_{ij}^{+}(\boldsymbol{\sigma})\mathrm{d}t - \sum_j \Gamma_{ij}^{-}(\boldsymbol{\sigma})\mathrm{d}t, \tag{2.25}$$

where $\Gamma_{ij}(\boldsymbol{\sigma})$ is the transition probability (s^{-1}) or propensity function for process j at site i when the state $\boldsymbol{\sigma}$ is observed. The term $\Gamma_{ij}(\boldsymbol{\sigma})\mathrm{d}t$ gives the probability of state $\boldsymbol{\sigma}$ undergoing a change due to some move j at site i within the time increment $\mathrm{d}t$ [57]. The '+' and '−' signs denote whether site i is entering state $\boldsymbol{\sigma}$ or leaving state $\boldsymbol{\sigma}$. 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 $P(\boldsymbol{\sigma}_{eq})$ is equal to zero. Eqn (2.24) becomes:

$$\sum_{\substack{\boldsymbol{\sigma}' \\ \boldsymbol{\sigma}'_{eq} \neq \boldsymbol{\sigma}}} G(\boldsymbol{\sigma}' \rightarrow \boldsymbol{\sigma}) P(\boldsymbol{\sigma}'_{eq}) = \sum_{\substack{\boldsymbol{\sigma}' \\ \boldsymbol{\sigma}' \neq \boldsymbol{\sigma}_{eq}}} G(\boldsymbol{\sigma} \rightarrow \boldsymbol{\sigma}') P(\boldsymbol{\sigma}_{eq}), \tag{2.26}$$

where [55, 58]:

$$P(\boldsymbol{\sigma}_{eq}) = Z^{-1} \exp\left(-\frac{E(\boldsymbol{\sigma}_{eq})}{k_B T}\right), \tag{2.27}$$

$$Z = \sum_{\boldsymbol{\sigma}} \exp\left(-\frac{E(\boldsymbol{\sigma})}{k_B T}\right). \tag{2.28}$$

The sum of all the probabilities of the system undergoing a transition from state $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma'}$ 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]:

$$\Delta t = -\frac{1}{\sum_i \sum_j \Gamma_{ij}} \ln(U), \tag{2.29}$$

$$t_{new} = t_{old} + \Delta t, \tag{2.30}$$

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]:

$$\Gamma_{ads,i} = \nu_{ads}\left(1-\sigma_i\right), \tag{2.31}$$

$$\Gamma_{desorb,i} = \nu_{desorb}\sigma_i \exp\left(-\frac{\phi_{binding}(i)}{k_B T}\right). \tag{2.32}$$

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

$$E = \sum_{i}^{N} E_i = \sum_{i}^{N}\sum_{j}^{n_c} J(r_{ij})\delta_{\sigma_i\sigma_j} \qquad \sigma = 0 \text{ or } 1, \tag{2.33}$$

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:

$$\Gamma_{d,i} = \nu_d \exp\left(-\frac{E(\sigma_{init},\sigma_{final})}{k_B T}\right), \tag{2.34}$$

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]:

$$E = \sum_{i}^{N} E_i = \sum_{i}^{N}\sum_{j}^{n_c} J(r_{ij})\delta_{\sigma_i\sigma_j} \qquad \sigma = 0, 1, \ldots, Q. \tag{2.35}$$

In the case of polycrystalline crystal growth, the Hamiltonian would be modified to include

interactions between sites of different states,

$$E = \sum_{i}^{N} E_i = \sum_{i}^{N} \sum_{j}^{n_c} J(r_{ij}, \sigma_i, \sigma_j) \qquad \sigma = 0, 1, \ldots, Q. \tag{2.36}$$

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]:

$$\Gamma_{dep} = \nu_a \exp\left(\frac{E_{dep}}{k_B T}\right), \tag{2.37}$$

$$\Gamma_{diss} = \nu_a \exp\left(-\frac{n_n \phi_{ss}}{k_B T}\right), \tag{2.38}$$

$$\Gamma_{diff} = \nu_d \exp\left(\frac{E_A - E_B}{k_B T}\right), \tag{2.39}$$

where ν_a is the frequency rate of the event (s^{-1}), ϕ_{ss} is the average potential energy between solid atoms (eV), n_n is the number of occupied first nearest neighbors, ν_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.

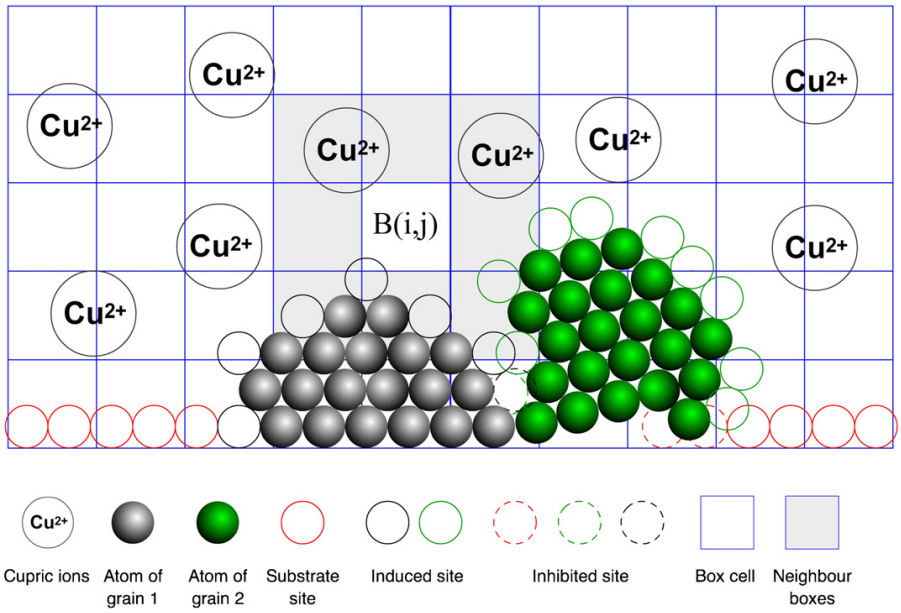


Figure 2.2: Schematics of polycrystalline system modelled in ref. [34].

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 angle associated with them. Figure 2.2 shows an example of this implementation. 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

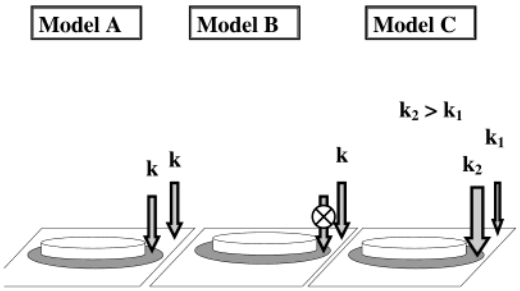


Figure 3.1: Possible mechanisms for atom attachment and their relative rates studied in ref. [17].

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 electrodeposition of polycrystalline systems have been restricted to two-dimensional studies. Liu et al. [34, 35] developed a cross-sectional two-dimensional KMC method for polycrystalline 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.

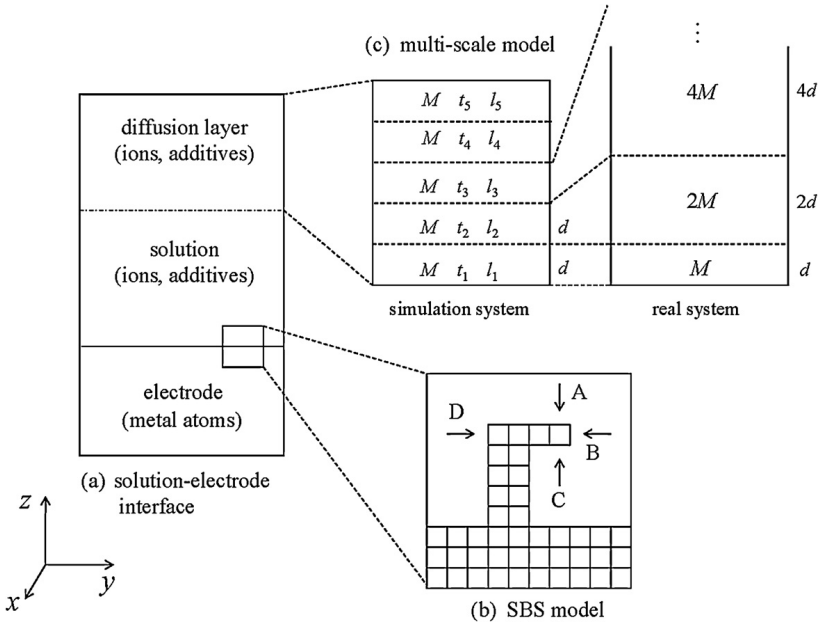


Figure 3.2: Simulation system in ref. [23].

Chapter 4

KMC-EAM Simulations of Electrodeposition onto a Single Crystal Substrate[1]

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 tem-

[1]The material in this chapter forms the basis for the published article T. Treeratanaphitak, M. D. Pritzker, and N. M. Abukhdeir. Kinetic Monte Carlo simulation of electrodeposition using the embedded-atom method. *Electrochim. Acta*, **121**, 407–414, 2014.

perature on surface morphology.

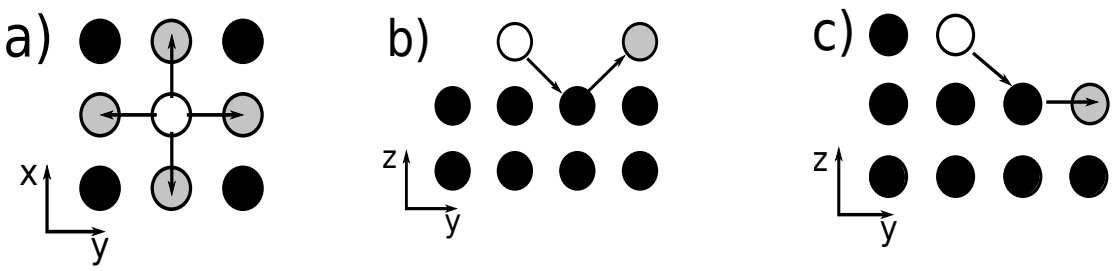
4.1 Methodology

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:

$$Cu^{2+}_{(aq)} + 2e^- \longrightarrow Cu^0_{(s)}. \tag{4.1}$$

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].



(a) (b) (c)

Figure 4.1: Schematics of three possible diffusion mechanisms: a) hopping b) atom exchange and c) step-edge atom exchange. White denotes an adatom, grey denotes the new location of the adatom, and black denotes an occupied site. Note that the atom locations are not drawn to scale.

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 ≤ 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 ΔE 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

Mechanism	Propensity Function	
Deposition	$\Gamma_{i,dep} = \dfrac{i_{dep}}{-zen_{dep}}$	(4.2)
Hopping	$\Gamma_{i,hop} = \begin{cases} \nu_d \exp\left(-\frac{E_{hop}}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_{hop}+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(4.3)
Atom exchange	$\Gamma_{i,exch} = \begin{cases} \nu_d \exp\left(-\frac{E_{exch}}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_{exch}+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(4.4)
Step-edge atom exchange	$\Gamma_{i,step} = \begin{cases} \nu_d \exp\left(-\frac{E_{step}}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_{step}+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(4.5)

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

Parameter	Definition	Value
n_{dep}	number of possible deposition sites per unit area	varies [=] sites m^{-2}
e	elementary charge	1.602×10^{-19} C
z	number of electrons transferred in reduction reaction	2
ν_d	atomic vibrational frequency	2×10^{13} s^{-1}
E_{hop}	hopping activation energy	0.5 eV [43]
E_{exch}	atom exchange activation energy	0.7 eV [43]
E_{step}	step-edge atom exchange activation energy	0.2 eV [43]

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, 2.5×10^4 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.

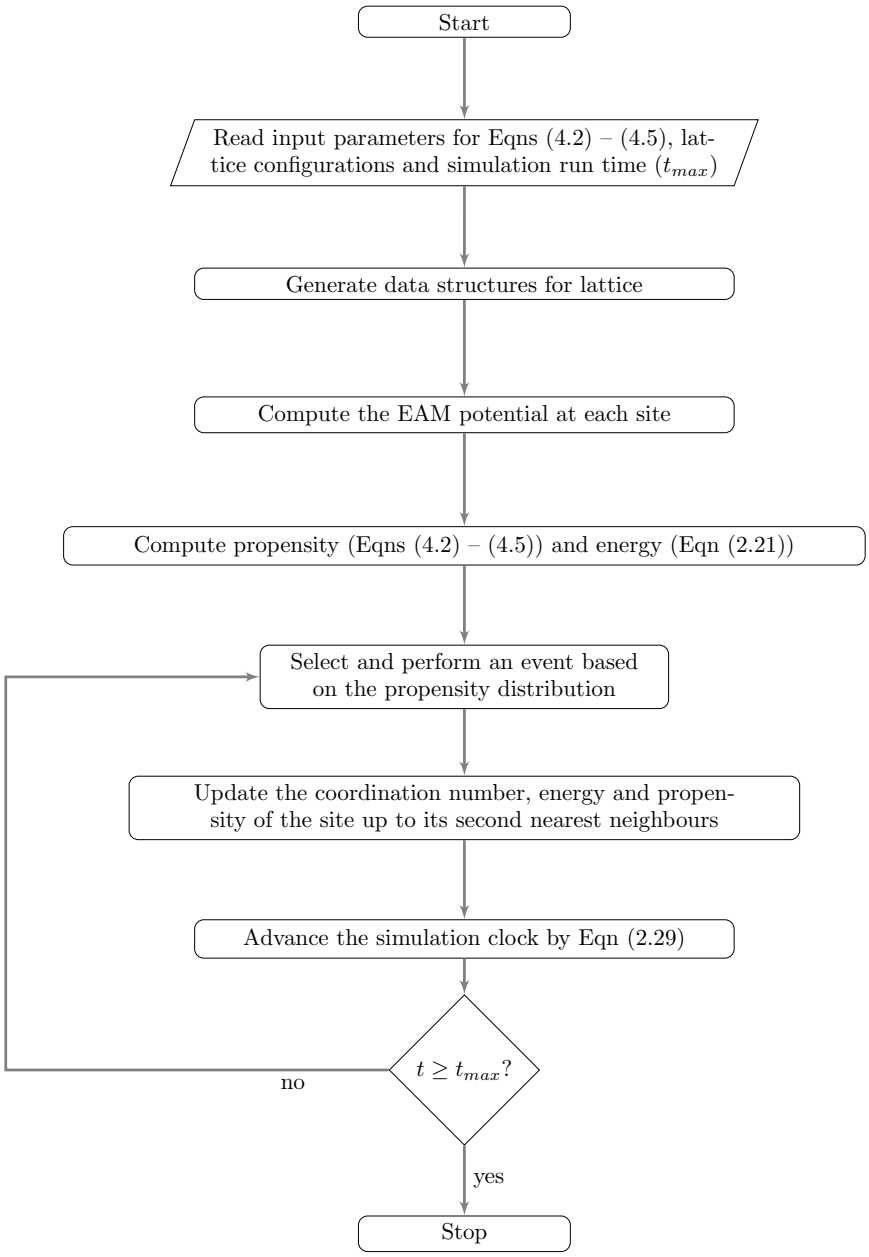


Figure 4.2: Flowchart of KMC-EAM algorithm

4.2 Results and Discussion

KMC-EAM simulations are performed to model electrodeposition of a fixed number (2.5×10^4) 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 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]:

$$i_L = 0.620 z F D^{2/3} w^{1/2} \nu^{-1/6} c. \tag{4.6}$$

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}), ν 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

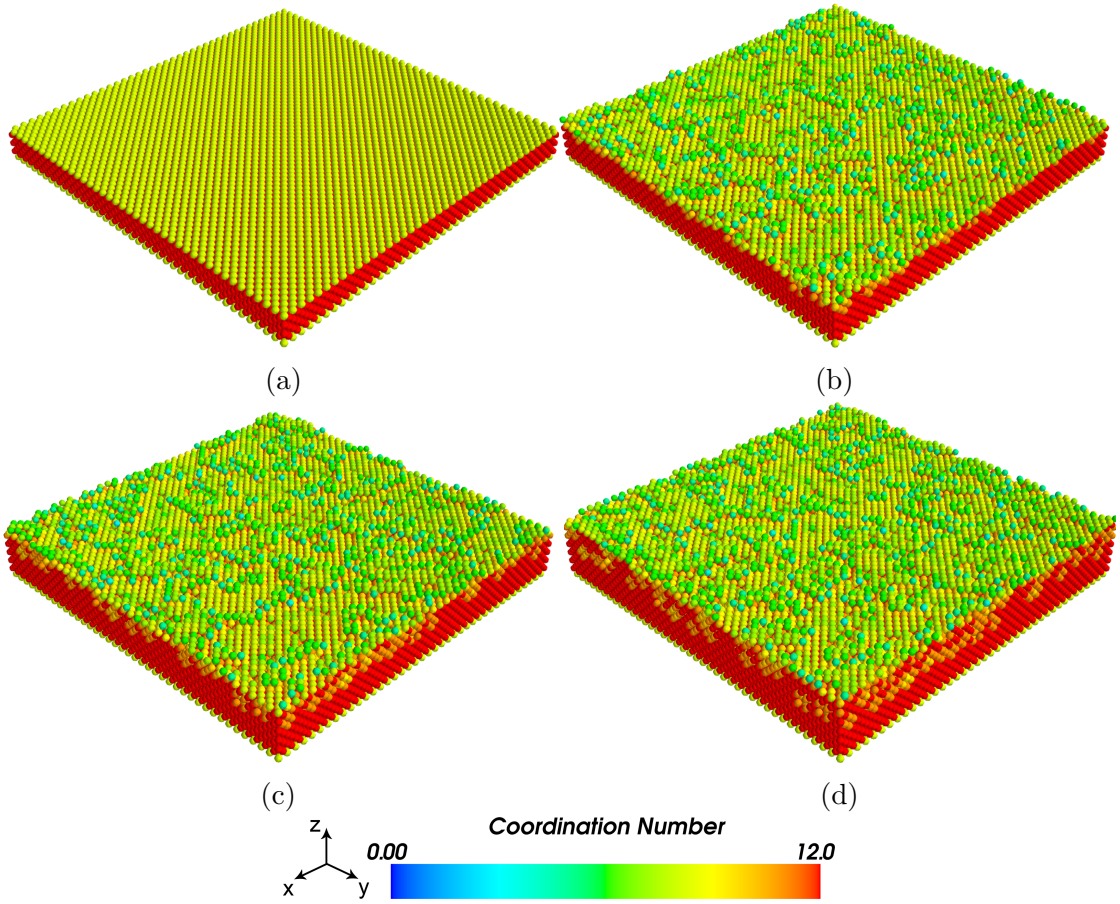


Figure 4.3: Morphology evolution of the configuration from KMC-EAM simulation at a) 0 s, b) 0.1 s, c) 0.2 s, and d) 0.3 s. The current density is -1000 A m^{-2} and the operating temperature is 300 K. Colours denote coordination number (blue to red in ascending order). The substrate surface area is $40a_{Cu} \times 40a_{Cu}$ (≈ 210 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 (δ_E) and average absolute relative coordination number difference (δ_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.

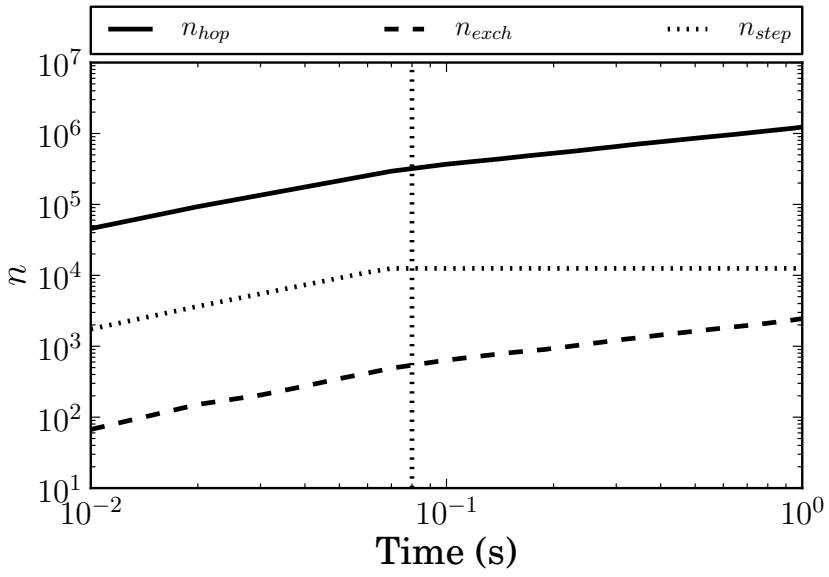
4.2.1 Kinetics of Diffusion Events

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

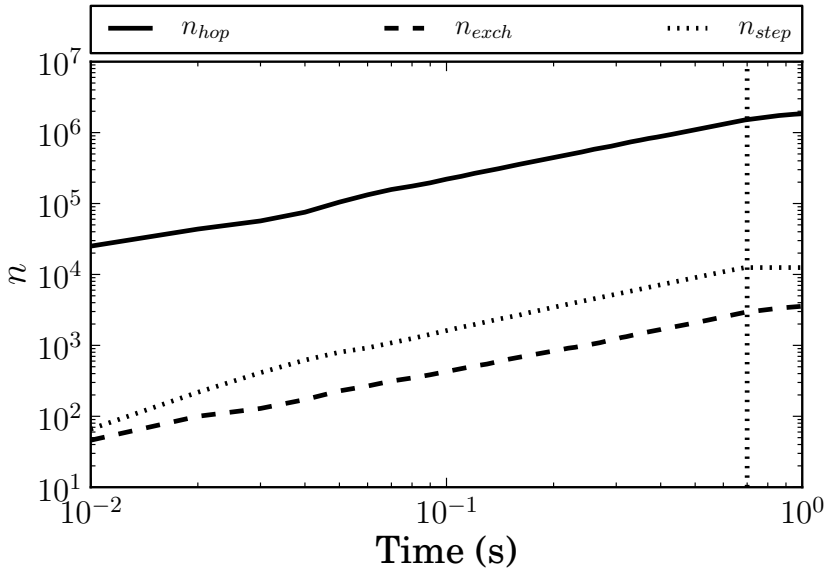
the vertical line in Figure 4.4), the step-edge atom exchange diffusion (Figure 4.1c) ceases in both simulations while both the hopping (Figure 4.1a) and atom exchange (Figure 4.1b) surface diffusion mechanisms persist. During both the initial electrodeposition and relaxation regimes, diffusion events 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_{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_{step} is independent of deposition rate, comparing Figures 4.4a and 4.4b. Instead, the value of n_{step} at any time is related primarily to the total number of atoms deposited up to that point.



(a)



(b)

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.

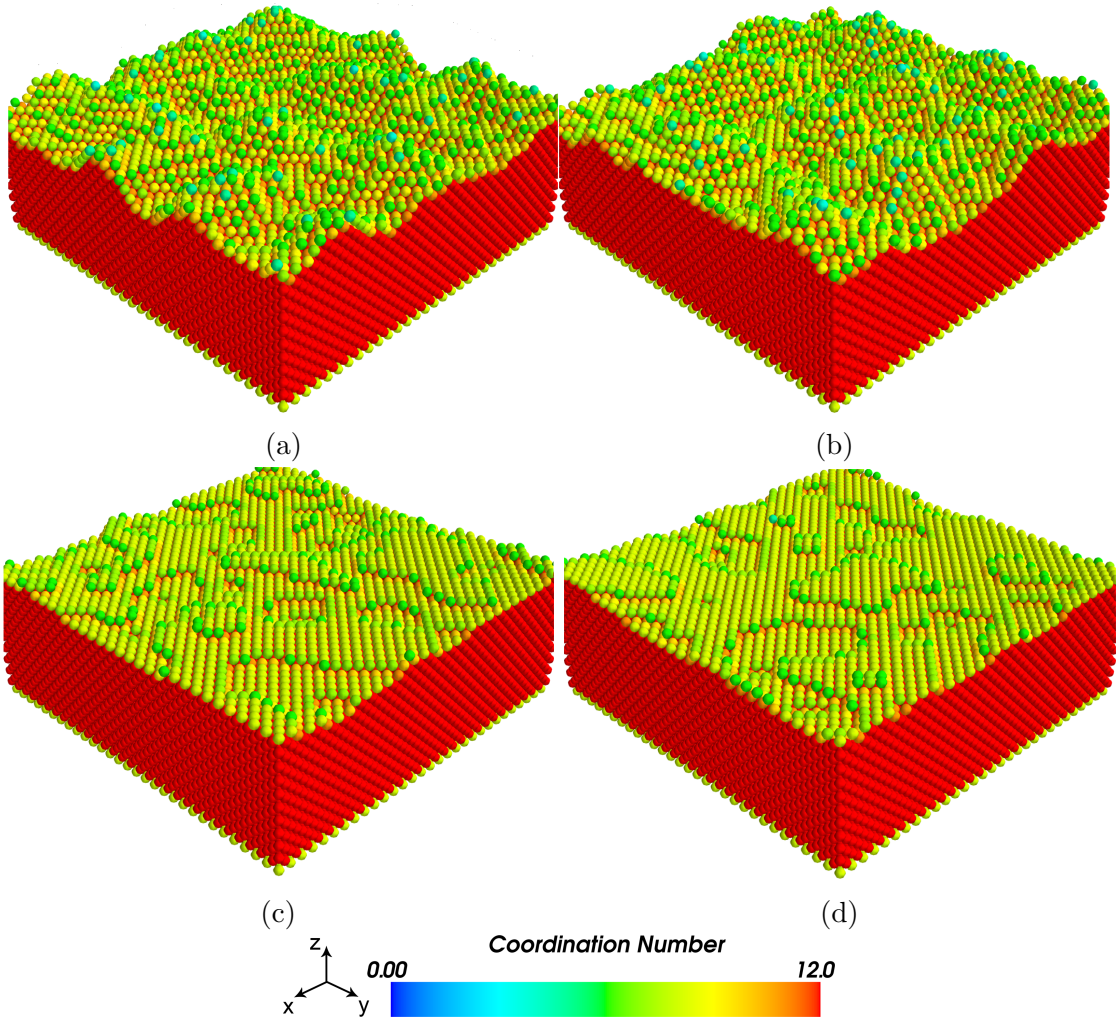


Figure 4.5: Equilibrium morphology at t = 5 s of simulations with a) hopping as the only diffusion mechanism deposited at -1000 A m^{-2}, b) hopping as the only diffusion mechanism deposited at -100 A m^{-2}, c) all 3 diffusion mechanisms deposited at -1000 A m^{-2} and d) all 3 diffusion mechanisms deposited at -100 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 $30a_{Cu} \times 30a_{Cu}$ (≈ 120 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 A m^{-2} and -1000 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 ($\overline{R}_{RMS}$) is calculated using [91]:

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

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 ($\overline{R}_{RMS}$), cluster area fraction ($\overline{A}$), average cluster perimeter ($\overline{P}$) and Euler characteristic ($\overline{\chi}$).

i_{dep} (A m^{-2})	Diffusion Mechanisms	$\overline{R}_{RMS}$ (nm)	$\overline{A}$	$\overline{P}$ (nm)	$\overline{\chi}$ (nm^{-1})
-1000	Hopping	0.279 ± 0.004	0.14 ± 0.00	8.4 ± 0.3	1651.2 ± 17.4
	All	0.156 ± 0.004	0.74 ± 0.00	101.6 ± 43.7	1620.8 ± 21.7
-100	Hopping	0.287 ± 0.000	0.16 ± 0.00	9.8 ± 0.7	1607.7 ± 34.8
	All	0.138 ± 0.004	0.80 ± 0.02	100.1 ± 29.3	1525.2 ± 265.1

acteristic 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 ($\overline{A}$) is the average fraction of the total cluster surface area relative to the total surface area. The average cluster perimeter ($\overline{P}$) is the average perimeter of the clusters in the domain. The average Euler characteristic ($\overline{\chi}$) 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, $\overline{A}$ 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 δ_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 δ_E is non-negligible but reasonable over the full range of applied current densities. The increase of δ_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 δ_E.

The difference in atom coordination number δ_C was also determined in order to compare the KMC-EAM equilibrium configurations to those of MD-EAM. These plots are not shown

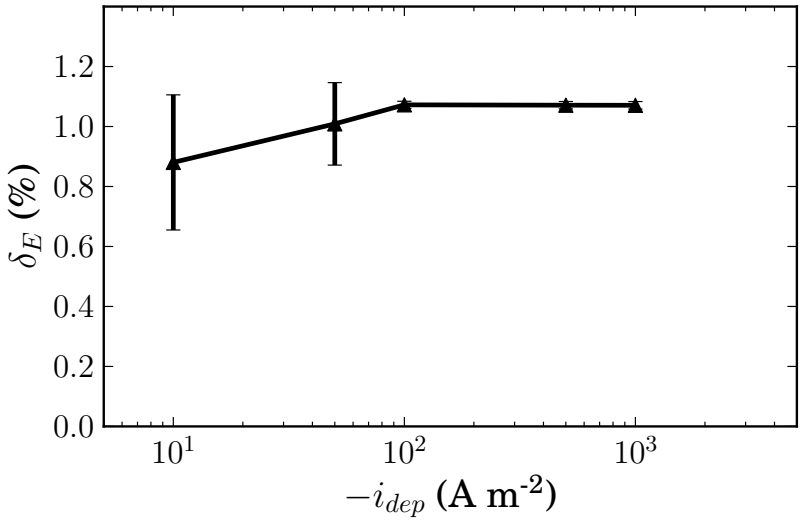


Figure 4.6: Effect of current density on δ_E obtained by MD-EAM and KMC-EAM simulations at 300 K, $h_s = 1.1$ nm and $f_s = 1.0$.

since the values of δ_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 δ_C. Thus, the difference in energy δ_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 δ_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 δ_E is observed as in the previous case with δ_E being non-negligible (ideally, $\delta_E \approx 0$) but small over the full range of operating temperatures. The values of δ_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

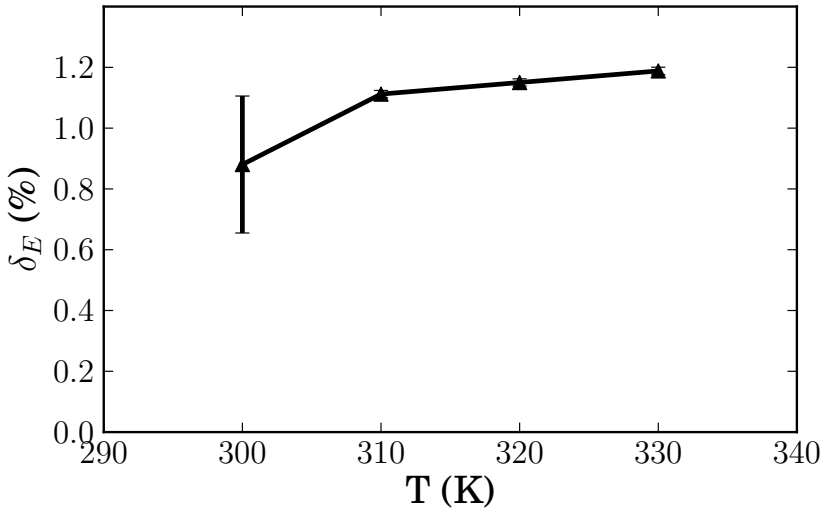


Figure 4.7: Effect of temperature on δ_E obtained by MD-EAM and KMC-EAM simulations at -10 A m^{-2}, $h_s = 1.1$ nm and $f_s = 1.0$.

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 δ_E and δ_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:

$$Cu^{2+}_{(aq)} + 2e^{-} \leftrightharpoons Cu^{0}_{(s)}. \tag{5.1}$$

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}:

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

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:

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

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

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

Mechanism	Propensity Function	
Deposition	$\Gamma_{i,dep} = \dfrac{i_{dep}}{-zen_{dep}}$	(5.6)
Dissolution	$\Gamma_{i,diss} = \dfrac{i_{diss}}{zen_{diss}}$	(5.7)
Hopping	$\Gamma_{i,hop} = \begin{cases} \nu_d \exp\left(-\frac{E_{hop}}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_{hop}+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(5.8)
Atom exchange	$\Gamma_{i,exch} = \begin{cases} \nu_d \exp\left(-\frac{E_{exch}}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_{exch}+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(5.9)
Step-edge atom exchange	$\Gamma_{i,step} = \begin{cases} \nu_d \exp\left(-\frac{E_{step}}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_{step}+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(5.10)
Grain boundary diffusion	$\Gamma_{i,b} = \begin{cases} \nu_d \exp\left(-\frac{E_b}{k_B T}\right) & \Delta E \leq 0 \\ \nu_d \exp\left(-\frac{E_b+\Delta E}{k_B T}\right) & \Delta E > 0 \end{cases}$	(5.11)

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

Parameter	Definition	Value
n_{dep}	number of possible deposition sites per unit area	varies [=] sites m^{-2}
n_{diss}	number of possible dissolution sites per unit area	varies [=] sites m^{-2}
e	elementary charge	1.602×10^{-19} C
z	number of electrons transferred in reduction reaction	2
ν_d	atomic vibrational frequency	2×10^{13} s^{-1}
E_{hop}	hopping activation energy	0.5 eV [43]
E_{exch}	atom exchange activation energy	0.7 eV [43]
E_{step}	step-edge atom exchange activation energy	0.2 eV [43]
E_b	grain boundary diffusion activation energy	0.5 eV [97]

Table 5.3: Parameters used in Eqns (5.5) and (5.4) [50].

Parameter	Value
i^0_{Cu}	107.75 A m^{-2}
α_a	1.08
α_c	0.39

5.2.2 Propensity Scaling

In this study, the surface energies may vary between grains since the substrate is polycrystalline. Subsequently, 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:

$$\Gamma'_{i,dep} = \Gamma_{i,dep}\frac{\Delta E}{E_{sub}}, \tag{5.12}$$

where ΔE 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 ΔE 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:

$$\Gamma'_{i,dep} = \Gamma_{i,dep}\frac{\Delta E}{E_{sub}}s_{dep}, \tag{5.13}$$

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:

$$\overline{\Gamma_{dep}} = \frac{1}{n_{dep}A}\sum_i \Gamma'_{i,dep}, \tag{5.14}$$

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

$$s_{dep} = \frac{\overline{\Gamma_{dep}}}{\Gamma_{i,dep}}. \tag{5.15}$$

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

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

$$\Gamma^{'}_{i,diss} = \Gamma_{i,diss} \frac{E_{sub}}{-\Delta E}. \tag{5.16}$$

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 $E_{sub}/-\Delta E$. The negative sign in front of ΔE is due to the value of ΔE being positive in dissolution processes. The scaling factor is determined based on the same procedure as deposition, but now by making use of $\Gamma^{'}_{i,diss}$ and n_{diss} (defined in Table 5.2):

$$\overline{\Gamma_{diss}} = \frac{1}{n_{diss} A} \sum_{i} \Gamma^{'}_{i,diss}. \tag{5.17}$$

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:

$$\mathbf{x}^{'}_{i} = \mathbf{R}\mathbf{x}_{i} + \mathbf{t}, \tag{5.18}$$

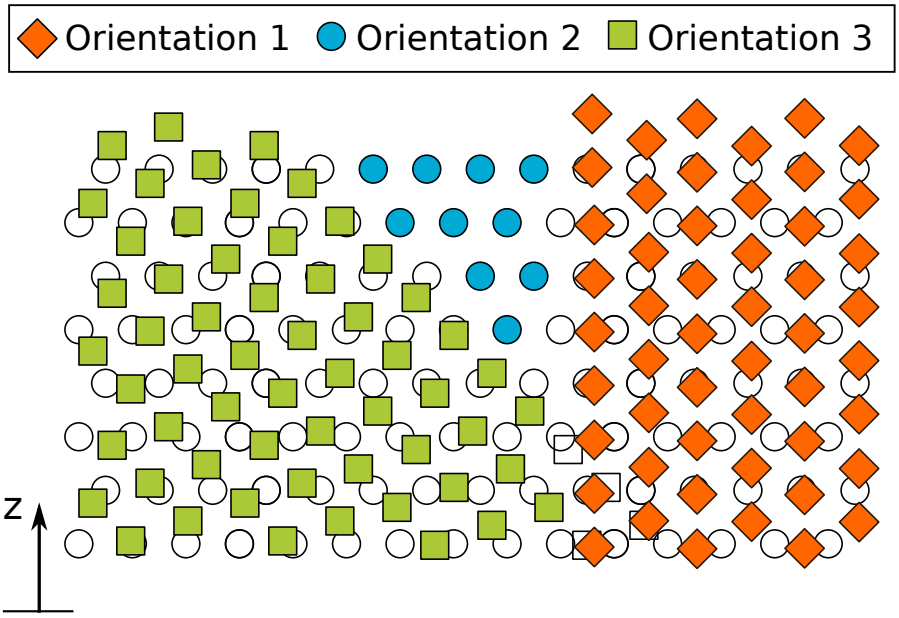


Figure 5.1: Representation of a polycrystalline system in KMC-EAM. Sites that are colored are occupied sites. Occupied sites are not allowed to overlap with occupied sites from another orientation.

where $\mathbf{x}_i^{'}$ is a 3×1 vector representing the new coordinates of site i, $\mathbf{R}$ is a 3×3 rotation matrix, $\mathbf{x}_i$ is a 3×1 vector of reference coordinates for site i and $\mathbf{t}$ is a 3×1 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 $0.75h_s$. All sites above $0.75h_s$ 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 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

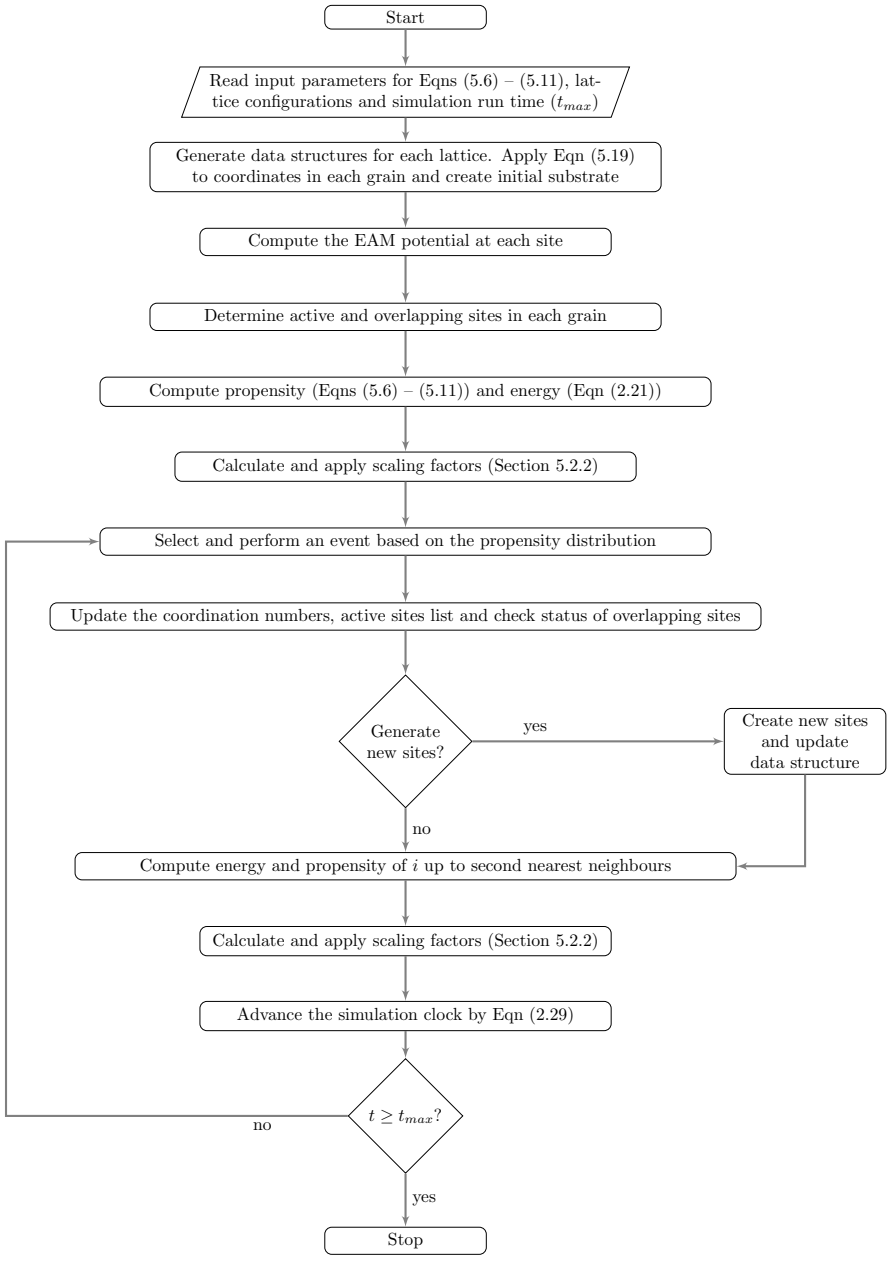


Figure 5.2: Flowchart of polycrystalline KMC-EAM algorithm

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 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 $\eta = -0.10$ V and $\eta = -0.05$ 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.

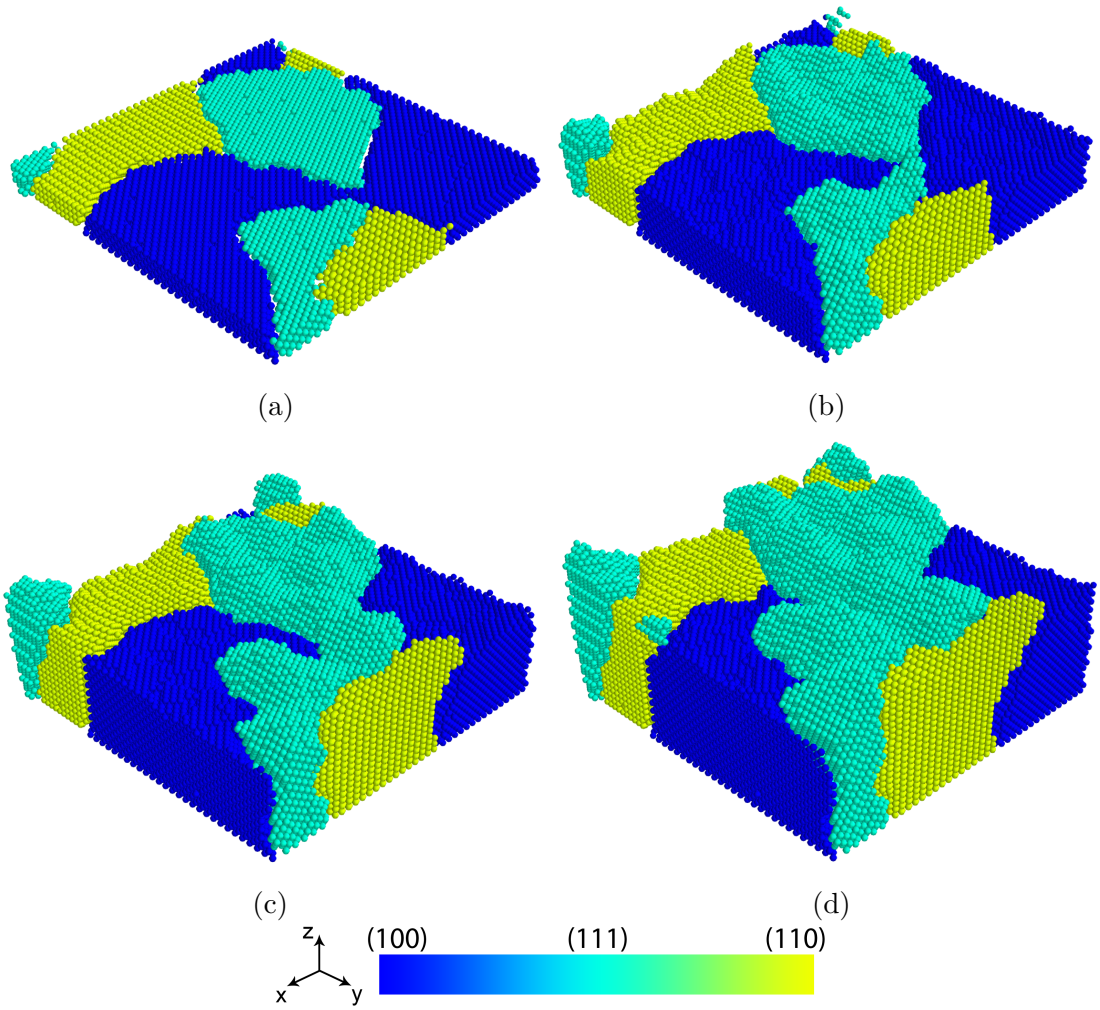


Figure 5.3: Morphology evolution when a) 0 % (the randomly generated substrate), b) 33.3 %, c) 66.7 % and d) 100 % of the 70000 atoms have been deposited at $\eta = -0.15$ V. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu}$ (≈ 210 nm^2).

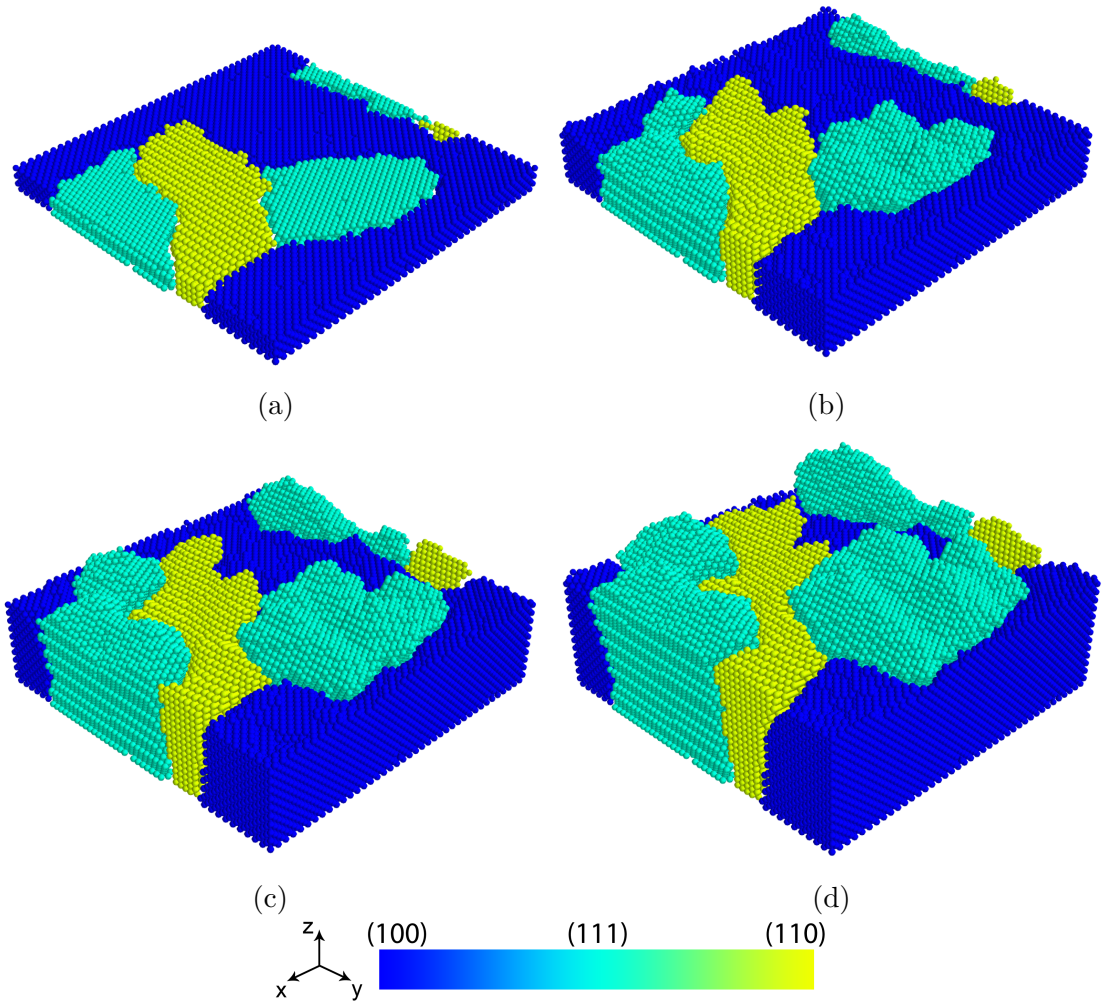


Figure 5.4: Morphology evolution when a) 0 % (the randomly generated substrate), b) 33.3 %, c) 66.7 % and d) 100 % of the 70000 atoms have been deposited at $\eta = -0.10$ V. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu}$ (≈ 210 nm^2).

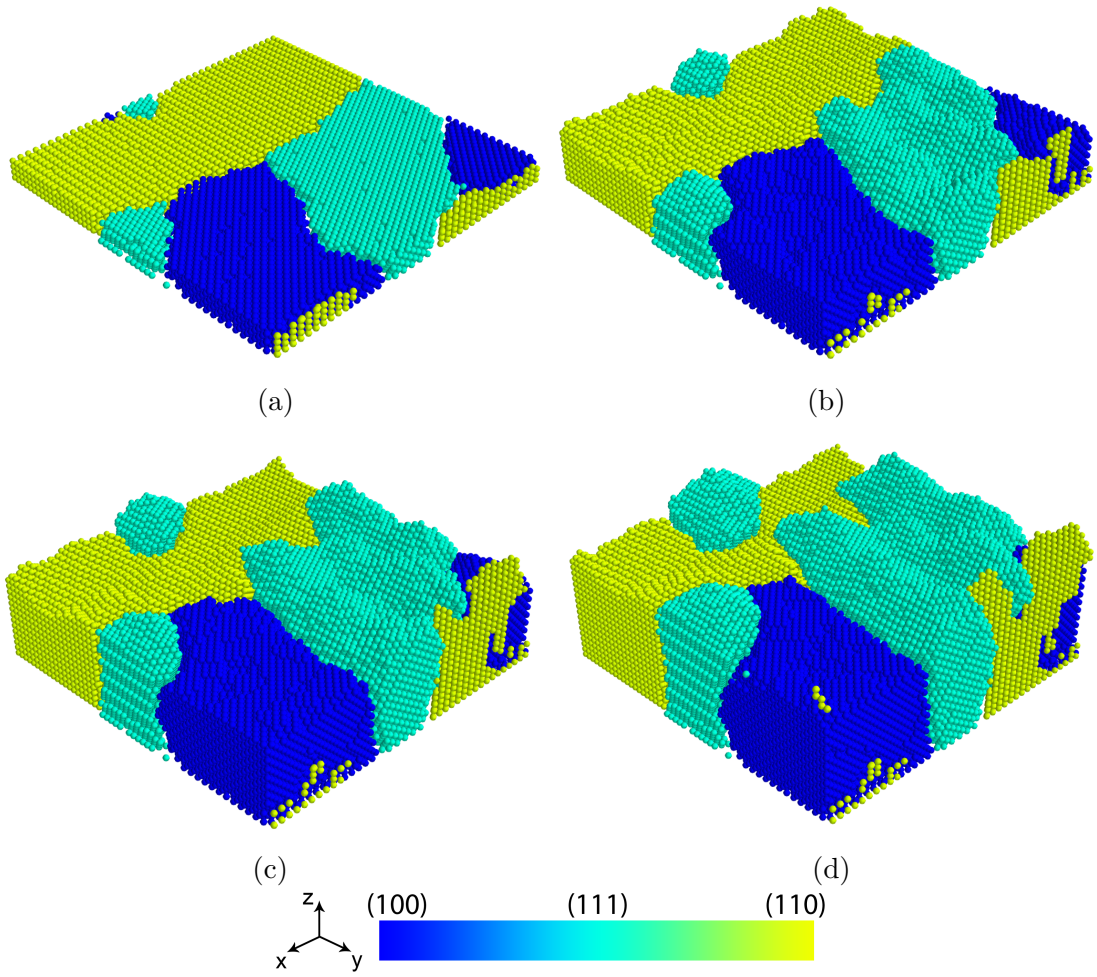


Figure 5.5: Morphology evolution when a) 0 % (the randomly generated substrate), b) 33.3 %, c) 66.7 % and d) 100 % of the 70000 atoms have been deposited at $\eta = -0.05$ V. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu}$ (≈ 210 nm^2).

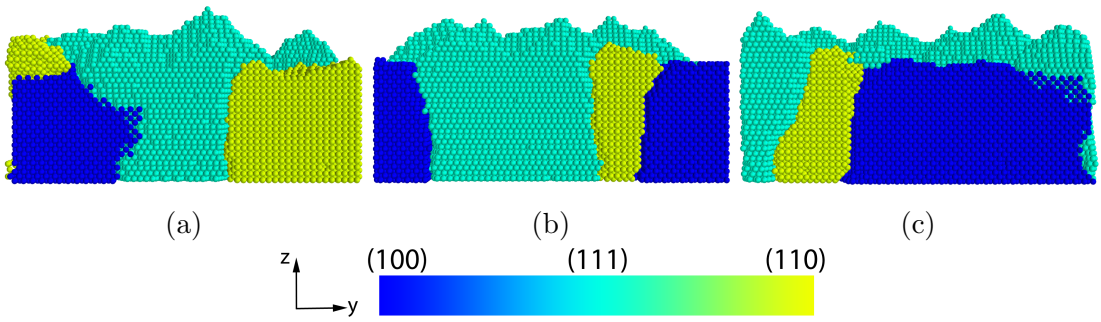
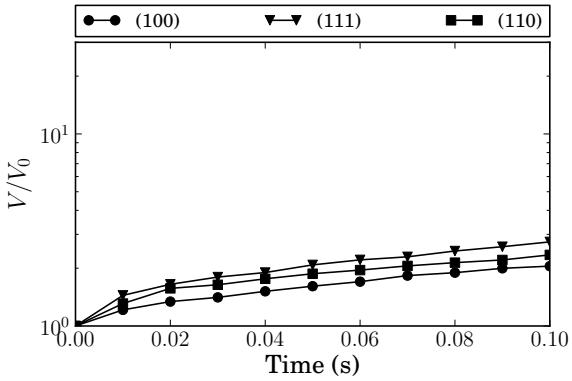


Figure 5.6: Side view of final deposit morphologies at a) $\eta = -0.05$ V, b) $\eta = -0.10$ V and c) $\eta = -0.15$ V. Colours denote grain orientation. The surface area of the substrates are $40a_{Cu} \times 40a_{Cu}$ (≈ 210 nm^2).

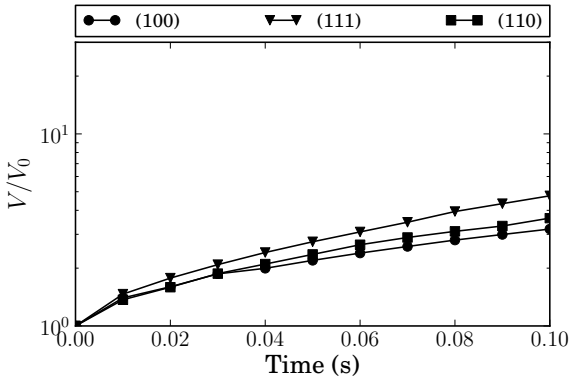
5.3.1 Effect of Domain Size

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.

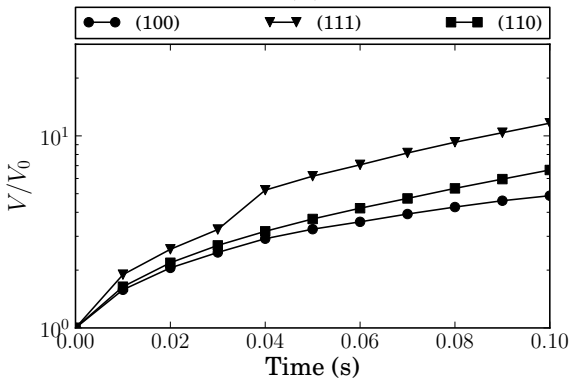
In order to determine this critical (minimum) size, simulations of varying domain sizes were performed. The domain sizes chosen are $20a_{Cu} \times 20a_{Cu} \times 40a_{Cu}$, $25a_{Cu} \times 25a_{Cu} \times 40a_{Cu}$, $30a_{Cu} \times 30a_{Cu} \times 40a_{Cu}$, $35a_{Cu} \times 35a_{Cu} \times 40a_{Cu}$ 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



(a)



(b)



(c)

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.

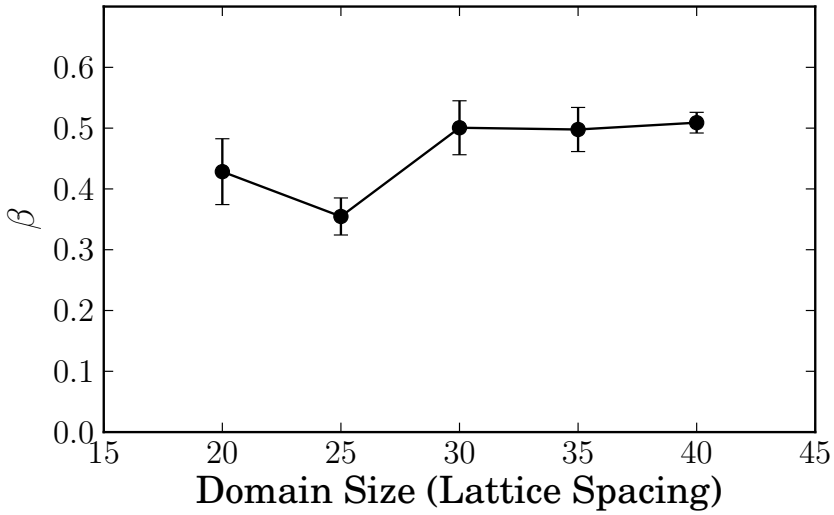


Figure 5.8: Variation of average roughness exponent with domain size. Error bars denote standard deviation.

to follow a power law relationship with time:

$$R_{RMS} = Ct^{\beta}, \tag{5.19}$$

$$\log R_{RMS} = \log C + \beta \log t, \tag{5.20}$$

where C is a constant and β is the power law exponent.

The average value of β 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 $30a_{Cu} \times 30a_{Cu}$ (≈ 120 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-

Table 5.4: Comparison of β values for copper deposition

Type	Study	β	Ref.
Simulation	Polycrystalline KMC-EAM	0.58 ± 0.07	This study
	Multiscale KMC	0.04 ± 0.06	[15]
Experimental	Single Crystal Cu(111) Electrodeposition	0.51	[15]
	Cu Sputtering	0.62 ± 0.07	[111]
	Stirred Cu Electrodeposition	0.45 ± 0.05	[110]

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 A m^2 and -100 A m^2 is in the order of 10^{-2} 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

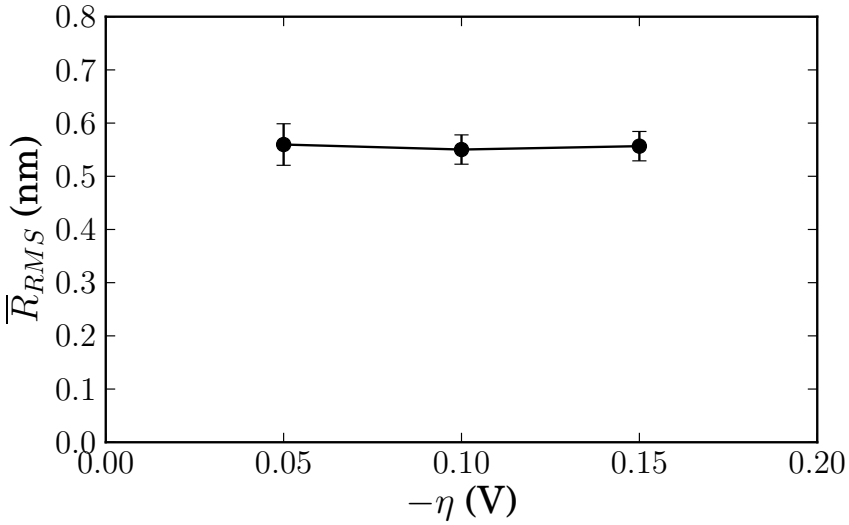


Figure 5.9: Variation of the average final roughness with overpotential. Error bars denote standard deviation.

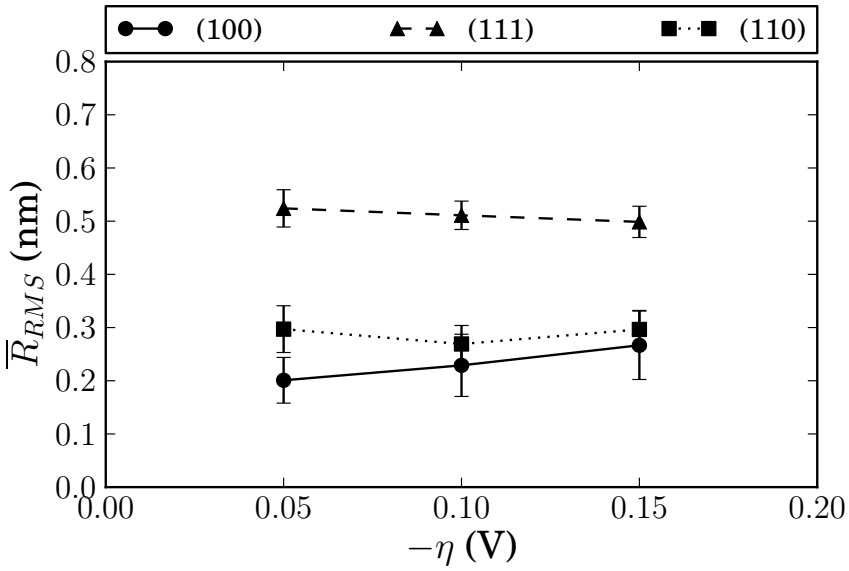


Figure 5.10: Variation of the average final roughness of each grain orientation with overpotential. Error bars denote standard deviation.

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.

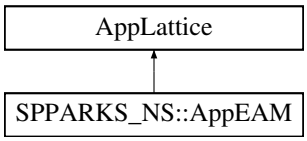
Appendix – KMC-EAM Documentation

The following documentation is generated using Doxygen[2].

SPPARKS_NS::AppEAM Class Reference

`#include <app_eam.h>`

Inheritance diagram for SPPARKS_NS::AppEAM:



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

- 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 denstiy (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)

```
00053                                                   :
00054     AppLattice(spk,narg,arg)
00055 {
00056     ninteger = 4;       // 1) site type, 2) site grain id,
00057                          // 3) coordination number, 4) deposition site indicator
00058     ndouble = 2;        // rho - embedding density, phi - site energy
00059     delpropensity = 1;
00060     delevent = 0;
00061     allow_kmc = 1;
00062     allow_rejection = 0;
00063     allow_masking = 0;
00064     numrandom = 1;
00065
00066     dt_sweep = 1;
00067
00068     seed_n = 0;
00069     count = 0;
00070
00071     pulse = 0;
00072
00073     on_dt = off_dt = prev_time = 0.;
00074
00075     i0 = 0.;
00076     eta = 0.;
00077     alpha_a = alpha_c = 0.;
00078     ox_rate = 0.;
00079     i_diss = i_dep = 0.;
00080
00081     n_hop = n_atomexch = n_step = n_grain = n_boundary = n_diss = 0;
00082
00083     scale_dep_on = 1.;
00084     scale_dep_off = 1.;
00085     scale_diss_on = 1.;
00086     scale_diss_off = 1.;
00087
00088     dep_sites_count = 0;
00089     diss_sites_count = 0;
00090
00091     charge = 1.602176565e-19;   // charge of an electron (C)
00092
00093     // parse arguments
00094     if ((narg < 16) || (strcmp(name,"eam") != 0))
00095         error->all(FLERR,"Illegal app_style command");
```

```
00096
00138     seedlayer_size = atof(arg[1]);
00139     seed_frac = atof(arg[2]);
00140
00141     z_me = atof(arg[4]);
00142     n_max = atoi(arg[5]);
00143     Ed = atof(arg[6]);
00144     Ed_exch = atof(arg[7]);
00145     Ed_step = atof(arg[8]);
00146     v_d = atof(arg[9]);
00147     v_exch = atof(arg[10]);
00148     v_step = atof(arg[11]);
00149     Ed_grain = atof(arg[12]);
00150     Ed_boundary = atof(arg[13]);
00151     v_grain = atof(arg[14]);
00152     v_boundary = atof(arg[15]);
00153
00154     // set up plating conditions
00155     if (strcmp(arg[3], "p") == 0)
00156     {
00157         if (narg < 21)
00158             error->all(FLERR, "Deposition parameters not specified");
00159
00160         eta = atof(arg[16]);
00161         alpha_a = atof(arg[17]);
00162         alpha_c = atof(arg[18]);
00163         i0 = atof(arg[19]);
00164
00165         dep_mode = 1;
00166
00167         if (strcmp(arg[20], "dc") == 0)
00168         {
00169             pulse = 0;
00170             pulse_on = 1;
00171         }
00172
00173         if (strcmp(arg[20], "pp") == 0)
00174         {
00175             if (narg != 23)
00176                 error->all(FLERR, "Pulse-plating conditions required");
00177
00178             pulse = 1;
00179             pulse_on = 1;
00180             on_dt = atof(arg[21]);
00181             off_dt = atof(arg[22]);
00182         }
00183
00184         if (strcmp(arg[20], "pr") == 0)
00185         {
00186             if (narg != 24)
00187                 error->all(FLERR, "Pulse-plating conditions required");
00188
00189             pulse = 2;
00190             pulse_on = 1;
00191             on_dt = atof(arg[21]);
00192             off_dt = atof(arg[22]);
00193             ox_rate = atof(arg[23]);
```

```
00194
00195         }
00196
00197     }
00198     else if (strcmp(arg[3], "g") == 0)
00199     {
00200         if (narg < 18)
00201             error->all(FLERR, "Deposition current density not specified");
00202
00203         i_dep = atof(arg[16]);
00204
00205         dep_mode = 0;
00206
00207         if (strcmp(arg[17], "dc") == 0)
00208         {
00209             pulse = 0;
00210             pulse_on = 1;
00211         }
00212
00213         if (strcmp(arg[17], "pp") == 0)
00214         {
00215             if (narg != 20)
00216                 error->all(FLERR, "Pulse-plating conditions required");
00217
00218             pulse = 1;
00219             pulse_on = 1;
00220             on_dt = atof(arg[18]);
00221             off_dt = atof(arg[19]);
00222         }
00223
00224         if (strcmp(arg[17], "pr") == 0)
00225             error->all(FLERR, "Pulse-reverse is unsupported in galvanostatic mode");
00226     }
00227
00228     if ((seed_frac <= 0.) || (seed_frac > 1.0))
00229         error->all(FLERR,"Fraction must be between 0 and 1.");
00230
00231 }
```

AppEAM::~AppEAM ( ) [virtual]

```
00236 {
00237 }
```

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

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.

```
01122 {
01123   vec_int& type_i = lattice[grain]->iarray[0];
01124   vec_int& dep_flag_i = lattice[grain]->iarray[3];
01125
01126   vec_vec_int& global_neighbors = lattice[grain]->global_neighbors;
01127   vec_int& num_global_neighbors = lattice[grain]->num_global_neighbors;
01128
01129   vec_coordinates& coordinates_i = lattice[grain]->coordinates;
01130
01131   double x_i, y_i, z_i, x_j, y_j, z_j, dx, dy, dz, rsq;
01132   double cutoff = ((lattice[grain]->latconst)/1.01)*((lattice[grain]->latconst)/1.01);
01133
01134   double xmid = domain->midpoint(0);
01135   double ymid = domain->midpoint(1);
01136   double zmid = domain->midpoint(2);
01137
01138   for (int ii = 0; ii < active_sites[grain].size(); ii++)
01139   {
01140     int i = active_sites[grain][ii];
01141
01142     int flag = 0;   // flag indicating whether dep_flag needs to be reverted back from
01143
01144     x_i = coordinates_i[i][0];
01145     y_i = coordinates_i[i][1];
01146     z_i = coordinates_i[i][2];
01147
01148     // only search over global neighbors of site i to speed up the search
01149     // this is ok given the cutoff used here and in finding global neighbors
01150     for (int jj = 0; jj < num_global_neighbors[i]; jj++)
01151     {
01152       int j = global_neighbors[i][jj];
01153       pair_int j_local = global_to_grain(j);
01154
01155       // if global neighbor is in the same grain, ignore
01156       if (j_local.second == grain)
01157         continue;
01158
01159       // if global neighbor not an active site of another grain, ignore
```

```
01160       if (!active_sites_check(j_local.first,j_local.second))
01161         continue;
01162
01163       array_coordinates& coordinates_j = lattice[j_local.second]->coordinates[j_local.first];
01164
01165       x_j = coordinates_j[0];
01166       y_j = coordinates_j[1];
01167       z_j = coordinates_j[2];
01168
01169       dx = x_i - x_j;
01170       dy = y_i - y_j;
01171       dz = z_i - z_j;
01172
01173       if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01174         dx = fabs(dx) - 2.*xmid;
01175
01176       if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01177         dy = fabs(dy) - 2.*ymid;
01178
01179       if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01180         dz = fabs(dz) - 2.*zmid;
01181
01182       rsq = dx*dx + dy*dy + dz*dz;
01183
01184       if (rsq <= cutoff)
01185       {
01186         int& type_j = (lattice[j_local.second]->iarray[0])[j_local.first];
01187         int& dep_flag_j = (lattice[j_local.second]->iarray[3])[j_local.first];
01188
01189         if (type_i[i] != type_j)
01190         {
01191           if (type_i[i] == 0)
01192           {
01193             dep_flag_i[i] = -1;
01194             dep_flag_j = -2;
01195             flag = 1;   // if site i still is inhibited, flag = 1
01196           }
01197           else
01198           {
01199             dep_flag_j = -1;
01200             dep_flag_i[i] = -2;
01201             flag = 1;
01202           }
01203         }
01204         else if ((type_i[i] == 1) && (type_j == 1))
01205         {
01206           dep_flag_i[i] = -2;
01207           dep_flag_j = -2;
01208           flag = 1;   // if site i still is inhibited, flag = 1
01209         }
01210       }
01211     }
01212
01213     // if initially the site has reached the other grain but now has changed
01214     // revert dep_flag back to 0
01215     if (flag == 0)
01216       dep_flag_i[i] = 0;
```

```
01217   }
01218 }
```

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

site	Local index of site that needs checking
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.

```
01227 {
01228   if (!active_sites_check(site,grain))
01229     return;
01230
01231   int& type_i = (lattice[grain]->iarray[0])[site];
01232   int& dep_flag_i = (lattice[grain]->iarray[3])[site];
01233
01234   vec_int& global_neighbors = lattice[grain]->global_neighbors[site];
01235   int& num_global_neighbors = lattice[grain]->num_global_neighbors[site];
01236
01237   double x_i, y_i, z_i, x_j, y_j, z_j, dx, dy, dz, rsq;
01238   double cutoff = ((lattice[grain]->latconst)/1.01)*((lattice[grain]->latconst)/1.01);
01239
01240   double xmid = domain->midpoint(0);
01241   double ymid = domain->midpoint(1);
01242   double zmid = domain->midpoint(2);
01243
01244   array_coordinates& coordinates_i = lattice[grain]->coordinates[site];
01245
01246   x_i = coordinates_i[0];
01247   y_i = coordinates_i[1];
01248   z_i = coordinates_i[2];
01249
01250   int flag = 0;   // flag indicating whether dep_flag needs to be reverted back from -1
01251
01252   // this distance is chosen to avoid large gaps between grains
01253   for (int jj = 0; jj < num_global_neighbors; jj++)
01254   {
01255     int j = global_neighbors[jj];
01256     pair_int j_local = global_to_grain(j);
01257
```

```
01258     // if global neighbor is in the same grain, ignore
01259     if (j_local.second == grain)
01260       continue;
01261
01262     // if global neighbor not an active site of another grain, ignore
01263     if (!active_sites_check(j_local.first,j_local.second))
01264       continue;
01265
01266     array_coordinates& coordinates_j = lattice[j_local.second]->coordinates[j_local.first];
01267
01268     x_j = coordinates_j[0];
01269     y_j = coordinates_j[1];
01270     z_j = coordinates_j[2];
01271
01272     dx = x_i - x_j;
01273     dy = y_i - y_j;
01274     dz = z_i - z_j;
01275
01276     if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01277       dx = fabs(dx) - 2.*xmid;
01278
01279     if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01280       dy = fabs(dy) - 2.*ymid;
01281
01282     if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01283       dz = fabs(dz) - 2.*zmid;
01284
01285     rsq = dx*dx + dy*dy + dz*dz;
01286
01287     if (rsq <= cutoff)
01288     {
01289       int& type_j = (lattice[j_local.second]->iarray[0])[j_local.first];
01290       int& dep_flag_j = (lattice[j_local.second]->iarray[3])[j_local.first];
01291
01292       // if sites are less than the cutoff distance away and one of them is
01293       // unoccupied, the unoccupied site is inhibited
01294       if (type_i != type_j)
01295       {
01296         if (type_i == 0)
01297         {
01298           dep_flag_i = -1;
01299           dep_flag_j = -2;
01300           flag = 1;   // if site i still is inhibited, flag = 1
01301         }
01302         else
01303         {
01304           dep_flag_j = -1;
01305           dep_flag_i = -2;
01306           flag = 1;
01307         }
01308       }
01309       else if ((type_i == 1) && (type_j == 1))
01310       {
01311         dep_flag_i = -2;
01312         dep_flag_j = -2;
01313         flag = 1;   // if site i still is inhibited, flag = 1
01314       }
```

```
01315     }
01316   }
01317
01318   // if initially the site has reached the other grain but now has changed
01319   // revert dep_flag back to 0
01320   if (flag == 0)
01321     dep_flag_i = 0;
01322 }
```

void AppEAM::coord_update (int *i*, int *grain*) [private]

Updates the coordination number of site i

Parameters

i	Local index of site i
grain	Grain ID of site i

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

```
01583 {
01584   vec_int& type = lattice[grain]->iarray[0];
01585   vec_int& coordnum = lattice[grain]->iarray[2];
01586
01587   vec_int& num_neighbors = lattice[grain]->num_neighbors;
01588   vec_vec_int& neighbors = lattice[grain]->neighbors;
01589
01590   coordnum[i] = 0;
01591
01592   for (int jj = 0; jj < num_neighbors[i]; jj++)
01593   {
01594     int j = neighbors[i][jj];
01595     coordnum[j] = 0;
01596     for (int k = 0; k < num_neighbors[j]; k++)
01597     {
01598       if (type[neighbors[j][k]] != 0)
01599         coordnum[j]++;
01600     }
01601
01602     if (type[j] != 0)
01603       coordnum[i]++;
01604   }
01605 }
```

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

i	Local index of site i
grain	Grain ID of site i

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

```
01616 {
01617   vec_int& dep_flag = lattice[grain]->iarray[3];
01618
01619   double xmid = domain->midpoint(0);
01620   double ymid = domain->midpoint(1);
01621   double zmid = domain->midpoint(2);
01622
01623   vec_int direction(6,0);   // create 6x1 vector and initialize all values to zero
01624
01625   double x_loc, y_loc, z_loc;
01626   double j_x_loc, j_y_loc, j_z_loc;
01627   double dx, dy, dz;
01628
01629   vec_coordinates& coordinates = lattice[grain]->coordinates;
01630
01631   vec_vec_int& neighbors = lattice[grain]->neighbors;
01632   vec_int& num_neighbors = lattice[grain]->num_neighbors;
01633
01634   vec_int& coordnum = lattice[grain]->iarray[2];
01635
01636   vec_int& type = lattice[grain]->iarray[0];
01637
01638   // get local coordinates of site i
01639   x_loc = get_local_coordinates(coordinates[i], grain, 0);
01640   y_loc = get_local_coordinates(coordinates[i], grain, 1);
01641   z_loc = get_local_coordinates(coordinates[i], grain, 2);
01642
01650   for (int jj = 0; jj < num_neighbors[i]; jj++)
```

```
01651   {
01652     int j = neighbors[i][jj];
01653
01654     // if site is not vacant, skip
01655     if (type[j] != 0)
01656       continue;
01657
01658     // if dep_flag = -1 (vacant, on a grain boundary) - skip since we don't want
01659     // sites in the grain boundary to undergo atom exchange
01660     if (dep_flag[j] == -1)
01661       continue;
01662
01663     // if bulk, ignore
01664     if (coordnum[j] > 6)
01665       continue;
01666
01667     // get j's local coordinates
01668     j_x_loc = get_local_coordinates(coordinates[j], grain, 0);
01669     j_y_loc = get_local_coordinates(coordinates[j], grain, 1);
01670     j_z_loc = get_local_coordinates(coordinates[j], grain, 2);
01671
01672     // get the distance away from site i in terms of local coordinates
01673     dx = x_loc - j_x_loc;
01674     dy = y_loc - j_y_loc;
01675     dz = z_loc - j_z_loc;
01676
01677     // if periodic boundary condition is used, make sure dx, dy, and dz are correct
01678     if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01679       dx = fabs(dx) - 2.*xmid;
01680
01681     if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01682       dy = fabs(dy) - 2.*ymid;
01683
01684     if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01685       dz = fabs(dz) - 2.*zmid;
01686
01687     // every time dx, dy, dz is not zero, add it to the count in direction array
01688     // store them in the reverse order so that when we use max_element and there's
01689     // a case where two elements are equal, it will favor the z-direction
01690     if ((dx > 0.) && (fabs(dx) > 0.001))
01691       direction[5]++;
01692
01693     if ((dx < 0.) && (fabs(dx) > 0.001))
01694       direction[4]++;
01695
01696     if ((dy > 0.) && (fabs(dy) > 0.001))
01697       direction[3]++;
01698
01699     if ((dy < 0.) && (fabs(dy) > 0.001))
01700       direction[2]++;
01701
01702     if ((dz > 0.) && (fabs(dz) > 0.001))
01703       direction[1]++;
01704
01705     if ((dz < 0.) && (fabs(dz) > 0.001))
01706       direction[0]++;
01707
```

```
01708   }
01709
01710   int sum = std::accumulate(direction.begin(), direction.end(), 0);
01711
01721   // find where the max element in direction vector is (max_element returns an iterator)
01722   int loc = std::distance(direction.begin(), std::max_element(direction.begin(),
    direction.end()));
01723
01724   loc = 5 - loc;   // converting back to x->z ordering
01725
01726   // if there are no sites that satisfy the previous criteria, site i is at the grain
01727   // boundary
01728   if (sum == 0)
01729     loc = -1;
01730
01731   return loc;
01732 }
```

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

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

Parameters

i	Local index of site i
grain	Grain ID of site i

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.

```
01329 {
01330   vec_int& coordnum = lattice[grain]->iarray[2];
01331
01332   vec_coordinates& coordinates = lattice[grain]->coordinates;
01333
01334   double x = coordinates[i][0];
01335   double y = coordinates[i][1];
01336
01337   double x_max = domain->boxhi[0];
01338   double y_max = domain->boxhi[1];
01339
01340   // max distance away from the box boundary to be considered an edge site
01341   double max_dist = (lattice[grain]->next_site)*(lattice[grain]->latconst);
01342
01343   // if the site is at the edge of the box at the x-axis and is non-periodic in x
01344   if (((x <= max_dist) || (fabs(x - x_max) <= max_dist))
01345     && (domain->periodicity[0] == 0))
01346   {
01347     if (coordnum[i] >= 5)   // coordination number of edge sites in the bulk >= 4
01348       return 1;
01349     else if (((y <= max_dist) || (fabs(y - y_max) <= max_dist)) && (coordnum[i] >= 3))
01350     // unless it is at the corner, in that case coordnum >= 3
```

```
01351       return 1;
01352     else
01353       return 0;
01354   }
01355   else if (((y <= max_dist) || (fabs(y - y_max) <= max_dist))
01356       && (domain->periodicity[1] == 0))
01357   {
01358     if (coordnum[i] >= 5)
01359       return 1;
01360     else if (((x <= max_dist) || (fabs(x - x_max) <= max_dist)) && (coordnum[i] >= 3))
01361       return 1;
01362     else
01363       return 0;
01364   }
01365   else
01366     return 0;
01367
01368 }
```

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

n	Local index of site n
grain	Grain ID of site n
direction	Direction that step-edge is occuring

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.

```
01741 {
01749   vec_int& type = lattice[grain]->iarray[0];
01750   vec_coordinates& coordinates = lattice[grain]->coordinates;
01751
01752   // if site is edge of box, return 1 since the rest of the criteria will not apply
01753   if (coordinates[n][floor(direction*0.5)] < (lattice[grain]->next_site)
    *((lattice[grain]->latconst)/1.01))
01754     return 1;
01755
01756   vec_int& coordnum = lattice[grain]->iarray[2];
01757
```

```
01758   // if coordination number is not greater than 4 and not at the edge, the site cannot
01759   // be a potential destination for step-edge atom exchange
01760   if ((coordnum[n] < 5) && (coordinates[n][floor(direction*0.5)] >=
    (lattice[grain]->next_site)*((lattice[grain]->latconst)/1.01)))
01761     return 0;
01762
01763   vec_int& neighbors = lattice[grain]->neighbors[n];
01764   int& num_neighbors = lattice[grain]->num_neighbors[n];
01765
01766   double dx, dy, dz;
01767   int neigh_count = 0;
01768
01769   double xmid = domain->midpoint(0);
01770   double ymid = domain->midpoint(1);
01771   double zmid = domain->midpoint(2);
01772
01773   for (int jj = 0; jj < num_neighbors; jj++)
01774   {
01775     int j = neighbors[jj];
01776
01777     if (direction == 0)
01778     {
01779       dx = coordinates[n][0] - coordinates[j][0];
01780
01781       if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01782         dx = fabs(dx) - 2.*xmid;
01783
01784       if ((dx < 0.) && (fabs(dx) > 0.001) && (type[j] == 1))
01785         neigh_count++;
01786     }
01787     else if (direction == 1)
01788     {
01789       dx = coordinates[n][0] - coordinates[j][0];
01790
01791       if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01792         dx = fabs(dx) - 2.*xmid;
01793
01794       if ((dx > 0.) && (fabs(dx) > 0.001) && (type[j] == 1))
01795         neigh_count++;
01796     }
01797     else if (direction == 2)
01798     {
01799       dy = coordinates[n][1] - coordinates[j][1];
01800
01801       if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01802         dy = fabs(dy) - 2.*ymid;
01803
01804       if ((dy < 0.) && (fabs(dy) > 0.001) && (type[j] == 1))
01805         neigh_count++;
01806     }
01807     else if (direction == 3)
01808     {
01809       dy = coordinates[n][1] - coordinates[j][1];
01810
01811       if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01812         dy = fabs(dy) - 2.*ymid;
01813
```

```
01814       if ((dy > 0.) && (fabs(dy) > 0.001) && (type[j] == 1))
01815         neigh_count++;
01816     }
01817     else if (direction == 4)
01818     {
01819       dz = coordinates[n][2] - coordinates[j][2];
01820
01821       if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01822         dz = fabs(dz) - 2.*zmid;
01823
01824       if ((dz < 0.) && (fabs(dz) > 0.001) && (type[j] == 1))
01825         neigh_count++;
01826     }
01827     else if (direction == 5)
01828     {
01829       dz = coordinates[n][2] - coordinates[j][2];
01830
01831       if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01832         dz = fabs(dz) - 2.*zmid;
01833
01834       if ((dz > 0.) && (fabs(dz) > 0.001) && (type[j] == 1))
01835         neigh_count++;
01836     }
01837   }
01838   // if the 4 nearest neighbors above site n are vacant, return 1
01839   if (neigh_count == 4)
01840     return 1;
01841
01842   return 0;
01843
01844 }
```

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 {
```

```
00565   pair_int site_local;
00566
00567   int seed_count = 0;  // counter for number of sites occupied in seed layer
00568   int ind = 0;
00569   double einitial;
00570   double efinal;
00571   int flag;
00572
00573   if (seed_frac != 1.0)
00574   // if the desired fraction is not 1, randomly assign atoms to sites in the seed layer
00575   {
00576     double xmid = domain->midpoint(0);
00577     double ymid = domain->midpoint(1);
00578     double zmid = domain->midpoint(2);
00579
00580     double dx, dy, dz, rsq;
00581     double cutoff = ((lattice[0]->latconst)/1.01)*((lattice[0]->latconst)/1.01);
00582
00583     while (seed_count <= seed_n)
00584     {
00585       // select site using random number
00586       ind = int (random->uniform()*seedlayer.size());
00587
00588       if (ind == seedlayer.size())
00589       // making sure that the index will not exceed the dim. of array
00590         ind--;
00591
00592 flag = 0;
00593       site_local = global_to_grain(seedlayer[ind]);
00594
00595       int& type_i = (lattice[site_local.second]->iarray[0])[site_local.first];
00596       vec_int& num_global_neighbors = lattice[site_local.second]->num_global_neighbors;
00597       vec_vec_int& global_neighbors = lattice[site_local.second]->global_neighbors;
00598
00599       array_coordinates& coordinates_site = lattice[site_local.second]->
    coordinates[site_local.first];
00600
00601       // check global neighbors within cutoff to prevent overlap
00602       for (int i = 0; i < num_global_neighbors[site_local.first]; i++)
00603       {
00604         int j = global_neighbors[site_local.first][i];
00605         pair_int j_local = global_to_grain(j);
00606
00607         array_coordinates& coordinates_j = lattice[j_local.second]->coordinates[j_local.first];
00608
00609         dx = coordinates_site[0] - coordinates_j[0];
00610         dy = coordinates_site[1] - coordinates_j[1];
00611         dz = coordinates_site[2] - coordinates_j[2];
00612
00613         if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
00614           dx = fabs(dx) - 2.*xmid;
00615
00616         if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
00617           dy = fabs(dy) - 2.*ymid;
00618
00619         if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
00620           dz = fabs(dz) - 2.*zmid;
```

```
00621
00622         rsq = dx*dx + dy*dy + dz*dz;
00623
00624         if (rsq >= cutoff)
00625           continue;
00626
00627         int& type_j = (lattice[j_local.second]->iarray[0])[j_local.first];
00628
00629         // if one of the site's neighbors from a different grain is occupied
00630         // this site has to be unoccupied (same idea as check_grain_distance)
00631         if ((type_j == 1) && (site_local.second != j_local.second))
00632           flag = 1;   // flip the flag to prevent deposition
00633       }
00634
00635       // if no issues, try depositing a site here
00636       if (flag == 0)
00637       {
00638         einitial = system_energy(site_local.first,-1,0,site_local.second,site_local.second);
00639         // change type to occupied
00640         type_i = 1;
00641
00642         efinal = system_energy(site_local.first,-1,1,site_local.second,site_local.second);
00643
00644         // check energy
00645         if (efinal <= einitial)
00646         {
00647           seed_count++;
00648           efinal = system_energy(site_local.first,-1,2,site_local.second,site_local.second);
00649         }
00650         else  // else, revert type back
00651         {
00652           type_i = 0;
00653         }
00654       }
00655     }
00656   }
00657   else
00658   {
00659     // 'grow' the seed up to seedlayer_size then polish it back down to 0.75*height
00660
00661     int seed_sites = lattice_count*2;   // get the number of seed sites
00662
00663     int seed_sites_count = 0;
00664
00665     int next_grain = 0; // grain ID of next site - to ensure equal distribution of seeds
00666
00667     vec_int occ_sites;  // vector containing occupied sites in seed
00668
00669     while (seed_sites_count < seed_sites)
00670     {
00671       ind = int (random->uniform()*seedlayer.size());  // pick a site
00672
00673       if (ind == seedlayer.size())
00674         // making sure that the index will not exceed the dim. of array
00675         ind--;
00676
00677       site_local = global_to_grain(seedlayer[ind]);
```

```
00678
00679       array_coordinates& coordinates_i = lattice[site_local.second]->
    coordinates[site_local.first];
00680       int& type_i = (lattice[site_local.second]->iarray[0])[site_local.first];
00681
00682       // if the site is at the bottom z limit of box, plant the seed
00683       if ((coordinates_i[2] <= (lattice[site_local.second]->next_site)*
    ((lattice[site_local.second]->latconst)/1.01)) &&
00684       (site_local.second == next_grain))
00685       {
00686         type_i = 1;
00687         seed_sites_count++;
00688         occ_sites.push_back(seedlayer[ind]);  // add to occ_sites
00689         seedlayer.erase(seedlayer.begin() + ind);   // remove from seedlayer
00690         coord_update(site_local.first,site_local.second);   // update coordnum
00691         // add to active_sites
00692         active_sites[site_local.second].push_back(site_local.first);
00693
00694         vec_int& num_neighbors = lattice[site_local.second]->num_neighbors;
00695         vec_vec_int& neighbors = lattice[site_local.second]->neighbors;
00696
00697
00698         check_grain_distance(site_local.first,site_local.second);
00699
00700         // add the nearest neighbors to active_sites as well
00701         for (int jj = 0; jj < num_neighbors[site_local.first]; jj++)
00702         {
00703           int j = neighbors[site_local.first][jj];
00704           active_sites[site_local.second].push_back(j);
00705           check_grain_distance(j,site_local.second);
00706         }
00707
00708         // update dep_flag
00709         update_dep_flag(site_local.second);
00710
00711         next_grain++;   // move on to next grain
00712       }
00713       // if looped over all grains, start over
00714       if (next_grain == lattice_count)
00715         next_grain = 0;
00716     }
00717     // now fill in the rest based on the seed sites
00718     // let it loop over occ_sites.size() - this will grow as the number of occ_sites
00719     // increases
00720     int k = 0;
00721     while (k < occ_sites.size())
00722     {
00723       int site = occ_sites[k];
00724
00725       site_local = global_to_grain(site);
00726
00727       vec_int& num_neighbors = lattice[site_local.second]->num_neighbors;
00728       vec_vec_int& neighbors = lattice[site_local.second]->neighbors;
00729
00730       array_coordinates& coordinates_site = lattice[site_local.second]->
    coordinates[site_local.first];
00731
```

```
00732       // loop over all first nearest neighbors
00733       for (int j = 0; j < num_neighbors[site_local.first]; j++)
00734       {
00735         // get global index to find site in seedlayer
00736         int neigh_global = grain_to_global(std::make_pair(neighbors[site_local.first][j],
    site_local.second));
00737
00738         auto ind = std::find(seedlayer.begin(), seedlayer.end(), neigh_global);
00739
00740         // if the neighbor is in the seedlayer vector, it's a seed site
00741         if (ind != seedlayer.end())
00742         {
00743           site_local = global_to_grain(*ind);
00744
00745           int dep_flag_i = lattice[site_local.second]->iarray[3][site_local.first];
00746
00747           // check dep_flag to prevent overlaps and to avoid computing the
00748           // energy (sites that will cause efinal > einitial are excluded by
00749           // dep_flag)
00750           if (dep_flag_i != 1)
00751             continue;
00752
00753           einitial = system_energy(site_local.first,-1,0,site_local.second,site_local.second);
00754
00755           // change type to 1
00756           lattice[site_local.second]->iarray[0][site_local.first] = 1;
00757
00758           efinal = system_energy(site_local.first,-1,1,site_local.second,site_local.second);
00759
00760           if (efinal > einitial)
00761           {
00762             lattice[site_local.second]->iarray[0][site_local.first] = 0;
00763             seedlayer.erase(ind);
00764             continue;
00765           }
00766
00767           occ_sites.push_back(*ind);
00768           seedlayer.erase(ind);
00769           coord_update(site_local.first,site_local.second);
00770
00771           // if it was not in active_sites, add it
00772           if (!active_sites_check(site_local.first, site_local.second))
00773             active_sites[site_local.second].push_back(site_local.first);
00774
00775           check_grain_distance(site_local.first,site_local.second);
00776
00777           for (int jj = 0; jj < num_neighbors[site_local.first]; jj++)
00778           {
00779             int j = neighbors[site_local.first][jj];
00780             if (!active_sites_check(j, site_local.second))
00781             {
00782               active_sites[site_local.second].push_back(j);
00783               check_grain_distance(j,site_local.second);
00784             }
00785           }
00786
00787           update_dep_flag(site_local.second);
```

```
00788
00789         }
00790       }
00791       k++;
00792     }
00793     // now that we have the substrate, we 'polish' it
00794     for (int i = 0; i < occ_sites.size(); i++)
00795     {
00796       site_local = global_to_grain(occ_sites[i]);
00797
00798       array_coordinates& coordinates_i = lattice[site_local.second]->coordinates[site_local.first];
00799
00800       if (coordinates_i[2] > polish_height)
00801         lattice[site_local.second]->iarray[0][site_local.first] = 0;
00802
00803       coord_update(site_local.first,site_local.second);
00804     }
00805     seed_n = occ_sites.size();
00806   }
00807 }
```

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

Calculates the local coordinates of a site

Parameters

coord	Coordinates of site
grain	Grain ID of site
direction	Direction (x, y, or z) of the local coordinates to return

Calculates the local 'direction'-coordinate of site.

```
01021 {
01022   matrix& inv_rot_mat = lattice[grain]->inv_rot_mat;
01023   array_coordinates& trans_mat = lattice[grain]->trans_mat;
01024   double local_coord;
01025
01026   if (direction == 0)
01027     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]);
01028   else if (direction == 1)
01029     local_coord = inv_rot_mat[1][0]*(coord[0] - trans_mat[0]) + inv_rot_mat[1][1]
    *(coord[1] - trans_mat[1]) + inv_rot_mat[1][2]*(coord[2] - trans_mat[2]);
01030   else if (direction == 2)
01031     local_coord = inv_rot_mat[2][0]*(coord[0] - trans_mat[0]) + inv_rot_mat[2][1]
    *(coord[1] - trans_mat[1]) + inv_rot_mat[2][2]*(coord[2] - trans_mat[2]);
01032   else
01033     error->all(FLERR, "Invalid direction");
01034
01035   return local_coord;
```

```
01036
01037 }
```

virtual void SPPARKS_NS::AppEAM::grow_app ( ) [inline], [virtual]

```
00048 {};
```

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
00252   // if potentiostatic, apply Butler-Volmer to get the current densities
00253   if (dep_mode == 1)
00254   {
00255     i_dep = i0*exp(-1. * alpha_c * eta * t_inverse);
00256     i_diss = i0*exp(alpha_a * eta * t_inverse);
00257
00258     if (pulse == 2)
00259     {
00260       i_a_dep = i0*exp(-1. * alpha_c * ox_rate * t_inverse);
00261       i_a_diss = i0*exp(alpha_a * ox_rate * t_inverse);
00262     }
00263   }
00264
00265   // vector of vector of doubles containing the propensity contribution of each
00266   // dep/diss event for each site
00267   dep_diss_prob.resize(N_total, vec_double(4, 0.));
00268
00269   if (pair == NULL)
00270     error->all(FLERR,"App EAM requires a pair potential");
00271
00272   if (temperature == 0.0)
00273     error->warning(FLERR, "Nothing will happen if temperature = 0.");
00274
00275   pair_int i_local;
00276
00277   // sum of dep/diss flags of each grain
00278   dep_sites_count_grain.resize(lattice_count, 0);
00279   diss_sites_count_grain.resize(lattice_count, 0);
```

```
00280
00281   vec_int seedlayer;
00282
00283   // find sites in seed layer
00284   for (int i = 0; i < lattice_count; i++)
00285   {
00286     int& N = lattice[i]->N;
00287
00288     vec_coordinates& coordinates = lattice[i]->coordinates;
00289
00290     for (int j = 0; j < N; j++)
00291     {
00292       if (coordinates[j][2] < seedlayer_size)
00293       {
00294         i_local = std::make_pair(j,i);
00295         seedlayer.push_back(grain_to_global(i_local));
00296       }
00297     }
00298
00299   }
00300
00301
00303
00304   seed_n = static_cast<int> (ceil(seed_frac*seedlayer.size())); // number of occupied
     sites in the seed layer
00305
00306   polish_height = 0.75*seedlayer_size;
00307
00308   if (seed_n > N_total)
00309     error->all(FLERR,"Specified density exceeds number of sites");
00310
00311   if (seed_n > 0)
00312     gen_seed_layer(ranapp, seedlayer);
00313
00314   std::cout << "Seeded " << seed_n << " sites." << std::endl;
00315
00316   // initializes coordination number, dep_flags, and active_sites
00317   for (int k = 0; k < lattice_count; k++)
00318   {
00319     int& N = lattice[k]->N;
00320
00321     active_sites[k].clear();  // clear out active_sites and start over to prevent
    double counting them
00322
00323     vec_int& num_neighbors = lattice[k]->num_neighbors;
00324     vec_vec_int& neighbors = lattice[k]->neighbors;
00325     vec_int& coordnum = lattice[k]->iarray[2];
00326     vec_int& dep_flag = lattice[k]->iarray[3];
00327     vec_coordinates& coordinates = lattice[k]->coordinates;
00328     vec_int& grainid = lattice[k]->iarray[1];
00329
00330     for (int i = 0; i < N; i++)
00331     {
00332       coord_update(i,k);
00333       // identify the edges of the active site
00334       if ((coordnum[i] != 12) && (coordnum[i] != 0))
00335         active_sites[k].push_back(i);
```

```
00336
00337       if ((seed_n == 0) && (coordinates[i][2] < (lattice[k]->next_site)*
    ((lattice[k]->latconst)/1.01)))
00338         active_sites[k].push_back(i);
00339
00340       grainid[i] = k;   // assign grain id
00341       dep_flag[i] = 0;  // clear out dep_flag
00342     }
00343
00344
00345   }
00346
00347   // check whether grain needs growing
00348   for (int k = 0; k < lattice_count; k++)
00349   {
00350     vec_int& type = lattice[k]->iarray[0];
00351     // check the grain (only check occupied active sites)
00352     for (int i = 0; i < active_sites[k].size(); i++)
00353     {
00354       if (type[active_sites[k][i]] == 1)
00355         update_status(i,k,0);
00356     }
00357   }
00358
00359   // check the distance between grains and flag any sites that are in close proximity
00360   for (int k = 0; k < lattice_count; k++)
00361     check_grain_distance(k);
00362
00363   dep_sites_count = 0;   // reset dep_sites_count - it might have been changed by
     gen_seed_layer
00364   diss_sites_count = 0;
00365
00366   // update dep_flag
00367   for (int k = 0; k < lattice_count; k++)
00368   {
00369     dep_sites_count_grain[k] = 0;  // clear out grain count
00370     diss_sites_count_grain[k] = 0;    // clear out grain count
00371     update_dep_flag(k);     // now update dep_flag
00372   }
00373 }
```

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

j	Local index of site j
grain	Grain ID of the two sites

Check if j is already part of neighbors2[i]

```
00813 {
00814   vec_int& num_neighbors2 = lattice[grain]->num_neighbors2;
00815   vec_vec_int& neighbors2 = lattice[grain]->neighbors2;
00816
00817   if (i == j)
00818     return 1;
00819
00820   if (j == -1)
00821     return 1;
00822
00823   if (num_neighbors2[i] == 0)
00824     return 0;
00825
00826   // using stl to do the search instead of for loop
00827   auto loc = std::find(neighbors2[i].begin(), neighbors2[i].end(), j);
00828
00829   if (loc != neighbors2[i].end())
00830     return 1;
00831
00832   // lattice site not in 2nd nearest neighbors list
00833   return 0;
00834 }
```

void AppEAM::scale_propensities ( ) [private]

Scales the deposition propensities such that the average = propensity with i_dep

Calculates the average propensity for each event based on the propensities stored in dep_diss_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.

```
00410 {
00411   diss_sites.clear(); // clear old data
00412   dep_sites.clear();
00413
00414   double sum = 0.0;
00415   double scale = 1.;
00416   double area = (domain->length(0))*(domain->length(1));
00417
00418   // find the deposition and dissolution sites whose propensity is > 1e-10
00419   // loop over active sites in each grain
00420   for (int j = 0; j < lattice_count; j++)
00421   {
00422     for (int ii = 0; ii < active_sites[j].size(); ii++)
```

```
00423     {
00424       int i = active_sites[j][ii];
00425       int k = grain_to_global(std::make_pair(i,j));
00426       // find all sites that can undergo deposition
00427       if ((lattice[j]->iarray[3][i] == 1) && (lattice[j]->iarray[0][i] == 0)
    && (propensity[k] > 1e-10))
00428         dep_sites.push_back(k);
00429       // now all sites that can undergo dissolution
00430       else if ((lattice[j]->iarray[3][i] == 2) && (lattice[j]->iarray[0][i] == 1)
    && (propensity[k] >1e-10))
00431         diss_sites.push_back(k);
00432
00433     }
00434   }
00435
00436   // start with deposition
00437   for (int i = 0; i < dep_sites.size(); i++)
00438   {
00439     // remove the contribution from total propensity for now, will add scaled value back to it
00440     propensity[dep_sites[i]] -= dep_diss_prob[dep_sites[i]][0];
00441     sum += dep_diss_prob[dep_sites[i]][0];  // add it to sum
00442   }
00443
00444   // find numx - area is in Angstroms^2, so we need to convert to nm^2
00445   double numx = dep_sites_count*100./area;
00446   // i_dep is in pA/nm^2, 1e-12 is to convert it to A/nm^2
00447   double avg = i_dep * 1.e-12/ (z_me*charge*numx);  // desired average propensity
00448
00449   sum = sum / dep_sites.size();  // divide by dep_sites.size() to get 'average' propensity
00450
00451   if (sum < 1e-15)  // to prevent divide by zero error
00452     scale = 1.;
00453   else
00454     scale = avg/sum; // get the new scaling parameter
00455
00456   scale_dep_on *= scale;  // apply that to existing scaling parameter
00457
00458   for (int i = 0; i < dep_sites.size(); i++)
00459   {
00460     dep_diss_prob[dep_sites[i]][0] *= scale; // apply the scale to old probs
00461     propensity[dep_sites[i]] += dep_diss_prob[dep_sites[i]][0];  // add back to propensity
00462   }
00463   // now dep rate when turned off - only in pulse-reverse
00464   if (pulse == 2)
00465   {
00466     sum = 0.;   // zero out sum
00467
00468     // again, remove contribution from propensity, add that to sum
00469     for (int i = 0; i < dep_sites.size(); i++)
00470     {
00471       propensity[dep_sites[i]] -= dep_diss_prob[dep_sites[i]][1];
00472       sum += dep_diss_prob[dep_sites[i]][1];
00473     }
00474
00475     avg = i_a_dep * 1.e-12/ (z_me*charge*numx);  // desired average propensity
00476
00477     sum = sum / dep_sites.size();  // divide by dep_sites.size() to get 'average' propensity
```

```
00478
00479     if (sum < 1e-15)  // to prevent divide by zero error
00480       scale = 1.;
00481     else
00482       scale = avg/sum; // get the new scaling parameter
00483
00484     scale_dep_off *= scale;
00485
00486     for (int i = 0; i < dep_sites.size(); i++)
00487     {
00488       dep_diss_prob[dep_sites[i]][1] *= scale; // apply the scaling factor
00489       propensity[dep_sites[i]] += dep_diss_prob[dep_sites[i]][1];  // add back to propensity
00490     }
00491   }
00492   solve->update(dep_sites,propensity);  // update the solver
00493
00494   // now for dissolution events - first check if there are any sites that fit the criteria
00495   if (diss_sites.size() != 0)
00496   {
00497     sum = 0.0;  // clear out sum
00498
00499     // repeat same procedure
00500     for (int i = 0; i < diss_sites.size(); i++)
00501     {
00502       propensity[diss_sites[i]] -= dep_diss_prob[diss_sites[i]][2];
00503       sum += dep_diss_prob[diss_sites[i]][2];
00504     }
00505
00506     numx = diss_sites_count*100./area;  // now with diss_sites_count for dissolution
00507     avg = i_diss * 1.e-12/ (z_me*charge*numx);  // desired average propensity
00508
00509     sum = sum / diss_sites.size();  // divide by dep_sites.size() to get 'average' propensity
00510     if (sum < 1e-15)
00511       scale = 1.;
00512     else
00513       scale = avg/sum; // get the new scaling parameter
00514
00515     scale_diss_on *= scale;
00516
00517     for (int i = 0; i < diss_sites.size(); i++)
00518     {
00519       dep_diss_prob[diss_sites[i]][2] *= scale; // apply the scaling factor
00520       propensity[diss_sites[i]] += dep_diss_prob[diss_sites[i]][2];  // add back to propensity
00521     }
00522
00523     // diss rate when off -only in PR
00524     if (pulse == 2)
00525     {
00526       sum = 0.;
00527       for (int i = 0; i < diss_sites.size(); i++)
00528       {
00529         propensity[diss_sites[i]] -= dep_diss_prob[diss_sites[i]][3];
00530         sum += dep_diss_prob[diss_sites[i]][3];
00531       }
00532
00533       avg = i_a_diss * 1.e-12/ (z_me*charge*numx);  // desired average propensity
00534       sum = sum / diss_sites.size();  // divide by dep_sites.size() to get 'average' propensity
```

```
00535
00536       if (sum < 1e-15)
00537         scale = 1.;
00538       else
00539         scale = avg/sum; // get the new scaling parameter
00540
00541       scale_diss_off *= scale;
00542
00543       for (int i = 0; i < diss_sites.size(); i++)
00544       {
00545         dep_diss_prob[diss_sites[i]][3] *= scale; // apply the scaling factor
00546         propensity[diss_sites[i]] += dep_diss_prob[diss_sites[i]][3];  // add back to propensity
00547       }
00548     }
00549     solve->update(diss_sites,propensity);   // update solver
00550   }
00551
00552 }
```

void AppEAM::setup_app ( ) [virtual]

Set up the application.

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

```
00380 {
00381   // compute the site energies for the first time
00382   for (int j = 0; j < lattice_count; j++)
00383   {
00384     for (int i = 0; i < lattice[j]->N; i++)
00385       pair->energy(i, j, lattice[j]->num_global_neighbors, lattice[j]->global_neighbors,
    lattice[j]->coordinates, lattice[j]->iarray[0], lattice[j]->darray[0], lattice[j]->darray[1]);
00386
00387   }
00388
00389   max_eng = -3.54;
00390 }
```

void AppEAM::setup_end_app ( ) [virtual]

Set up the application after propensity is computed.

Scale the propensities before application starts

```
00398 {
00399   scale_propensities();
00400 }
```

double AppEAM::site_energy (int *i*, int *grain*) [virtual]

Computes the energy of site i

```
01373 {
01374   vec_int& type = lattice[grain]->iarray[0];
01375
01376   vec_double& rho = lattice[grain]->darray[0];
01377   vec_double& phi = lattice[grain]->darray[1];
01378
01379   vec_coordinates& coordinates = lattice[grain]->coordinates;
01380   vec_int& num_global_neighbors = lattice[grain]->num_global_neighbors;
01381   vec_vec_int& global_neighbors = lattice[grain]->global_neighbors;
01382
01383   if (type[i] != 0)
01384     return pair->energy(i, grain, num_global_neighbors, global_neighbors, coordinates, type, rho,
    phi);
01385   else
01386     return 0.;
01387 }
```

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

```
02358 {
02359   if (propensity[i] == 0.)
02360   {
02361     naccept--;
02362     return;
02363   }
02364
02365   // if plating was originally on but now have exceeded on time, turn it off
02366   if ((pulse != 0) && (pulse_on == 1) && ((time - prev_time) > on_dt))
02367   {
02368     pulse_on = 0;
02369     prev_time = time;
02370   }
02371   // if it was off but have exceeded off time, turn it on
02372   else if ((pulse != 0) && (pulse_on == 0) && ((time - prev_time) > off_dt))
02373   {
02374     pulse_on = 1;
02375     prev_time = time;
02376   }
02377
02378   pair_int i_local = global_to_grain(i);
02379
02380   int jj;
02381
02382   vec_int& type = lattice[i_local.second]->iarray[0];
```

```
02383   vec_int& coordnum = lattice[i_local.second]->iarray[2];
02384   vec_int& dep_flag = lattice[i_local.second]->iarray[3];
02385
02386   vec_coordinates& coordinates = lattice[i_local.second]->coordinates;
02387   vec_int& num_neighbors = lattice[i_local.second]->num_neighbors;
02388   vec_vec_int& neighbors = lattice[i_local.second]->neighbors;
02389   vec_int& num_global_neighbors = lattice[i_local.second]->num_global_neighbors;
02390   vec_vec_int& global_neighbors = lattice[i_local.second]->global_neighbors;
02391   vec_int& num_neighbors2 = lattice[i_local.second]->num_neighbors2;
02392   vec_vec_int& neighbors2 = lattice[i_local.second]->neighbors2;
02393   double xmid = domain->midpoint(0);
02394   double ymid = domain->midpoint(1);
02395   double zmid = domain->midpoint(2);
02396
02397   // sites = array of site indices that have to be updated
02398   sites.clear();
02399
02400   // clear out diff_sites
02401   // first column of diff_sites contains the indices to sites that i can diffuse to
02402   // the second column is a flag indicating the diffusion mechanism
02403   // 0 = grain boundary diffusion; 1 = hopping; 2 = atom exchange on plateau;
02404   // 3 = atom exchange on edge
02405
02406   diff_sites.clear();
02407
02408   // zero out diff_count
02409   int diff_count = 0;
02410   int grain_j = i_local.second;
02411   int j = -1;
02412   int flag = 0;   // flag to indicate diffusion across a grain (=1)
02413   int step_flag = 0;  // flag to indicate whether step-edge atom exchange is possible or not
02414   double einitial;
02415   double prob = 0.0;
02416   double efinal;
02417
02418   // threshold = random number * total propensity of site i
02419   double threshold = random->uniform()*propensity[i];
02420
02421   i = i_local.first;
02422
02423   // we only need to update the status of the site and the system energy
02424   if (type[i] == 0)
02425   {
02426     if (count >= n_max)
02427     {
02428       naccept--;
02429       return;
02430     }
02431
02432     if ((dep_flag[i] == 1) && (pulse_on == 1))
02433     {
02434       einitial = system_energy(i,j,0,i_local.second,i_local.second);
02435
02436       type[i] = 1;
02437
02438       // find surface area in (Angstroms)^2
02439       double area = (domain->length(0))*(domain->length(1));
```

```
02440
02441       // possible dep sites per unit area (sites/nm^2)
02442       double numx = dep_sites_count*100./area;
02443
02444       efinal = system_energy(i,j,1,i_local.second,i_local.second);
02445
02446       // Budevski et al. p.28
02447       prob += i_dep * 1.e-12 *(efinal - einitial) * scale_dep_on/ (
    z_me*charge*numx*max_eng);
02448
02449       if (prob >= threshold)
02450       {
02451         // need to run system_energy to calculate the energy after the move
02452         // otherwise, the energy will not be updated
02453         efinal = system_energy(i,j,2,i_local.second,i_local.second);
02454         count++;
02455         grain_j = i_local.second;
02456         goto update;
02457       }
02458       else
02459       {
02460         type[i] = 0;
02461         naccept--;
02462         return;
02463       }
02464     }
02465     // deposition during anodic part of pulse-reverse
02466     else if ((dep_flag[i] == 1) && (pulse == 2) && (pulse_on == 0))
02467     {
02468       einitial = system_energy(i,j,0,i_local.second,i_local.second);
02469
02470       type[i] = 1;
02471
02472       // find surface area in (Angstroms)^2
02473       double area = (domain->length(0))*(domain->length(1));
02474
02475       // possible dep sites per unit area (sites/nm^2)
02476       double numx = dep_sites_count*100./area;
02477
02478       efinal = system_energy(i,j,1,i_local.second,i_local.second);
02479
02480       // Budevski et al. p.28
02481       prob += i_a_dep * 1.e-12 *(efinal - einitial)*scale_dep_off / (
    z_me*charge*numx*max_eng);
02482
02483       if (prob >= threshold)
02484       {
02485         // need to run system_energy to calculate the energy after the move
02486         // otherwise, the energy will not be updated
02487         efinal = system_energy(i,j,2,i_local.second,i_local.second);
02488         count++;
02489         grain_j = i_local.second;
02490         goto update;
02491       }
02492       else
02493       {
02494         type[i] = 0;
```

```
02495         naccept--;
02496         return;
02497       }
02498     }
02499
02500     // if neither are satisfied
02501     naccept--;
02502     return;
02503
02504   }
02505   // if the event is diffusion, have to loop over all possible diffusion sites
02506   else
02507   {
02508     // special case of diffusion - grain boundary migration
02509     // the event will affect sites in 2 different grains
02510 //     for (int kk = 0; kk < num_global_neighbors[i]; kk++)
02511 //     {
02512 //       int k = global_neighbors[i][kk];
02513 //       pair_int k_local = global_to_grain(k);
02514 //
02515 //       // if they are in the same grain, ignore it
02516 //       if (k_local.second == i_local.second)
02517 //         continue;
02518 //
02519 //       // we have already identified possible destination sites using dep_flag
02520 //       // when dep_flag[k] = -1, atom i can diffuse to site k (based on the
02521 //       // criteria of assigning -1 to dep_flag)
02522 //       vec_int& dep_flag_k = lattice[k_local.second]->iarray[3];
02523 //
02524 //       // if dep_flag[k] is not -1, the two sites are not close enough
02525 //       if (dep_flag_k[k_local.first] != -1)
02526 //         continue;
02527 //
02528 //       vec_int& type_k = lattice[k_local.second]->iarray[0];
02529 //
02530 //       einitial = system_energy(i,k_local.first,0,i_local.second, k_local.second);
02531 //
02532 //       type[i] = 0;
02533 //       type_k[k_local.first] = 1;
02534 //
02535 //       efinal = system_energy(i,k_local.first,1,i_local.second, k_local.second);
02536 //
02537 //       if (efinal <= einitial)
02538 //         prob += v_grain*exp(-Ed_grain*t_inverse);
02539 //       else  // if efinal > einitial, this event is impossible
02540 //         prob += v_grain*exp(-(Ed_grain + efinal - einitial)*t_inverse);
02541 //
02542 //       if (prob >= threshold)
02543 //       {
02544 //         efinal = system_energy(i,k_local.first,2,i_local.second, k_local.second);
02545 //
02546 //         j = k_local.first;
02547 //         grain_j = k_local.second;
02548 //         n_grain++;
02549 //
02550 //         flag = 1;
02551 //
```

```
02552 //         // flip the dep_flag of i and k as well since they swap states
02553 //         dep_flag[i] = -1;
02554 //         dep_flag_k[k_local.first] = -2;
02555 //         break;
02556 //       }
02557 //       else
02558 //       {
02559 //         type_k[k_local.first] = 0;
02560 //         type[i] = 1;
02561 //         j = -1;
02562 //       }
02563 //     }
02564 //
02565 //     if (j != -1)
02566 //       goto update;
02567 //
02568
02569     if (coordinates[i][2] < (lattice[i_local.second]->next_site)
    *((lattice[i_local.second]->latconst)/1.01))
02570     {
02571       naccept--;
02572       return;
02573     }
02574     // dissolution when not pulse-reverse
02575     if ((dep_flag[i] == 2) && (pulse_on == 1) && (count < n_max))
02576     {
02577
02578       einitial = system_energy(i,j,0,i_local.second,i_local.second);
02579
02580       type[i] = 0;
02581
02582       // get efinal without permanently changing the energy
02583       efinal = system_energy(i,j,1,i_local.second,i_local.second);
02584
02585       double area = (domain->length(0))*(domain->length(1)); // find surface area in (Angstroms)^2
02586
02587       double numx = diss_sites_count*100./area; // possible dep sites per unit area
     (sites/nm^2)
02588
02589       // only need einitial, efinal is zero
02590       // sites with higher energy is less negative, max_eng/einitial will be higher
02591       prob += i_diss * 1.e-12 * max_eng*scale_diss_on/ (
    z_me*charge*numx*(einitial-efinal));
02592 //       prob += i_diss * 1.e-12 / (z_me*charge*numx);
02593
02594       if (prob >= threshold)
02595       {
02596         // need to run system_energy to calculate the energy after the move
02597         // otherwise, the energy will not be updated
02598         efinal = system_energy(i,j,2,i_local.second,i_local.second);   // energy after
     proposed move
02599         n_diss++;
02600         grain_j = i_local.second;
02601         goto update;
02602       }
02603       else
02604         type[i] = 1;
```

```
02605
02606     }
02607     // anodic part of pulse-reverse -- remove sites at i_a_diss
02608     if ((dep_flag[i] == 2) && (pulse_on == 0) && (pulse == 2) && (
    count < n_max))
02609     {
02610       einitial = system_energy(i,j,0,i_local.second,i_local.second);
02611
02612       type[i] = 0;
02613
02614       // get efinal without permanently changing the energy
02615       efinal = system_energy(i,j,1,i_local.second,i_local.second);
02616
02617       double area = (domain->length(0))*(domain->length(1)); // find surface area in (Angstroms)^2
02618
02619       double numx = diss_sites_count*100./area; // possible dep sites per unit area
     (sites/nm^2)
02620
02621       // sites with higher energy is less negative, max_eng/(einitial-efinal) will be higher
02622       prob += i_a_diss * 1.e-12 * max_eng* scale_diss_off/
    (z_me*charge*numx*(einitial-efinal));  // convert i_dep to A/nm^2 and calculate propensity
02623
02624       if (prob >= threshold)
02625       {
02626         // need to run system_energy to calculate the energy after the move
02627         // otherwise, the energy will not be updated
02628         efinal = system_energy(i,j,2,i_local.second,i_local.second);
02629         n_diss++;
02630         grain_j = i_local.second;
02631         goto update;
02632       }
02633       else
02634         type[i] = 1;
02635
02636     }
02637
02638     if (coordnum[i] > 6)
02639     {
02640       naccept--;
02641       return;
02642     }
02643
02644     if (edge_check(i, i_local.second))
02645     {
02646       naccept--;
02647       return;
02648     }
02649
02650     // loop over first nearest neighbors to see if there's any neighbor that has
02651     // dep_flag = -1 (close to or overlapping with another grain)
02652     // if there is, change step_flag to -1 to prevent step-edge atom exch
02653     for (int kk = 0; kk < num_neighbors[i]; kk++)
02654     {
02655       int k = neighbors[i][kk];
02656
02657       if (dep_flag[k] == -1)
02658         step_flag = 1;
```

```
02659     }
02660
02661     // get the direction that the local coordinate of i and j will be compared
02662     int direction = determine_direction(i, i_local.second);
02663     int dir;
02664     double local_coord_i, local_coord_k, local_coord_n;
02665
02666     if (direction != -1)
02667     {
02668       dir = static_cast<int> (floor(direction*0.5));
02669       local_coord_i = get_local_coordinates(coordinates[i], i_local.second, dir)
    ;
02670
02671       // if we're dealing with lower limit, multiply the coordinate by -1 to get the
02672       // right comparison
02673       if (direction % 2 == 0)
02674         local_coord_i *= -1;
02675     }
02676
02677     double dx, dy, dz, rsq;
02678     double cutoff = (lattice[i_local.second]->latconst)*(lattice[i_local.second]->latconst);
02679     // find diff_sites and diff_count
02680     // diff_sites is an array containing the indices of sites that i can diffuse to
02681     for (int kk = 0; kk < num_global_neighbors[i]; kk++)
02682     {
02683       int k = global_neighbors[i][kk];
02684       pair_int k_local = global_to_grain(k);
02685
02686       int& type_k = (lattice[k_local.second]->iarray[0])[k_local.first];
02687       int& coordnum_k = (lattice[k_local.second]->iarray[2])[k_local.first];
02688       int& dep_flag_k = (lattice[k_local.second]->iarray[3])[k_local.first];
02689
02690       array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];
02691
02692       dx = coordinates[i][0] - coordinates_k[0];
02693       dy = coordinates[i][1] - coordinates_k[1];
02694       dz = coordinates[i][2] - coordinates_k[2];
02695
02696       if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
02697         dx = fabs(dx) - 2.*xmid;
02698
02699       if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
02700         dy = fabs(dy) - 2.*ymid;
02701
02702       if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
02703         dz = fabs(dz) - 2.*zmid;
02704
02705       rsq = dx*dx + dy*dy + dz*dz;
02706
02707       if (rsq > cutoff)
02708         continue;
02709
02710       // hopping mechanism across grains
02711       // coordnum should be > 1, or else we're diffusing into thin air
02712       if ((i_local.second != k_local.second) && (type_k == 0)
02713           && (coordnum_k > 1) && (dep_flag[i] == -2)
02714           && (exchange_destination(k_local.first, k_local.second, direction)))
```

```
02715       {
02716         // hopping to another grain on the surface
02717         diff_sites.push_back(int_array_2 ());
02718         diff_sites[diff_count][1] = 4;
02719         diff_sites[diff_count][0] = k;
02720         diff_count++;
02721 
02722       }
02723     }  // end of global neighbors loop
02724 
02725     for (int kk = 0; kk < num_neighbors[i]; kk++)
02726     {
02727       int k = neighbors[i][kk];
02728 
02729       if (direction != -1)
02730       {
02731         local_coord_k = get_local_coordinates(coordinates[k], i_local.second, dir);
02732 
02733         if (direction % 2 == 0)
02734           local_coord_k *= -1;
02735       }
02736       // regular hopping
02737       if ((type[k] == 0) && (coordnum[k] > 1) && (dep_flag[k] != -1))
02738       {
02739         // special case - grain boundary diffusion
02740         // assume that it happens only via hopping for now
02741         // Herzig2005 and Suzuki2005
02742         diff_sites.push_back(int_array_2 ());
02743 
02744         if (dep_flag[i] == -2)
02745           diff_sites[diff_count][1] = 0;
02746         else
02747           diff_sites[diff_count][1] = 1;
02748 
02749         diff_sites[diff_count][0] = k;
02750         diff_count++;  // # of possible diffusion sites
02751       }
02752 
02753       if ((type[k] == 1) && (local_coord_k < local_coord_i) && (coordnum[k] >= 6)
02754         && (direction != -1) && (dep_flag[i] != -2))
02755       // exchange
02756       {
02757         for (int nn = 0; nn < num_neighbors[k]; nn++)
02758         {
02759           int n = neighbors[k][nn]; // destination site
02760 
02761           if (dep_flag[n] == -1)
02762             continue;
02763 
02764           if (type[n] != 0)
02765             continue;
02766 
02767           if (exchange_destination(n, i_local.second, direction) == 0)
02768             continue;
02769 
02770           local_coord_n = get_local_coordinates(coordinates[n], i_local.second, dir);
02771 
```

```
02772           if (direction % 2 == 0)
02773             local_coord_n *= -1;
02774
02775           if ((fabs(local_coord_n - local_coord_i)  < 0.001)
02776             && (coordnum[k] >= 8))
02777           {
02778           // atom exchange
02779             diff_sites.push_back(int_array_2 ());
02780             diff_sites[diff_count][0] = n;
02781             diff_sites[diff_count][1] = 2;
02782             diff_count++;
02783           }
02784           else if ((fabs(local_coord_n - local_coord_k)  < 0.001)
02785               && (coordnum[k] < 8) && (step_flag == 0)
02786               && (((direction % 2 == 0)
02787                   && (coordinates[i][dir] < coordinates[n][dir])
02788                   && (fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001))
02789                 || ((direction % 2 == 1)
02790                   && (coordinates[i][dir] > coordinates[n][dir])
02791                   && (fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001))))
02792           {
02793           // step edge atom exchange
02794             diff_sites.push_back(int_array_2 ());
02795             diff_sites[diff_count][0] = n;
02796             diff_sites[diff_count][1] = 3;
02797             diff_count++;
02798           }
02799         }
02800       }   // end of exchange loop
02801       }   // end of nearest neighbors loop
02802     // if we are assuming that the diffusion rate is the same for all configurations
02803     // and sites
02804     // if diff is independent of the energy of the system
02805     for (int k = 0; k < diff_count; k++)
02806     {
02807       j = diff_sites[k][0];  // find site j from diff_sites
02808       grain_j = i_local.second;
02809       pair_int j_local;
02810
02811       // special case for hopping - hop across grains
02812       if (diff_sites[k][1] == 4)
02813       {
02814         j_local = global_to_grain(j);
02815         j = j_local.first;
02816         grain_j = j_local.second;
02817
02818         einitial = system_energy(i,j_local.first,0,i_local.second,j_local.second);
02819
02820         // type[j] = 1
02821         lattice[j_local.second]->iarray[0][j_local.first] = 1;
02822         type[i] = 0;
02823
02824         efinal = system_energy(i,j_local.first,1,i_local.second,j_local.second);
02825       }
02826       else
02827       {
02828         einitial = system_energy(i,j,0,i_local.second,i_local.second);
```

```
02829
02830         type[j] = 1;
02831         type[i] = 0;  // in surface diffusion, i becomes vacant
02832         efinal = system_energy(i,j,1,i_local.second,i_local.second);
02833       }
02834
02835       if (efinal <= einitial)
02836       {
02837         if (diff_sites[k][1] == 0)
02838           prob += v_boundary*exp(-1.0*Ed_boundary*t_inverse);
02839         else if ((diff_sites[k][1] == 1) || (diff_sites[k][1] == 4))
02840           prob += v_d*exp(-1.0*Ed*t_inverse);
02841         else if (diff_sites[k][1] == 2)
02842           prob += v_exch*exp(-1.0*Ed_exch*t_inverse);
02843         else if (diff_sites[k][1] == 3)
02844           prob += v_step*exp(-1.0*Ed_step*t_inverse);
02845
02846         if (prob >= threshold)
02847         {
02848           efinal = system_energy(i,j,2,i_local.second,grain_j);   // do the energy update
02849           if (diff_sites[k][1] == 0)
02850             n_boundary++;
02851           else if (diff_sites[k][1] == 1)
02852             n_hop++;
02853           else if (diff_sites[k][1] == 2)
02854             n_atomexch++;
02855           else if (diff_sites[k][1] == 3)
02856             n_step++;
02857           else if (diff_sites[k][1] == 4)
02858             n_hop++;
02859
02860           break;  // move accepted, break the for loop
02861         }
02862         else
02863         // if prob < threshold, restore the types if i and j
02864         {
02865           if (diff_sites[k][1] != 4)
02866             type[j] = 0;
02867           else
02868             lattice[j_local.second]->iarray[0][j_local.first] = 0;
02869
02870           type[i] = 1;
02871           j = -1;   // set j = -1 since nothing happened
02872         }
02873       }
02874       else
02875       {
02876         if (diff_sites[k][1] == 0)
02877           prob += v_boundary*exp(-1.0*(Ed_boundary + efinal - einitial)*t_inverse);
02878         else if ((diff_sites[k][1] == 1) || (diff_sites[k][1] == 4))
02879           prob += v_d*exp(-1.0*(Ed + efinal - einitial)*t_inverse);
02880         else if (diff_sites[k][1] == 2)
02881           prob += v_exch*exp(-1.0*(Ed_exch + efinal - einitial)*t_inverse);
02882         else if (diff_sites[k][1] == 3)
02883           prob += v_step*exp(-1.0*(Ed_step + efinal - einitial)*t_inverse);
02884
02885         if (prob >= threshold)
```

```
02886         {
02887           efinal = system_energy(i,j,2,i_local.second,grain_j);   // do the energy update
02888           if (diff_sites[k][1] == 0)
02889             n_boundary++;
02890           else if (diff_sites[k][1] == 1)
02891             n_hop++;
02892           else if (diff_sites[k][1] == 2)
02893             n_atomexch++;
02894           else if (diff_sites[k][1] == 3)
02895             n_step++;
02896           else if (diff_sites[k][1] == 4)
02897             n_hop++;
02898
02899           break;
02900         }
02901         else  // restore the types if i and j
02902         {
02903           if (diff_sites[k][1] != 4)
02904             type[j] = 0;
02905           else
02906             lattice[j_local.second]->iarray[0][j_local.first] = 0;
02907
02908           type[i] = 1;
02909           j = -1;
02910         }
02911       }
02912     }   // end of diff_count loop
02913
02914     if (j == -1)  // if nothing happened, skip the rest and return
02915     {
02916       naccept--;
02917       return;
02918     }
02919     goto update;
02920
02921   }
02922   update:
02923   coord_update(i,i_local.second);  // update the coordination numbers
02924
02925   if (j != -1)  // if there's a site j, update that too
02926     coord_update(j,grain_j);
02927
02928   // check the status of i as an active site and check if grain needs growing
02929   update_status(i,i_local.second,1);
02930
02931   // same for j if there is a site j
02932   if (j != -1)
02933     update_status(j,grain_j,1);
02934
02935   // check distance between grains
02936   check_grain_distance(i,i_local.second);
02937
02938   for (int mm = 0; mm < num_neighbors[i]; mm++)
02939     check_grain_distance(neighbors[i][mm], i_local.second);
02940
02941   if (j != -1)
02942   {
```

```
02943     check_grain_distance(j,grain_j);
02944
02945     for (int mm = 0; mm < lattice[grain_j]->num_neighbors[j]; mm++)
02946       check_grain_distance(lattice[grain_j]->neighbors[j][mm], grain_j);
02947   }
02948
02949   if (grain_j != i_local.second)
02950     flag = 1;   // flip the flag so that the update is done on the right grain
02951
02952   for (int m = 0; m < lattice_count; m++)
02953     update_dep_flag(m);
02954
02955   scale_propensities();
02956
02957   // update the propensities for site i and its 2nd nearest neighbors - use global indices
02958   int ii = grain_to_global(i_local);
02959   sites.push_back(ii);
02960   propensity[ii] = site_propensity(ii);
02961
02962   for (int mm = 0; mm < num_neighbors2[i]; mm++)
02963   {
02964     int m = neighbors2[i][mm];
02965     pair_int grain_m = std::make_pair(m, i_local.second);
02966     m = grain_to_global(grain_m);
02967     sites.push_back(m);
02968     propensity[m] = site_propensity(m);
02969   }
02970
02971   if (j != -1)  // if diffusion, need to update j and its 2nd nn as well
02972   {
02973     if (flag == 0)  // if same grain
02974     {
02975       if (!neighbor_check(i,j, i_local.second))
02976       {
02977         jj = grain_to_global(std::make_pair(j,grain_j));
02978         sites.push_back(jj);
02979         propensity[jj] = site_propensity(jj);
02980       }
02981       for (int mm = 0; mm < num_neighbors2[j]; mm++)
02982       {
02983         int m = neighbors2[j][mm];
02984         if (!neighbor_check(i,m, i_local.second))
02985         // if it's a '2nd nn' of i, it would have already been taken care of in
02986         // the first for loop; this check is there to optimize the update
02987         {
02988           pair_int grain_m = std::make_pair(m, i_local.second);
02989           m = grain_to_global(grain_m);
02990           sites.push_back(m);
02991           propensity[m] = site_propensity(m);
02992         }
02993       }
02994     }
02995     else  // if another grain is affected, flag = 1 and the update is done differently
02996     {
02997       jj = grain_to_global(std::make_pair(j,grain_j));
02998       sites.push_back(jj);
02999       propensity[jj] = site_propensity(jj);
```

```
03000
03001       vec_int& num_neighbors2_j = lattice[grain_j]->num_neighbors2;
03002       vec_vec_int& neighbors2_j = lattice[grain_j]->neighbors2;
03003
03004       for (int mm = 0; mm < num_neighbors2_j[j]; mm++)
03005       {
03006         int m = neighbors2_j[j][mm];
03007         pair_int grain_m = std::make_pair(m, grain_j);
03008         m = grain_to_global(grain_m);
03009         sites.push_back(m);
03010         propensity[m] = site_propensity(m);
03011       }
03012
03013     }
03014   }
03015
03016   // update
03017   solve->update(sites,propensity);
03018 }
```

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 fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001 is to ensure that the two sites are in different layers.

```
01850 {
01851   int i_global = i;
01852
01853   pair_int i_local = global_to_grain(i);
01854
01855   double prob = 0.0;
01856
01857   int flag = 0;   // flag to indicate whether step-edge atom exchange is possible or not
01858
01859   i = i_local.first;   // get the local i and overwrite i
01860
01861   vec_int& type = lattice[i_local.second]->iarray[0];
01862   vec_int& coordnum = lattice[i_local.second]->iarray[2];
01863   vec_int& dep_flag = lattice[i_local.second]->iarray[3];
01864
01865   vec_coordinates& coordinates = lattice[i_local.second]->coordinates;
01866   vec_int& num_neighbors = lattice[i_local.second]->num_neighbors;
01867   vec_vec_int& neighbors = lattice[i_local.second]->neighbors;
01868   vec_int& num_global_neighbors = lattice[i_local.second]->num_global_neighbors;
01869   vec_vec_int& global_neighbors = lattice[i_local.second]->global_neighbors;
01870   vec_int& num_neighbors2 = lattice[i_local.second]->num_neighbors2;
01871   vec_vec_int& neighbors2 = lattice[i_local.second]->neighbors2;
01872
01873   double xmid = domain->midpoint(0);
01874   double ymid = domain->midpoint(1);
01875   double zmid = domain->midpoint(2);
01876
01877   // SPPARKS will calculate the propensities of all sites before performing site_event
01878   // site_event will only update the propensities of affected sites
01879
01880   // propensity of all events at site i
01881   // possible events: adsorption and diffusion, depending on the type of site i
01882   // if the site is part of the bulk or in the vacuum, return prob = 0
01883   if ((coordnum[i] == 12) || (coordnum[i] == 0))
01884     return prob;
01885   // if the temperature is 0 K, the rate = 0
01886   if (temperature == 0.0)
01887     return prob;
01888
01889   int j = -1;  // diffusion destination site index, -1 for adsorption
01890
01891   double einitial;  // energy of initial configuration
01892   double efinal;
01893
01894   if (type[i] == 0)
```

```
01895   {
01896     // if dep_flag[i] = -1 or 0, return prob = 0
01897     if (dep_flag[i] != 1)
01898       return prob;
01899
01900     if (count >= n_max)
01901       return prob;
01902
01903     einitial = system_energy(i,j,0,i_local.second,i_local.second);
01904
01905     type[i] = 1;
01906
01907     // get efinal without permanently changing the energy
01908     efinal = system_energy(i,j,1,i_local.second,i_local.second);
01909
01910     type[i] = 0;
01911
01912     double area = (domain->length(0))*(domain->length(1)); // find surface area in (Angstroms)^2
01913
01914     double numx = dep_sites_count*100./area; // possible dep sites per unit area
     (sites/nm^2)
01915     // convert i_dep to A/nm^2 and calculate propensity
01916     double prob_tmp = i_dep * 1.e-12 * (efinal - einitial)*scale_dep_on/ (
    z_me*charge*numx*max_eng);
01917
01918     if (prob_tmp < 0.)  // if calculated prob is < 0, let it = 0
01919     // this can occasionally happen in the seeding, but should not be frequent
01920     {
01921       dep_diss_prob[i_global][0] = 0.;
01922     }
01923     else
01924     {
01925       prob += prob_tmp;
01926       dep_diss_prob[i_global][0] = prob_tmp;
01927     }
01928
01929     if (pulse == 2)
01930     {
01931       einitial = system_energy(i,j,0,i_local.second,i_local.second);
01932
01933       type[i] = 1;
01934
01935       // find surface area in (Angstroms)^2
01936       double area = (domain->length(0))*(domain->length(1));
01937
01938       // possible dep sites per unit area (sites/nm^2)
01939       double numx = dep_sites_count*100./area;
01940
01941       efinal = system_energy(i,j,1,i_local.second,i_local.second);
01942
01943       type[i] = 0;
01944       // convert i_dep to A/nm^2 and calculate propensity
01945       double prob_tmp = i_a_dep * 1.e-12 *(efinal - einitial)*
    scale_dep_off/ (z_me*charge*numx*max_eng);
01946
01947       if (prob_tmp < 0.)
01948       {
```

```
01949         dep_diss_prob[i_global][1] = 0.;
01950       }
01951       else
01952       {
01953       // Budevski et al. p.28
01954         prob += prob_tmp;
01955         dep_diss_prob[i_global][1] = prob_tmp;
01956       }
01957     }
01958
01959   }
01960
01961   if (type[i] != 0)
01962   {
01963     for (int kk = 0; kk < num_neighbors[i]; kk++)
01964     {
01965       int k = neighbors[i][kk];
01966
01967       if ((coordnum[k] == 1) && (type[k] == 1))
01968         // if coordnum of one of i's neighbors is 1, that site will end up
01969         // floating in space when i diffuses away
01970         // this cannot happen, so if this is the case, i cannot diffuse
01971         return prob = 0.0;
01972     }
01973
01974     if (coordinates[i][2] < (lattice[i_local.second]->next_site)
    *((lattice[i_local.second]->latconst)/1.01))
01975       return prob;
01976
01977     // dissolution
01978     if ((dep_flag[i] == 2) && (count < n_max))
01979     {
01980       einitial = system_energy(i,j,0,i_local.second,i_local.second);
01981
01982       type[i] = 0;
01983
01984       // get efinal without permanently changing the energy
01985       efinal = system_energy(i,j,1,i_local.second,i_local.second);
01986
01987       type[i] = 1;
01988
01989       double area = (domain->length(0))*(domain->length(1)); // find surface area in (Angstroms)^2
01990
01991       double numx = diss_sites_count*100./area; // possible dep sites per unit area
     (sites/nm^2)
01992       // sites with higher energy is less negative, max_eng/(einitial-efinal) will be higher
01993       // if no i_diss is specified, the default is zero
01994       double prob_tmp = i_diss * 1.e-12 * (max_eng) *
    scale_diss_on/ (z_me*charge*numx*(einitial-efinal)); // convert i_dep to A/nm^2 and
     calculate propensity
01995
01996       if (prob_tmp < 0.)
01997       {
01998         dep_diss_prob[i_global][2] = 0.;
01999       }
02000       else
02001       {
```

```
02002         prob +=  prob_tmp;
02003         dep_diss_prob[i_global][2] = prob_tmp;
02004       }
02005
02006     }
02007     // anodic part of pulse-reverse -- remove sites at i_a_diss
02008     if ((dep_flag[i] == 2) && (pulse == 2) && (count < n_max))
02009     {
02010       einitial = system_energy(i,j,0,i_local.second,i_local.second);
02011
02012       type[i] = 0;
02013
02014       // get efinal without permanently changing the energy
02015       efinal = system_energy(i,j,1,i_local.second,i_local.second);
02016
02017       type[i] = 1;
02018
02019       double area = (domain->length(0))*(domain->length(1)); // find surface area in (Angstroms)^2
02020
02021       double numx = diss_sites_count*100./area; // possible dep sites per unit area
     (sites/nm^2)
02022
02023       // sites with higher energy is less negative, max_eng/(einitial-efinal) will be higher
02024       double prob_tmp = i_a_diss * 1.e-12 * (max_eng)*
    scale_diss_off / (z_me*charge*numx*(einitial-efinal));   // Budevski et al. p.28

02025
02026       if (prob_tmp < 0.)
02027       {
02028         dep_diss_prob[i_global][3] = 0.;
02029       }
02030       else
02031       {
02032         prob += prob_tmp;
02033         dep_diss_prob[i_global][3] = prob_tmp;
02034       }
02035
02036     }
02037
02038 //     // If site is occupied it can participate in diffusion across grains.
02039 //     for (int kk = 0; kk < num_global_neighbors[i]; kk++)
02040 //     {
02041 //       int k = global_neighbors[i][kk];
02042 //       pair_int k_local = global_to_grain(k);
02043 //
02044 //       // if they are in the same grain, ignore it
02045 //       if (k_local.second == i_local.second)
02046 //         continue;
02047 //
02048 //       // we have already identified possible destination sites using dep_flag
02049 //       // when dep_flag[k] = -1, atom i can diffuse to site k (based on the
02050 //       // criteria of assigning -1 to dep_flag)
02051 //       vec_int& dep_flag_k = lattice[k_local.second]->iarray[3];
02052 //
02053 //       // if dep_flag[k] is not -1, the two sites are not close enough
02054 //       if (dep_flag_k[k_local.first] != -1)
02055 //         continue;
```

```
02056 //
02057 //       vec_int& type_k = lattice[k_local.second]->iarray[0];
02058 //
02059 //       einitial = system_energy(i,k_local.first,0,i_local.second, k_local.second);
02060 //
02061 //       type[i] = 0;
02062 //       type_k[k_local.first] = 1;
02063 //
02064 //       efinal = system_energy(i,k_local.first,1,i_local.second, k_local.second);
02065 //
02066 //       type_k[k_local.first] = 0;
02067 //       type[i] = 1;
02068 //
02069 //       if (efinal <= einitial)
02070 //         prob += v_grain*exp(-Ed_grain*t_inverse);
02071 //       else
02072 //       // if we're moving to a higher energy state, need to account for the
02073 //       // configuration-specific increase in activation energy of the move
02074 //         prob += v_grain*exp(-(Ed_grain + efinal - einitial)*t_inverse);
02075 //
02076 //     }
02077
02078     if (edge_check(i,i_local.second))
02079     // if it is the edge and is part of the crystal, no diffusion (for non PBC)
02080       return prob;
02081
02082     // if site is part of the bulk, it cannot undergo diffusion
02083     if (coordnum[i] > 6)
02084       return prob;
02085
02086     // loop over first nearest neighbors to see if there's any neighbor that has
02087     // dep_flag = -1 (close to or overlapping with another grain)
02088     // if there is, change flag to -1 to prevent step-edge atom exch
02089     for (int kk = 0; kk < num_neighbors[i]; kk++)
02090     {
02091       int k = neighbors[i][kk];
02092
02093       if (dep_flag[k] == -1)
02094         flag = 1;
02095     }
02096
02097     // surface diffusion, site i becomes vacant while destination j becomes occupied
02098
02099     // determine the direction that is normal to site i; function returns int [0,5]
02100     int direction = determine_direction(i, i_local.second);
02101     double local_coord_i, local_coord_k, local_coord_n;
02102     int dir;
02103
02104     // function returns -1 when site is not exposed to the solution
02105     if (direction != -1)
02106     {
02108       dir = static_cast<int> (floor(direction*0.5));
02109       local_coord_i = get_local_coordinates(coordinates[i], i_local.second, dir);
02110
02111       // if we're dealing with lower limit, multiply the coordinate by -1 to get the
02112       // right comparison
02113       if (direction % 2 == 0)
```

```
02114         local_coord_i *= -1;
02115     }
02116
02117     double dx, dy, dz, rsq;
02118     // define cutoff for diffusion involving sites from different grains - use latconst
02119     double cutoff = (lattice[i_local.second]->latconst)*(lattice[i_local.second]->latconst);
02120
02121     // hopping to another grain on the surface - loop over global_neighbors
02122     for (int kk = 0; kk < num_global_neighbors[i]; kk++)
02123     {
02124       int k = global_neighbors[i][kk];
02125       pair_int k_local = global_to_grain(k);
02126
02127       int& type_k = (lattice[k_local.second]->iarray[0])[k_local.first];
02128       int& coordnum_k = (lattice[k_local.second]->iarray[2])[k_local.first];
02129       int& dep_flag_k = (lattice[k_local.second]->iarray[3])[k_local.first];
02130
02131       array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];
02132
02133       dx = coordinates[i][0] - coordinates_k[0];
02134       dy = coordinates[i][1] - coordinates_k[1];
02135       dz = coordinates[i][2] - coordinates_k[2];
02136
02137       if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
02138         dx = fabs(dx) - 2.*xmid;
02139
02140       if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
02141         dy = fabs(dy) - 2.*ymid;
02142
02143       if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
02144         dz = fabs(dz) - 2.*zmid;
02145
02146       rsq = dx*dx + dy*dy + dz*dz;
02147
02148       // if outside of cutoff, ignore that site
02149       if (rsq > cutoff)
02150         continue;
02151
02159       if ((i_local.second != k_local.second) && (type_k == 0)
02160         && (coordnum_k > 1) && (dep_flag_k == -1) && (dep_flag[i] == -2)
02161         && (exchange_destination(k_local.first, k_local.second, direction)))
02162       {
02163         // hopping to another grain on the surface
02164
02170         einitial = system_energy(i,k_local.first,0,i_local.second,k_local.second);
02171
02172         type_k = 1;  // update the type of site j
02173         type[i] = 0;  // in surface diffusion, i becomes vacant
02174
02175         efinal = system_energy(i,k_local.first,1,i_local.second,k_local.second);
02176
02177         // restore the types
02178         type_k = 0;
02179         type[i] = 1;
02180
02181         if (efinal <= einitial)
02182           prob += v_d*exp(-1.*Ed*t_inverse);
```

```
02183
02184         else
02185           prob += v_d*exp(-1.*(Ed + efinal - einitial)*t_inverse);
02186
02187       }
02188     }
02189     for (int kk = 0; kk < num_neighbors[i]; kk++)
02190     {
02191       int k = neighbors[i][kk];
02192       if (direction != -1)
02193       {
02194         local_coord_k = get_local_coordinates(coordinates[k], i_local.second, dir);
02195
02196         if (direction % 2 == 0)
02197           local_coord_k *= -1;
02198       }
02199       if ((type[k] == 0) && (coordnum[k] > 1) && (dep_flag[k] != -1))
02200       {
02201         // hopping
02202         // cannot diffuse to a site that has coordnum <= 1, it will be floating
02203         // if type[nearest neighbors] = 0, find the propensity of the move
02204         j = k;
02205
02206         einitial = system_energy(i,j,0,i_local.second,i_local.second);
02207
02208         type[j] = 1;  // update the type of site j
02209         type[i] = 0;  // in surface diffusion, i becomes vacant
02210
02211         efinal = system_energy(i,j,1,i_local.second,i_local.second);
02212
02213         // restore the types
02214         type[j] = 0;
02215         type[i] = 1;
02216
02217         if (efinal <= einitial)
02218         {
02219           if (dep_flag[i] == -2)  // special case, grain boundary diffusion
02220             prob += v_boundary*exp(-1.*Ed_boundary*t_inverse);
02221           else  // otherwise, regular hopping
02222             prob += v_d*exp(-1.*Ed*t_inverse);
02223         }
02224         else
02225         {
02226         // if we're moving to a higher energy state, need to account for the
02227         // configuration-specific increase in activation energy of the move
02228           if (dep_flag[i] == -2)
02229             prob += v_boundary*exp(-1.*(Ed_boundary + efinal - einitial)*t_inverse);
02230           else
02231             prob += v_d*exp(-1.*(Ed + efinal - einitial)*t_inverse);
02232         }
02233       }
02234       else if ((type[k] == 1) && (local_coord_k < local_coord_i)
02235          && (coordnum[k] >= 6) && (direction != -1) && (dep_flag[i] != -2))
02245       // exchange mechanism
02246       // coordnum = 6 is a kink site, can still participate in exchange
02247       // find intermediate sites, k, whose z[k] < z[i] (lower plane)
02248       // Antczak and Ehrlich, p. 88-104
```

```
02249       {
02250         for (int nn = 0; nn < num_neighbors[k]; nn++)
02251         {
02252         // loop over 1st nearest neighbors of k to find destination site
02253           int n = neighbors[k][nn];  // destination site
02254
02255           // if n is near an active site from another grain, move on
02256           if (dep_flag[n] == -1)
02257             continue;
02258
02259           if (type[n] != 0)
02260             continue;
02261
02262           // if n is not a possible exchange destination, move on
02263           if (exchange_destination(n, i_local.second,direction) == 0)
02264             continue;
02265
02266           local_coord_n = get_local_coordinates(coordinates[n], i_local.second, dir);
02267
02268           if (direction % 2 == 0)
02269             local_coord_n *= -1;
02270
02271           if ((fabs(local_coord_n - local_coord_i) < 0.001)
02272              && (coordnum[k] >= 8))
02273           {
02281            // atom exchange on a plateau
02282            // atom i will displace atom k, pushing atom m up to the same plane
02283            // as where i was initially
02284
02285             j = n;
02286
02287             einitial = system_energy(i,j,0,i_local.second,i_local.second);
02288
02289             type[j] = 1;
02290             type[i] = 0;
02291
02292             efinal = system_energy(i,j,1,i_local.second,i_local.second);
02293
02294             type[j] = 0;
02295             type[i] = 1;
02296
02297             if (efinal <= einitial)
02298               prob += v_exch*exp(-1.0*Ed_exch*t_inverse);
02299             else
02300               prob += v_exch*exp(-1.0*(Ed_exch+efinal-einitial)*t_inverse);
02301           }
02302           else if ((fabs(local_coord_n - local_coord_k) < 0.001)
02303               && (coordnum[k] < 8) && (flag == 0)
02304               && (((direction % 2 == 0)
02305                   && (coordinates[i][dir] < coordinates[n][dir])
02306                   && (fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001))
02307                 || ((direction % 2 == 1)
02308                   && (coordinates[i][dir] > coordinates[n][dir])
02309                   && (fabs(coordinates[i][dir]-coordinates[n][dir]) > 0.001))))
02310           {
02325           // step edge atom exchange at edges
02326           // atom i will displace atom k, pushing atom k outwards
```

```
02327           // coordnum[k] < 8 to ensure that it is at the edge
02328             j = n;
02329
02330             einitial = system_energy(i,j,0,i_local.second,i_local.second);
02331
02332             type[j] = 1;
02333             type[i] = 0;
02334
02335             efinal = system_energy(i,j,1,i_local.second,i_local.second);
02336
02337             type[j] = 0;
02338             type[i] = 1;
02339
02340             if (efinal <= einitial)
02341               prob += v_step*exp(-1.0*Ed_step*t_inverse);
02342             else
02343               prob += v_step*exp(-1.0*(Ed_step+efinal-einitial)*t_inverse);
02344           }
02345         }
02346       }
02347     }
02348   }
02349
02350   return prob;
02351 }
```

void AppEAM::stats (char * *strtmp*)

```
03024 {
03025   char big[8],format[256];
03026   strcpy(big,BIGINT_FORMAT);
03027
03028   if (solve)
03029   {
03030     sprintf(format,"%%10g %%10%s %%10d %%10d %%10d %%10d %%10d %%10d %%10d
    %%10d %%10d %%10g %%10g",&big[1]);
03031     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);
03032   }
03033   else
03034   {
03035     sprintf(format,"%%10g %%10%s %%10%s %%10d %%10d %%10d %%10d %%10d %%10d
    %%10d %%10d %%10g %%10g",&big[1],&big[1]);
03036     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);
03037   }
03038 }
```

void AppEAM::stats_header (char * *strtmp*)

```
03045 {
03046   sprintf(strtmp,"%10s %10s %10s %10s %10s %10s %10s %10s %10s %10s %10s %10s %10s", "Time",
    "Naccept","N_hop", "N_atomexch", "N_stepedge", "N_boundary", "N_grain", "N_dep", "N_diss",
    "sum_dep", "sum_diss",  "scale_dep", "scale_diss");
03047 }
```

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

i	Local index of site in question
j	If the event is a diffusion event, local index of destination site
flag	Flag to indicate whether a permanent update of the energies is required or not
grain_i	Grain ID of the site i
grain_j	Grain ID of the site j

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;
```

```
01418
01419   if (flag == 0)
01420   {
01421     double xmid = domain->midpoint(0);
01422     double ymid = domain->midpoint(1);
01423     double zmid = domain->midpoint(2);
01424     double dx, dy, dz, rsq;
01425     // use latconst as the cutoff
01426     double cutoff = (lattice[0]->latconst)*(lattice[0]->latconst);
01427
01428     // initialize the references to the grain data
01429     double& phi_i = (lattice[grain_i]->darray[1])[i];
01430     // create a copy of i's global_neighbors list
01431     vec_int global_neighbors_i(lattice[grain_i]->global_neighbors[i]);
01432     int num_global_neighbors_i = lattice[grain_i]->num_global_neighbors[i];
01433     array_coordinates& coordinates_i = lattice[grain_i]->coordinates[i];
01434
01435     old_eng = 0.0;  // zero out old_eng
01436
01437     for (int kk = 0; kk < lattice_count; kk++)
01438     {
01439       for (int k = 0; k < lattice[kk]->N; k++)
01440         old_eng += lattice[kk]->darray[1][k];   // add up energy before change
01441     }
01442
01443     // clear out phi_old
01444     phi_old.clear();
01445
01446     phi_old.push_back(array_coordinates());
01447     phi_old[0][0] = grain_to_global(std::make_pair(i, grain_i));
01448     phi_old[0][1] = phi_i;   // start adding sites that will see their energy changed
01449
01450     phi_old_count = 1;  // number of sites whose phi are stored in phi_old
01451     // loop over global neighbors (within EAM potential cutoff)
01452     for (int kk = num_global_neighbors_i-1; kk >= 0; kk--)
01453     {
01454       int k = global_neighbors_i[kk];
01455       pair_int k_local = global_to_grain(k);
01456
01457       array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];
01458
01459       dx = coordinates_i[0] - coordinates_k[0];
01460       dy = coordinates_i[1] - coordinates_k[1];
01461       dz = coordinates_i[2] - coordinates_k[2];
01462
01463       if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01464         dx = fabs(dx) - 2.*xmid;
01465
01466       if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01467         dy = fabs(dy) - 2.*ymid;
01468
01469       if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01470         dz = fabs(dz) - 2.*zmid;
01471
01472       rsq = dx*dx + dy*dy + dz*dz;
01473
01474       // ignore sites outside the cutoff
```

```
01475       if (rsq > cutoff)
01476       {
01477         global_neighbors_i.erase(global_neighbors_i.begin() + kk);
01478         continue;
01479       }
01480       phi_old.push_back(array_coordinates());
01481       phi_old[phi_old_count][0] = k;
01482       phi_old[phi_old_count][1] = lattice[k_local.second]->darray[1][k_local.first];
01483       phi_old_count++;
01484     }
01485
01486     // if the event is surface diffusion, the energy of site j and its 2nd
01487     // nearest neighbor are also stored in phi_old
01488     if (j != -1)
01489     {
01490       double& phi_j = lattice[grain_j]->darray[1][j];
01491
01492       vec_int& global_neighbors_j = lattice[grain_j]->global_neighbors[j];
01493       array_coordinates& coordinates_j = lattice[grain_j]->coordinates[j];
01494
01495       phi_old.push_back(array_coordinates());
01496       phi_old[phi_old_count][0] = grain_to_global(std::make_pair(j, grain_j));
01497       phi_old[phi_old_count][1] = phi_j;
01498       phi_old_count++;
01499
01500       // no need to create a copy for j's global_neighbor list
01501       for (int kk = 0; kk < global_neighbors_j.size(); kk++)
01502       {
01503         int k = global_neighbors_j[kk];
01504         pair_int k_local = global_to_grain(k);
01505
01506         array_coordinates& coordinates_k = lattice[k_local.second]->coordinates[k_local.first];
01507
01508         dx = coordinates_j[0] - coordinates_k[0];
01509         dy = coordinates_j[1] - coordinates_k[1];
01510         dz = coordinates_j[2] - coordinates_k[2];
01511
01512         if ((fabs(dx) > xmid) && (domain->periodicity[0] == 1))
01513           dx = fabs(dx) - 2.*xmid;
01514
01515         if ((fabs(dy) > ymid) && (domain->periodicity[1] == 1))
01516           dy = fabs(dy) - 2.*ymid;
01517
01518         if ((fabs(dz) > zmid) && (domain->periodicity[2] == 1))
01519           dz = fabs(dz) - 2.*zmid;
01520
01521         rsq = dx*dx + dy*dy + dz*dz;
01522
01523         if (rsq > cutoff)
01524           continue;
01525
01526         // if not part of i's global neighbor2 list - add to phi_old
01527         if (std::find(global_neighbors_i.begin(), global_neighbors_i.end(), k) ==
    global_neighbors_i.end())
01528         {
01529           phi_old.push_back(array_coordinates());
01530           phi_old[phi_old_count][0] = k;
```

```
01531           phi_old[phi_old_count][1] = lattice[k_local.second]->darray[1][
    k_local.first];
01532
01533           phi_old_count++;
01534         }
01535       }
01536     }
01537     return old_eng;
01538   }
01539   else if (flag == 1)  // find energy of new state and restore phis
01540   {
01541     eng += old_eng;
01542
01543     for (int kk = 0; kk < phi_old_count; kk++)
01544     {
01545       eng -= phi_old[kk][1];
01546       // find new energy contribution from that site
01547       int k = phi_old[kk][0];
01548       pair_int k_local = global_to_grain(k);
01549       phi_old[kk][2] = pair->energy(k_local.first, k_local.second,
    lattice[k_local.second]->num_global_neighbors, lattice[k_local.second]->global_neighbors,
    lattice[k_local.second]->coordinates, lattice[k_local.second]->iarray[0],
    lattice[k_local.second]->darray[0], lattice[k_local.second]->darray[1]);
     // store the new energy
01550       // note: pair_energy is called instead of site_energy so that the phi update
01551       // is done automatically; if type = 0, site_energy will return 0
01552       // without updating phi
01553       eng += phi_old[kk][2];
01554       // restore the value for the phis after calculating eng
01555       lattice[k_local.second]->darray[1][k_local.first] = phi_old[kk][1];
01556     }
01557   }
01558   else if (flag == 2)    // apply the change to phi
01559   {
01560     eng += old_eng;   // eng starts off with having the same value as old_eng
01561     // the sites that need their energies recalculated are already stored
01562     // in phi_old[kk][0]
01563
01564     for (int kk = 0; kk < phi_old_count; kk++)
01565     {
01566       // find contribution from sites in phi_old and subtract it from eng
01567       eng -= phi_old[kk][1];
01568       // add the new energy when the change is accepted
01569       int k = phi_old[kk][0];
01570       pair_int k_local = global_to_grain(k);
01571       lattice[k_local.second]->darray[1][k_local.first] = phi_old[kk][2];
01572       eng += phi_old[kk][2];   // add new contribution
01573     }
01574   }
01575   return eng;
01576 }
```

void AppEAM::update_dep_flag (int *grain*) [private]

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)

```
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
01054   vec_coordinates& coordinates = lattice[grain]->coordinates;
01055
01056   int diss_old = diss_sites_count_grain[grain];
01057   int dep_old = dep_sites_count_grain[grain];  // store old value
01058
01059   double min_dist = (lattice[grain]->next_site)*((lattice[grain]->latconst)/1.01);
01060   double max_dist_x = domain->boxhi[0] - (lattice[grain]->next_site)*
    ((lattice[grain]->latconst)/1.01);
01061   double max_dist_y = domain->boxhi[1] - (lattice[grain]->next_site)*
    ((lattice[grain]->latconst)/1.01);
01062   double max_dist_z = domain->boxhi[2] - (lattice[grain]->next_site)*
    ((lattice[grain]->latconst)/1.01);
01063
01064   // resets dep_sites_count_grain
01065   dep_sites_count_grain[grain] = 0;
01066   diss_sites_count_grain[grain] = 0;
01067
01068   for (int ii = 0; ii < active_sites[grain].size(); ii++)
01069   {
01070     int i = active_sites[grain][ii];
01071
01072     // if coordnum != 12 and 0, vacant, not in proximity of another grain, site is
01073     // available for deposition
01074     if ((type[i] == 0) && (coordnum[i] != 12) && (coordnum[i] != 0)
01075        && (dep_flag[i] >= 0))
01076       dep_flag[i] = 1;  // if possible deposition site, change the flag to 1
01077     // if site is occupied, has coordnum < 10 - it can undergo dissolution
01078     // the condition in coordnum and determine_direction is to ensure that it is
01079     // at the topmost layer in a particular plane
01080     else if ((coordnum[i] < 10) && (type[i] == 1) && (coordinates[i][0] > min_dist)
01081        && (coordinates[i][0] < max_dist_x) && (coordinates[i][1] > min_dist)
01082        && (coordinates[i][1] < max_dist_y) && (coordinates[i][2] > min_dist)
01083        && (coordinates[i][2] < max_dist_z) && (!edge_check(i,grain))
01084        && ((determine_direction(i,grain) != -1) || (dep_flag[i] >= 0)))
01085     // edge sites are excluded to prevent weird things from happening to the domain
01086       dep_flag[i] = 2;  // possible dissolution site, change flag to 2
01087     else
01088     {
01089       if (dep_flag[i] >= 0)
```

```
01090         dep_flag[i] = 0;  // else, dep_flag = 0
01091     }
01092
01093   }
01094
01095   // sum over all sites - doing this in the loop was not working for some reason
01096   for (int ii = 0; ii < active_sites[grain].size(); ii++)
01097   {
01098     int i = active_sites[grain][ii];
01099     if ((dep_flag[i] == 1) && (type[i] == 0))
01100       dep_sites_count_grain[grain]++;  // update the sum
01101     else if ((dep_flag[i] == 2) && (type[i] == 1))
01102       diss_sites_count_grain[grain]++;  // update the sum
01103
01104   }
01105
01106   diss_sites_count += (diss_sites_count_grain[grain] - diss_old);
01107   dep_sites_count += (dep_sites_count_grain[grain] - dep_old);  // add new value
01108 }
```

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

i	Local index of site i
grain	Grain ID of site i
flag	Flag indicating whether function is called in site_event() or not

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.

```
00841 {
00842   int N_old = lattice[grain]->N;
00843   vec_coordinates& coordinates = lattice[grain]->coordinates;
00844   vec_int& num_neighbors = lattice[grain]->num_neighbors;
00845   vec_vec_int& neighbors = lattice[grain]->neighbors;
00846   vec_int& coordnum = lattice[grain]->iarray[2];
00847   vec_int& dep_flag = lattice[grain]->iarray[3];
00848
00849   double x_max = domain->boxhi[0];
00850   double y_max = domain->boxhi[1];
00851   double z_max = domain->boxhi[2];
00852
00853   double x_loc, y_loc, z_loc;
00854
00855   matrix& inv_rot_mat = lattice[grain]->inv_rot_mat;
00856   array_coordinates& trans_mat = lattice[grain]->trans_mat;
00857
```

```
00858   vec_int& type = lattice[grain]->iarray[0];
00859   vec_int& grainid = lattice[grain]->iarray[1];
00860
00861   // find where i is in the active sites list
00862   auto loc = std::find(active_sites[grain].begin(), active_sites[grain].end(), i);
00863
00864   // max distance away from the box boundary to be considered an edge site
00865   double max_dist = (lattice[grain]->next_site)*(lattice[grain]->latconst);
00866
00867   // if not in the list and not at the edge, update active_sites
00868   if (loc == active_sites[grain].end())
00869   {
00870     if ((coordnum[i] != 12) && (coordnum[i] != 0))
00871     {
00872       active_sites[grain].push_back(i);
00873     }
00874     for (int mm = 0; mm < num_neighbors[i]; mm++)
00875     {
00876       int m = neighbors[i][mm];
00877
00878       // condition for an active site
00879       if ((coordnum[m] != 12) && (coordnum[m] != 0)
00880         && (!active_sites_check(m, grain)))
00881       {
00882         active_sites[grain].push_back(m);
00883       }
00884       else if (((coordnum[m] == 12) || (coordnum[m] == 0))
00885         && (active_sites_check(m, grain)))
00886       {
00887         auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
00888         active_sites[grain].erase(loc2);
00889         dep_flag[m] = 0;
00890       }
00891     }
00892
00893     return;
00894   }
00895   else
00896   {
00897     // check if i is no longer an active site
00898     if ((coordnum[i] == 12) || (coordnum[i] == 0))
00899     {
00900       active_sites[grain].erase(loc);
00901       dep_flag[i] = 0;
00902     }
00903
00904     // check its neighbors too
00905     for (int mm = 0; mm < num_neighbors[i]; mm++)
00906     {
00907       int m = neighbors[i][mm];
00908
00909       if ((coordnum[m] != 12) && (coordnum[m] != 0)
00910         && (!active_sites_check(m, grain)))
00911       {
00912         active_sites[grain].push_back(m);
00913       }
00914       else if (((coordnum[m] == 12) || (coordnum[m] == 0))
```

```
00915         && (active_sites_check(m, grain)))
00916       {
00917         auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
00918         active_sites[grain].erase(loc2);
00919         dep_flag[m] = 0;
00920       }
00921     }
00922   }
00923
00924   // if site is not at the edge, then check whether it needs to grow or not
00925   auto loc2 = std::find(neighbors[i].begin(), neighbors[i].end(), -1);
00926
00927   if ((loc2 != neighbors[i].end())
00928     && (fabs(coordinates[i][0]) > max_dist)
00929     && (fabs(coordinates[i][0] - x_max) >= max_dist)
00930     && (fabs(coordinates[i][1]) > max_dist)
00931     && (fabs(coordinates[i][1] - y_max) >= max_dist)
00932     && (fabs(coordinates[i][2]) > max_dist)
00933     && (fabs(coordinates[i][2] - z_max) >= max_dist)
00934     && (loc != active_sites[grain].end()))
00935     // if one of its neighbors = -1, grow the lattice
00936   {
00937     int grow = update_grain(i, grain);  // calls check_grain in app_lattice to check
00938     // if we did grow the grain the number of sites would have increased
00939     if (grow)
00940     {
00941       std::cout << "Updating propensity...";
00942
00943       for (int m = N_old; m < lattice[grain]->N; m++)
00944       {
00945         // update all the arrays, no need to update energy
00946         // and type since the new sites start out as unoccupied
00947         grainid[m] = grain;
00948         coord_update(m,grain);
00949
00950         // check the status of the new sites
00951         if ((coordnum[m] != 12) && (coordnum[m] != 0)
00952           && (!active_sites_check(m, grain)))
00953         {
00954           active_sites[grain].push_back(m);
00955         }
00956         else if (((coordnum[m] == 12) || (coordnum[m] == 0))
00957           && (active_sites_check(m, grain)))
00958         {
00959           auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
00960           active_sites[grain].erase(loc2);
00961           dep_flag[m] = 0;
00962         }
00963       }
00964
00965       for (int m = 0; m < lattice_count; m++)
00966         check_grain_distance(m);
00967
00968       for (int m = 0; m < lattice_count; m++)
00969         update_dep_flag(m);
00970
00971       dep_diss_prob.resize(N_total, vec_double(4, 0.));
```

```
00972       // if called from site_event(), need to extend the propensity
00973       // vector to accommodate the new sites and update existing propensities of
00974       // active sites
00975       if (flag == 1)
00976       {
00977         for (int m = N_old; m < lattice[grain]->N; m++)
00978           propensity.push_back(site_propensity(grain_to_global(std::make_pair(m, grain))));
00979
00980         for (int n = 0; n < active_sites[grain].size(); n++)
00981           site_propensity(grain_to_global(std::make_pair(active_sites[grain][n], grain)));
00982
00983         solve->resize(N_total,propensity);  // resize the propensity bin
00984       }
00985
00986       std::cout << " done" << std::endl;
00987
00988     }
00989
00990     // check if i is no longer an active site
00991     if ((coordnum[i] == 12) || (coordnum[i] == 0))
00992     {
00993       active_sites[grain].erase(loc);
00994       dep_flag[i] = 0;
00995     }
00996
00997     // check its neighbors too
00998     for (int mm = 0; mm < num_neighbors[i]; mm++)
00999     {
01000       int m = neighbors[i][mm];
01001
01002       if ((coordnum[m] != 12) && (coordnum[m] != 0)
01003         && (!active_sites_check(m, grain)))
01004       {
01005         active_sites[grain].push_back(m);
01006       }
01007       else if (((coordnum[m] == 12) || (coordnum[m] == 0))
01008         && (active_sites_check(m, grain)))
01009       {
01010         auto loc2 = std::find(active_sites[grain].begin(), active_sites[grain].end(), m);
01011         active_sites[grain].erase(loc2);
01012         dep_flag[m] = 0;
01013       }
01014     }
01015   }
01016 }
```

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]

Deposition 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

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

[1] M. Paunovic and M. Schlesinger. *Fundamentals of Electrochemical Deposition*. John Wiley and Sons, 2nd edition, 2006.

[2] D. Clark, D. Wood, and U. Erb. Industrial applications of electrodeposited nanocrystals. *Nanostruct. Mater.*, **9**, 755 – 758, 1997.

[3] S. Cherevko and C.-H. Chung. Direct electrodeposition of nanoporous gold with controlled multimodal pore size distribution. *Electrochem. Commun.*, **13**, 16 – 19, 2011.

[4] S. Karuppuchamy, K. Nonomura, T. Yoshida, T. Sugiura, and H. Minoura. Cathodic electrodeposition of oxide semiconductor thin films and their application to dye-sensitized solar cells. *Solid State Ionics*, **151**, 19 – 27, 2002.

[5] J. Elias, R. Tena-Zaera, and C. Lévy-Clément. Electrodeposition of ZnO nanowires with controlled dimensions for photovoltaic applications: Role of buffer layer. *Thin Solid Films*, **515**, 8553 – 8557, 2007.

[6] J. Zhang and C. M. Li. Nanoporous metals: fabrication strategies and advanced electrochemical applications in catalysis, sensing and energy systems. *Chem. Soc. Rev.*, **41**, 7016–7031, 2012.

[7] P. C. Andricacos, C. Uzoh, J. O. Dukovic, J. Horkans, and H. Deligianni. Damascene copper electroplating for chip interconnections. *IBM J. Res. Develop.*, **42**, 567 –574, 1998.

[8] P. M. Vereecken, R. A. Binstead, H. Deligianni, and P. C. Andricacos. The chemistry of additives in damascene copper plating. *IBM J. Res. Develop.*, **49**, 3 –18, 2005.

[9] N. Shaigan, D. G. Ivey, and W. Chen. Electrodeposition of NiLaCrO3 composite coatings for solid oxide fuel cell stainless steel interconnect applications. *J. Electrochem. Soc.*, **155**, D278–D284, 2008.

[10] J. Cho and C. V. Thompson. Grain size dependence of electromigration-induced failures in narrow interconnects. *Appl. Phys. Lett.*, **54**, 2577–2579, 1989.

[11] C. Ryu, K.-W. Kwon, A. L. Loke, H. Lee, T. Nogami, V. M. Dubin, R. A. Kavari, G. W. Ray, and S. S. Wong. Microstructure and reliability of copper interconnects. *IEEE T. Electron. Dev.*, **46**, 1113–1120, 1999.

[12] C. S. Hau-Riege and C. V. Thompson. Electromigration in Cu interconnects with very different grain structures. *Appl. Phys. Lett.*, **78**, 3451–3453, 2001.

[13] R. M. Stephens and R. C. Alkire. Simulation of kinetically limited nucleation and growth at monatomic step edges. *J. Electrochem. Soc.*, **154**, D418–D426, 2007.

[14] T. O. Drews, A. Radisic, J. Erlebacher, R. D. Braatz, P. C. Searson, and R. C. Alkire. Stochastic simulation of the early stages of kinetically limited electrodeposition. *J. Electrochem. Soc.*, **153**, C434–C441, 2006.

[15] T. O. Drews, J. C. Ganley, and R. C. Alkire. Evolution of surface roughness during copper electrodeposition in the presence of additives: Comparison of experiments and Monte Carlo simulations. *J. Electrochem. Soc.*, **150**, C325–C334, 2003.

[16] M. C. Gimenez, M. G. Del Popolo, E. P. M. Leiva, S. G. Garcia, D. R. Salinas, C. E. Mayer, and W. J. Lorenz. Theoretical considerations of electrochemical phase formation for an ideal Frank-van der Merwe system: Ag on Au(111) and Au(100). *J. Electrochem. Soc.*, **149**, E109–E116, 2002.

[17] M. C. Gimenez, M. G. Del Popolo, and E. P. M. Leiva. Kinetic Monte Carlo study of electrochemical growth in a heteroepitaxial system. *Langmuir*, **18**, 9087–9094, 2002.

[18] A. C. Frank, P. T. Sumodjo, and E. P. Leiva. A Monte Carlo model for the simulation of the electrodeposition of CoNi alloys onto glassy carbon. *ECS Trans.*, **25**, 53–63, 2010.

[19] S. Frank, D. E. Roberts, and P. A. Rikvold. Effects of lateral diffusion on morphology and dynamics of a microscopic lattice-gas model of pulsed electrodeposition. *J. Chem. Phys.*, **122**, 064705, 2005.

[20] S. Frank and P. A. Rikvold. Kinetic Monte Carlo simulations of electrodeposition: Crossover from continuous to instantaneous homogeneous nucleation within Avrami's law. *Surf. Sci.*, **600**, 2470–2487, 2006.

[21] L. Guo, A. Radisic, and P. C. Searson. Kinetic Monte Carlo Simulations of nucleation and growth in electrodeposition. *J. Phys. Chem. B*, **109**, 24008–24015, 2005.

[22] F. Berthier, B. Legrand, J. Creuze, and R. Tétot. Ag/Cu (001) electrodeposition: beyond the classical nucleation theory. *J. Electroanal. Chem.*, **562**, 127 – 134, 2004.

[23] Y. Kaneko, Y. Hiwatari, K. Ohara, and F. Asa. Kinetic Monte Carlo simulation of three-dimensional shape evolution with void formation using solid-by-solid model: Application to via and trench filling. *Electrochim. Acta*, **100**, 321–328, 2013.

[24] Y. Kaneko, Y. Hiwatari, K. Ohara, and F. Asa. Kinetic Monte Carlo approach to the effects of additives in electrodeposition. *ECS Trans.*, **35**, 7–12, 2011.

[25] Y. Kaneko, Y. Hiwatari, K. Ohara, and F. Asa. Monte Carlo simulation of damascene electroplating: effects of additives. *Mol. Simul.*, **32**, 1227–1232, 2006.

[26] Y. Kaneko, Y. Hiwatari, K. Ohara, and T. Murakami. Computer simulation of thin film growth with defect formation. *Surf. Coat. Technol.*, **169-170**, 215–218, 2003.

[27] Y. Kaneko, Y. Hiwatari, K. Ohara, and T. Murakami. Monte Carlo simulation of thin film growth with lattice defects. *J. Phys. Soc. Jpn.*, **69**, 3607–3613, 2000.

[28] Y. Kaneko, S. Nishimura, Y. Hiwatari, K. Ohara, and F. Asa. Monte Carlo and molecular dynamics studies of the effects of additives in electrodeposition. *J. Korean Phys. Soc.*, **54**, 1207–1211, 2009.

[29] T. Treeratanaphitak, M. D. Pritzker, and N. M. Abukhdeir. Kinetic Monte Carlo simulation of electrodeposition using the embedded-atom method. *Electrochim. Acta*, **121**, 407–414, 2014.

[30] T. J. Pricer, M. J. Kushner, and R. C. Alkire. Monte Carlo simulation of the electrodeposition of copper: I. Additive-free acidic sulfate solution. *J. Electrochem. Soc.*, **149**, C396–C405, 2002.

[31] X. Li, T. O. Drews, E. Rusli, F. Xue, Y. He, R. Braatz, and R. Alkire. Effect of additives on shape evolution during electrodeposition I. Multiscale simulation with dynamically coupled kinetic Monte Carlo and moving-boundary finite-volume codes. *J. Electrochem. Soc.*, **154**, D230–D240, 2007.

[32] E. Rusli, F. Xue, T. O. Drews, P. M. Vereecken, P. Andricacos, H. Deligianni, R. D. Braatz, and R. C. Alkire. Effect of additives on shape evolution during electrodeposition II. Parameter estimation from roughness evolution experiments. *J. Electrochem. Soc.*, **154**, D584–D597, 2007.

[33] Z. Zheng, R. M. Stephens, R. D. Braatz, R. C. Alkire, and L. R. Petzold. A hybrid multiscale kinetic Monte Carlo method for simulation of copper electrodeposition. *J. Comput. Phys.*, **227**, 5184–5199, 2008.

[34] J. Liu, C. Liu, and P. P. Conway. Kinetic Monte Carlo simulation of the electrodeposition of polycrystalline copper: Effects of substrates and deposition parameters on the microstructure of deposits. *Electrochim. Acta*, **97**, 132 – 142, 2013.

[35] J. Liu, C. Liu, and P. P. Conway. Kinetic Monte Carlo simulation of electrodeposition of polycrystalline Cu. *Electrochem. Commun.*, **11**, 2207–2211, 2009.

[36] J. Liu, C. Liu, and P. P. Conway. Kinetic Monte Carlo simulation of kinetically limited copper electrocrystallization on an atomically even surface. *Electrochim. Acta*, **54**, 6941–6948, 2009.

[37] Y. Y. Huang, Y. C. Zhou, and Y. Pan. Simulation of kinetically limited growth of electrodeposited polycrystalline Ni films. *Physica E*, **41**, 1673–1678, 2009.

[38] N. B. Luque and E. P. Leiva. On the application of computer simulations to the study of electrochemical nanostructuring and surface phase formation. *Electrochim. Acta*, **50**, 3161 – 3178, 2005.

[39] M. S. Daw and M. I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, **29**, 6443–6453, 1984.

[40] M. S. Daw, S. M. Foiles, and M. I. Baskes. The embedded-atom method: a review of theory and applications. *Mater. Sci. Rep.*, **9**, 251–310, 1993.

[41] J. B. Adams, S. M. Foiles, and W. G. Wolfer. Self-diffusion and impurity diffusion of fcc metals using the five-frequency model and the Embedded Atom Method. *J. Mater. Res.*, **4**, 102–112, 1989.

[42] S. M. Foiles, M. I. Baskes, C. F. Melius, and M. S. Daw. Calculation of hydrogen dissociation pathways on nickel using the embedded atom method. *J. Less-Common Met.*, **130**, 465–473, 1987.

[43] G. Antczak and G. Ehrlich. *Surface Diffusion*. Cambridge University Press, New York City, 2010.

[44] M. Mariscal, E. Leiva, K. Pötting, and W. Schmickler. The structure of electrodeposits – a computer simulation study. *Appl. Phys. A*, **87**, 385–389, 2007.

[45] A. F. Voter. Hyperdynamics: Accelerated molecular dynamics of infrequent events. *Phys. Rev. Lett.*, **78**, 3908–3911, 1997.

[46] A. F. Voter. A method for accelerating the molecular dynamics simulation of infrequent events. *J. Chem. Phys.*, **106**, 4665–4677, 1997.

[47] M. R. Sorensen and A. F. Voter. Temperature-accelerated dynamics for simulation of infrequent events. *J. Chem. Phys.*, **112**, 9599–9606, 2000.

[48] A. F. Voter, F. Montalenti, and T. C. Germann. Extending the time scale in atomistic simulaton of materials. *Annu. Rev. Mater. Res.*, **32**, 321–346, 2002.

[49] A. J. Bard and L. R. Faulkner. *Electrochemical Methods: Fundamentals and Applications*. John Wiley and Sons, New York, 2nd edition, 2001.

[50] R. Cabán and T. W. Chapman. Statistical analysis of electrode kinetics measurements: Copper deposition from CuSO4-H2SO4 solutions. *J. Electrochem. Soc.*, **124**, 1371–1379, 1977.

[51] E. Budevski, G. Staikov, and W. J. Lorenz. *Electrochemical Phase Formation and Growth: An Introduction to the Initial Stages of Metal Deposition*. Advances in Electrochemical Science and Engineering. VCH, Weinheim, Germany, 1996.

[52] A. Milchev. *Electrocrystallization: Fundamentals of Nucleation and Growth*. Kluwer Academic Publishers, 2002.

[53] D. Frenkel and B. Smit. *Understanding Molecular Simulation From Algorithms to Applications*. Academic Press, San Diego, California, 2nd edition, 2002.

[54] A. Chatterjee and D. G. Vlachos. An overview of spatial microscopic and accelerated kinetic Monte Carlo methods. *J. Comput. Aided Mater. Des.*, **14**, 253–308, 2007.

[55] K. A. Fichthorn and W. H. Weinberg. Theoretical foundations of dynamical Monte Carlo simulations. *J. Chem. Phys.*, **95**, 1090–1096, 1991.

[56] C. W. Gardiner. *Handbook of Stochastic Methods*. Springer-Verlag, 2nd edition, 1985.

[57] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, **58**, 35–55, 2007.

[58] D. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 3rd edition, 2009.

[59] G. Gilmer and S. Yip. Basic Monte Carlo models: Equilibrium and kinetics. In S. Yip, editor, *Handbook of Materials Modeling*, pages 613–628. Springer Netherlands, 2005.

[60] A. Bortz, M. Kalos, and J. Lebowitz. A new algorithm for Monte Carlo simulation of Ising spin systems. *J. Comput. Phys.*, **17**, 10 – 18, 1975.

[61] D. V. Schroeder. *An Introduction to Thermal Physics*. Addison Wesley Longman, 2000.

[62] K. Binder. Static and dynamic critical phenomena of the two-dimensional q-state Potts model. *J. Stat. Phys.*, **24**, 69–86, 1981.

[63] G. H. Gilmer and P. Bennema. Simulation of crystal growth with surface diffusion. *J. Appl. Phys.*, **43**, 1347–1360, 1972.

[64] Y. Kaneko. Modeling and simulation. In K. Kondo, R. N. Akolkar, D. P. Barkey, and M. Yokoi, editors, *Copper Electrodeposition for Nanofabrication of Electronics Devices*, volume 171 of *Nanostructure Science and Technology*, pages 63–95. Springer New York, 2014.

[65] D. A. Smith. Grain boundary structure and migration. In D. Wolf and S. Yip, editors, *Materials interfaces: Atomic-level structure and properties*, chapter 6, pages 212–227. Chapman and Hall, 1st edition, 1992.

[66] C. Herzig and Y. Mishin. Grain boundary diffusion in metals. In P. Heitjans and J. Kaerger, editors, *Diffusion in Condensed Matter*, pages 337–366. Springer Berlin Heidelberg, 2005.

[67] A. Kelly and K. M. Knowles. *Crystallography and Crystal Defects*. John Wiley and Sons, 2nd edition, 2012.

[68] H. Huang, G. H. Gilmer, and T. D. de la Rubia. An atomistic simulator for thin film deposition in three dimensions. *J. Appl. Phys.*, **84**, 3636–3649, 1998.

[69] S. Ruan and C. A. Schuh. Kinetic Monte Carlo simulations of nanocrystalline film deposition. *J. Appl. Phys.*, **107**, 073512, 2010.

[70] L. A. Zepeda-Ruiz, G. H. Gilmer, C. C. Walton, A. V. Hamza, and E. Chason. Surface morphology evolution during sputter deposition of thin films - lattice Monte Carlo simulations. *J. Cryst. Growth*, **312**, 1183 – 1187, 2010.

[71] H. S. Uhm and S. K. Hwang. Three-dimensional Monte Carlo computer simulation of grain growth in electro-plated pure Ni. *Met. Mater. Int.*, **10**, 113–121, 2004.

[72] C. C. Battaile. The kinetic Monte Carlo method: Foundation, implementation, and applications. *Comput. Methods Appl. Mech. Engrg.*, **197**, 3386 – 3398, 2008.

[73] P. Bruschi, A. Nannini, and F. Pieri. Monte Carlo simulation of polycrystalline thin film deposition. *Phys. Rev. B*, **63**, 035406, 2000.

[74] H. Huang and L. Zhou. Atomistic simulator of polycrystalline thin film deposition in three dimensions. *J. Comput. Aided Mater. Des.*, **11**, 59–74, 2004.

[75] H. Huang. Texture evolution during thin film deposition. In S. Yip, editor, *Handbook of Materials Modeling*, chapter 2, pages 1039–1049. Springer Netherlands, 2005.

[76] J. E. Rubio, M. Jaraiz, I. Martin-Bragado, J. M. Hernandez-Mangas, J. Barbolla, and G. H. Gilmer. Atomistic Monte Carlo simulations of three-dimensional polycrystalline thin films. *J. Appl. Phys.*, **94**, 163–168, 2003.

[77] S. W. Levine and P. Clancy. A simple model for the growth of polycrystalline Si using the kinetic Monte Carlo simulation. *Modell. Simul. Mater. Sci. Eng.*, **8**, 751, 2000.

[78] X. Tan, Y. Zhou, and X. Zheng. Pulsed-laser deposition of polycrystalline Ni films: A three-dimensional kinetic Monte Carlo simulation. *Surf. Sci.*, **588**, 175 – 183, 2005.

[79] A. F. Voter. Introduction to the kinetic Monte Carlo method. In K. E. Sickafus, E. A. Kotomin, and B. P. Uberuaga, editors, *Radiation Effects in Solids*, volume 235 of *NATO Science Series*, chapter 1, pages 1–23. Springer Netherlands, 2007.

[80] C. C. Battaile and D. J. Srolovitz. Kinetic Monte Carlo simulation of chemical vapor deposition. *Annu. Rev. Mater. Res.*, **32**, 297–319, 2002.

[81] C. Battaile. Monte Carlo methods for simulating thin film deposition. In S. Yip, editor, *Handbook of Materials Modeling*, chapter 7, pages 2363–2377. Springer Netherlands, 2005.

[82] P. Allongue and F. Maroun. Metal electrodeposition on single crystal metal surfaces mechanisms, structure and applications. *Curr. Opin. Solid State Mater. Sci.*, **10**, 173 – 181, 2006.

[83] E. Mattsson and J. O. Bockris. Galvanostatic studies of the kinetics of deposition and dissolution in the copper + copper sulphate system. *Trans. Faraday Soc.*, **55**, 1586–1601, 1959.

[84] B. E. Conway and J. O. Bockris. On the calculation of potential energy profile diagrams for processes in electrolytic metal deposition. *Electrochim. Acta*, **3**, 340 – 366, 1961.

[85] D. Wolf. Atomic-level geometry of crystalline interfaces. In D. Wolf and S. Yip, editors, *Materials interfaces: Atomic-level structure and properties*, chapter 1, pages 1–57. Chapman and Hall, 1st edition, 1992.

[86] M. E. Straumanis and L. S. Yu. Lattice parameters, densities, expansion coefficients and perfection of structure of Cu and of Cu–In α phase. *Acta Crystallogr. A*, **25**, 676–682, 1969.

[87] S. Plimpton, C. Battaile, M. Chandross, L. Holm, A. Thompson, V. Tikare, G. Wagner, E. Webb, X. Zhou, C. G. Cardona, and A. Slepoy. Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo. Sandia report SAND2009-6226, Sandia National Laboratories, 2009.

[88] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, **104**, 1876–1889, 2000.

[89] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, **117**, 1 – 19, 1995.

[90] J. Vazquez-Arenas, M. Pritzker, and M. Fowler. Kinetic and hydrodynamic implications of 1-D and 2-D models for copper electrodeposition under mixed kinetic-mass transfer control. *Electrochim. Acta*, **89**, 717 – 725, 2013.

[91] E. Gadelmawla, M. Koura, T. Maksoud, I. Elewa, and H. Soliman. Roughness parameters. *J. Mater. Process. Technol.*, **123**, 133 – 145, 2002.

[92] H. Mantz, K. Jacobs, and K. Mecke. Utilizing Minkowski functionals for image analysis: a marching square algorithm. *J. Stat. Mech: Theory Exp.*, **2008**, P12015, 2008.

[93] J. Ferrón, L. Gómez, J. Gallego, J. Camarero, J. E. Prieto, V. Cros, A. L. Vázquez de Parga, J. J. de Miguel, and R. Miranda. Influence of surfactants on atomic diffusion. *Surf. Sci.*, **459**, 135 – 148, 2000.

[94] H.-J. Ernst, F. Fabre, and J. Lapujoulade. Growth of Cu on Cu(100). *Surf. Sci.*, **275**, L682 – L684, 1992.

[95] L. Yang, T. S. Rahman, and M. S. Daw. Surface vibrations of Ag(100) and Cu(100): A molecular-dynamics study. *Phys. Rev. B*, **44**, 13725–13733, 1991.

[96] G. Gottstein and L. S. Shvindlerman. *Grain Boundary Migration in Metals: Thermodynamics, Kinetics, Applications*. CRC Press, 2nd edition, 2009.

[97] A. Suzuki and Y. Mishin. Atomistic modeling of point defects and diffusion in copper grain boundaries. *Interface Sci.*, **11**, 131–148, 2003.

[98] P. Sonnweber-Ribic, P. Gruber, G. Dehm, and E. Arzt. Texture transition in Cu thin films: Electron backscatter diffraction vs. X-ray diffraction. *Acta Mater.*, **54**, 3863 – 3870, 2006.

[99] H. Pick, G. Storey, and T. Vaughan. The structure of electrodeposited copper I. an experimental study of the growth of copper during electrodeposition. *Electrochim. Acta*, **2**, 165 – 176, 1960.

[100] H. L. Wei, H. Huang, C. H. Woo, R. K. Zheng, G. H. Wen, and X. X. Zhang. Development of $< 110 >$ texture in copper thin films. *Appl. Phys. Lett.*, **80**, 2290–2292, 2002.

[101] B. Hong, C. Jiang, and X. Wang. Influence of complexing agents on texture formation of electrodeposited copper. *Surf. Coat. Technol.*, **201**, 7449 – 7452, 2007.

[102] L. Lu, N. R. Tao, L. B. Wang, B. Z. Ding, and K. Lu. Grain growth and strain release in nanocrystalline copper. *J. Appl. Phys.*, **89**, 6408–6414, 2001.

[103] A. Ibanez and E. Fatás. Mechanical and structural properties of electrodeposited copper and their relation with the electrodeposition parameters. *Surf. Coat. Technol.*, **191**, 7 – 16, 2005.

[104] C. H. Seah, S. Mridha, and L. H. Chan. Quality of electroplated copper films produced using different acid electrolytes. *J. Vac. Sci. Technol. B*, **17**, 2352–2356, 1999.

[105] S. Tao and D. Y. Li. Tribological, mechanical and electrochemical properties of nanocrystalline copper deposits produced by pulse electrodeposition. *Nanotechnology*, **17**, 65, 2006.

[106] L. Vitos, A. Ruban, H. Skriver, and J. Kollar. The surface energy of metals. *Surf. Sci.*, **411**, 186–202, 1998.

[107] M. Jia, Y. Lai, Z. Tian, and Y. Liu. Calculation of the surface free energy of fcc copper nanoparticles. *Modell. Simul. Mater. Sci. Eng.*, **17**, 015006, 2009.

[108] J. J. De Miguel, A. Sánchez, A. Cebollada, J. M. Gallego, J. Ferrón, and S. Ferrer. The surface morphology of a growing crystal studied by thermal energy atom scattering (TEAS). *Surf. Sci.*, **189-190**, 1062 – 1068, 1987.

[109] W. Wulfhekel, N. N. Lipkin, J. Kliewer, G. Rosenfeld, L. C. Jorritsma, B. Poelsema, and G. Comsa. Conventional and manipulated growth of Cu/Cu(111). *Surf. Sci.*, **348**, 227 – 242, 1996.

[110] A. Iwamoto, T. Yoshinobu, and H. Iwasaki. Stable growth and kinetic roughening in electrochemical deposition. *Phys. Rev. Lett.*, **72**, 4025–4028, 1994.

[111] G. Palasantzas, S. A. Koch, and J. T. M. De Hosson. Growth front roughening of room-temperature deposited copper nanocluster films. *Appl. Phys. Lett.*, **81**, 1089–1091, 2002.