University of Technology, Sydney

Doctoral Thesis

Localised Probing of Precursor Coefficients Using Electron Beam Induced Deposition and Etching

Author:
Jared Craig Cullen

Supervisor:
A Prof. Mike Ford, Prof. Milos Toth, & Dr. Charlene Lobo

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Materials and Technology for Energy Efficiency
School of Physics and Advanced Materials

January 2016

Declaration of Authorship

I, Jared Craig Cullen, declare that this thesis titled, Localised Probing of Precursor Coefficients Using Electron Beam Induced Deposition and Etching, and the work presented in it is my own.

- I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.
- I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signed: ____________________

Date: ____________________

UNIVERSITY OF TECHNOLOGY, SYDNEY

Abstract

Faculty of Science
School of Physics and Advanced Materials

Doctor of Philosophy

Localised Probing of Precursor Coefficients Using Electron Beam Induced Deposition and Etching

by Jared Craig Cullen

Electron beam induced etching (EBIE) and deposition (EBID) are direct-write deposition techniques in which an electron beam is used for chemical precursor dissociation. Both techniques are capable of nanometer-scale resolution, but applications have been limited by poor understanding of the underlying reaction mechanisms and rate parameters. Here, a hybrid Continuum-Monte Carlo model has been designed and implemented, enabling modelling of the temporal and spatial evolution of nanostructures fabricated by EBID and EBIE. This hybrid model is used to perform Arrhenius analysis of the deposition rates of nanostructures grown by EBID and EBIE, from which both precursor desorption and diffusion rate parameters can be obtained. These parameters are of fundamental interest in physical chemistry and surface science fields but also are key to optimisation of chemical vapour deposition (CVD), EBID, EBIE, and related surface processing and nanofabrication techniques. Methods used to determine the activation energy and pre-factors for desorption and diffusion are described in detail. The limitations of these methods, growth conditions needed to minimise errors, and applications to the chemistry, physics and nanotechnology communities are also discussed.

Acknowledgements

My PhD has been an eventful ride full of ups and downs, but throughout it all a number of people have helped me persevere and I would like to take the time now to thank them.

My supervisors, A Prof. Michael J. Ford, Prof. Milos Toth, and Dr. Charlene Lobo, thank you all for the support you gave me across the many years, it has been invaluable. Whether it be coming up with solutions to software bugs or bouncing around ideas for papers I could always count on all of you. Also, a special thank you to Mr. Alan Bahm for introducing me to a number of new programming concepts and just always being an email away when I needed help.

My uni mates, Mark Lockrey, Chris Elbadawi, Kit Fair, James Bishop, thanks guys for helping me in the lab and the many problems I had...talking it out really helps; I'll really miss the dumpling lunches.

My university, University of Technology, Sydney, thank you to everyone who has taught me across the many the years and for giving me a place to study in several courses for the 9+ years that I've been there.

My family, Mum, Dad, and my brother, thank you for always being there when I needed to talk about something and giving me advice or even reminding me to call when I haven't in a while...I hope I've made you proud.

My partner, Juainni, thank you most of all for sticking by me all these years, especially when it seemed to go on and on. You have no idea how much that means to me.

To the many people that I've most likely forgotten, thank you too.

-Jared

Contents

Declaration of Authorship **ii**

Abstract **iii**

Acknowledgements **iv**

List of Figures **viii**

List of Tables **xiii**

1 Introduction **1**
1.1 Project Objectives and Approach 1
1.2 Thesis Outline 2
1.3 Published Work 3

2 Literature Review **4**
2.1 Physical Processes 5
2.1.1 Adsorption 5
2.1.2 Desorption 6
2.1.3 Surface Diffusion 7
2.1.4 Electron Induced Dissociation 7
2.2 Experimental Applications 8
2.2.1 The Effect of Various Experimental Parameters 9
2.2.2 Resolution of EBID structures 11
2.3 Modelling Methods 13
2.3.1 Continuum Models 13
2.3.2 Monte Carlo Models 17
2.4 Summary 21

3 Hybrid Continuum-Monte Carlo Model **22**
3.1 Continuum Equations 25
3.1.1 Discretization of Time and Space 27
3.1.2 Verification 30
3.2 Electron Trajectory Simulation 33
3.2.1 Single Scattering Model 35
3.2.2 Parametric Model 39

3.2.3 Model Extensions 40
3.2.3.1 Backscattered & Forward Scattered Electrons 40
3.2.3.2 Secondary Electrons 40
3.2.4 Verification 43
3.3 Model Extensions 45
3.3.1 Surface Evolution & Electron Beam Projection 46
3.3.2 Surface Diffusion Modelling 46
3.3.2.1 Free Form Movement 50
3.3.2.2 Area Remapping 51
3.3.3 Verification 53

4 Hybrid Model Simulator Details 55
4.1 User Operation 55
4.2 Tutorial: How to run the EBIED simulator 58
4.2.1 Mac OS X 58
4.2.2 Ubuntu - Linux 60
4.3 Summary of Parameters within the EBIED Simulator Input File 61
4.3.1 Simulation Parameters 61
4.3.2 Electron Beam Parameters 62
4.3.3 Material Parameters 63
4.3.4 Precursor Parameters 65
4.3.5 Module Toggles 67
4.3.6 Miscellaneous Parameters 68
4.4 Summary of Outputs from the EBIED Simulator 69
4.5 Summary of Warning and Error Messages within the EBIED Simulator . . 70

5 Localized Probing of Gas Molecule Adsorption Energies and Desorption Attempt Frequencies 74
5.1 Introduction 74
5.2 Arrhenius analysis of deposition rates 76
5.2.1 Athermal adsorption flux condition 81
5.2.2 Reaction-rate limited growth condition 82
5.2.3 Negligible diffusion condition 84
5.2.4 Steady state growth condition 85
5.3 General implications for the determination of adsorbate properties 86
5.4 Conclusion 88

6 Electron Beam Induced Deposition As a Technique for the Analysis of Precursor Molecule Diffusion Barriers and Pre-Factors 89
6.1 Introduction 89
6.2 Roles of Desorption and Diffusion in EBID 90
6.3 Adsorbate Transport Through Diffusion 91
6.4 Extraction of Diffusion Energies and Pre-Exponential Factors 94
6.5 Pre-requisites 98
6.5.1 Condition 1 - Steady State Growth 99

6.5.2 Condition 2 - Significant adsorbate concentration gradient 99
6.5.3 Condition 3 - Diffusion-dominated replenishment 100
6.5.4 Condition 4 - Efficient adsorbate consumption 101
6.6 Limitations of the Arrhenius analysis method 103
6.7 Conclusion 104

7 Conclusion 106

A Diffusion Test Code 107

B Hybrid Continuum-Monte Carlo Simulator Code 112
B.1 Constants 112
B.2 Variable Structures 112
B.3 Function Prototypes 114
B.4 Simulator Core 119
B.5 EBID/EBIE & EBIED Solver 142
B.6 Electron Flux Profile 153
B.7 Linear Interpolation 157
B.8 Monte Carlo Data Collection 158
B.9 Monte Carlo Electron Trajectories 162
B.10 Monte Carlo Surface Setup 182
B.11 Read Input Parameters 184
B.12 Read Previous Simulation Data 194
B.13 Output Current Simulation Data 195
B.14 Create Output Files 199
B.15 Crank-Nicholson Solver 200
B.16 Surface Evolution 201
B.17 Print to Logfile 206

C Model Derivations 208
C.1 EBID/EBIE - Uniform Grid Spacing Derivation 208
C.1.1 Von Neumann Stability 211
C.2 EBID/EBIE - Non-Uniform Grid Spacing Derivation 216
C.3 EBIED - Derivation 219
C.3.1 Deposit Precursor Gas Concentration 219
C.3.2 Etchant Precursor Gas Concentration 221
C.3.3 Deposited Molecule Concentration 223

Bibliography 224

List of Figures

2.1 (Figure Replicated from Hoffmann, et al.[13]) The key concepts of EBIED technique: adsorption, desorption, surface diffusion, and electron beam induced dissociation of precursor molecules. The particular precursor determines whether the resultant dissociation product causes deposition or etching. 4

2.2 (Figure Replicated from Li, et al.[16]) Schematic diagram of the potential energy as a function of the distance from the surface. 5

2.3 The electron cross section of gaseous H_2O at various electron energies [24]. 8

2.4 (Figure Replicated from Schoenaker et al.[50]) (a) Dependence of the etching depth on the etching time for several beam currents. (b) Etching rate and yield for different beam currents. The experiment was performed by using a beam energy of 2 kV. The rate and yield calculations do not take into account the peripheral etching. 11

2.5 (Figure Replicated from Choi et al.[62]) (a) Deposition of tungsten pillar under different biased conditions. (b) Scattering of electron (SE, BSE, and PE) on the tungsten pillar under different biases. 12

2.6 (Figure Replicated from Lobo, et al. [69]) (a) Growth rate plotted as a function of radius (r) from the beam axis, calculated using currents of 0.4, 0.5, 0.52, 0.53, 0.55, 0.6, 0.7, 0.8 and 1 nA. Also shown is the total electron flux profile from figure 1(d). (b) Surface plots of the 0.4, 0.5 and 0.6 nA deposition rate profiles (linear scale) ($P_d = 10^{-2}$ Pa). 15

2.7 (Figure Replicated from Bishop et al.[63]) Steady state adsorbate dissociation rates calculated using Eqs. (4)-(6) as a function of substrate temperature, and the corresponding deposition rates measured using tetraethoxysilane precursor (○). 16

2.8 (Figure Replicated from Silvis-Cividjian et al.[76]) A typical curve showing the evolution of the cone diameter. 18

2.9 (Figure Replicated from Smith et al.[74]) Normalized comparison of the MTL pillar shape, RRL pillar shape, and Gaussian beam profiles. These profiles were taken from the 1 keV MTL and RRL pillars at the same height (12.5 nm). 100 000 random Gaussian samples of a 3 nm diameter beam superimposed on the substrate surface to show the beam profile. 19

2.10 (Figure Replicated from Smith et al.[75]) Cross sections through the top 50 nm of the pillars at varying diffusion coefficients. Top row, left to right: 1.0×10^{-8}, 1.0×10^{-9}, and 1.0×10^{-10} cm^{2}s^{-1}. Bottom row, left to right: 1.0×10^{-11}, and 0.0 cm^{2}s^{-1}. 20

3.1 (Figure Replicated from Utke, et al. [4]) AFM image and line scans of FEB deposits from $Cu(hfac)_2$ precursor. Exposure times are indicated. Indented apex shapes are due to depletion. Dashed lines represent gaussian fits of each deposit. 23
3.2 (Figure Adapted from Smith, et al. [74] with a CASINO Monte Carlo simulation of electron trajectories overlayed.) 2D time-resolved cross-sectional profiles and deposition events (based on electron type) from the gas dynamics simulations. (a) Initial monolayer (ML) present (run 6). (b) Surface diffusion (run 7). (c) Surface diffusion + boundary source (run 8). The normalized sample size is 100 000 electrons. 25
3.3 Representation of three finite difference methods used to solve differential equations in time (n) and space (x): (A) Crank-Nicholson, implicit method. (B) Backward Euler, implicit method. (C) Forward Euler, explicit method. 28
3.4 Example of particle diffusion, where a single spike of concentration spreads over time due to diffusion. Each curve is 10% into the total duration of the simulation except the first curve, which is the starting condition. This example has $D = 1.0 \times 10^8$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 1.0 \times 10^{-8}$ s with a total simulation duration of 1.0×10^{-7} seconds. 29
3.5 Example of particle diffusion, where a single spike of concentration spreads over time due to diffusion. Each curve is 10% into the total duration of the simulation except the first curve, which is the starting condition. This example has $D = 1.0 \times 10^8$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 5.0 \times 10^{-9}$ s with a total simulation duration of 1.0×10^{-7} seconds. 30
3.6 Example of particle diffusion, where a single spike of concentration spreads over time due to diffusion. Each curve is 10% into the total duration of the simulation except the first curve, which is the starting condition. This example has $D = 1.0 \times 10^8$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 1.0 \times 10^{-9}$ s with a total simulation duration of 1.0×10^{-7} seconds. 31
3.7 The dimensionless diffusion coefficient, D', as a function of spatial step, Δx, and time step, Δt, for a diffusion coefficient of 1.0×10^8 Å/s. The green region defines the parameter space where the dimensionless diffusion coefficient is below 0.5 and the red above 0.5. 31
3.8 The initial precursor concentration as a function of time, (A) The concentration from two separate simulations where one is EBID and the other EBIE, (B) The precursor concentration over time from a EBIED simulation. 32
3.9 (Figure Replicated from Lobo, et al. [69]) (a) Growth rate plotted as a function of radius (r) from the beam axis, calculated using currents of 0.4, 0.5, 0.52, 0.53, 0.55, 0.6, 0.7, 0.8 and 1 nA. Also shown is the total electron flux profile from figure 1(d). (b) Surface plots of the 0.4, 0.5 and 0.6 nA deposition rate profiles (linear scale) ($P_d = 10^{-2}$ Pa). 33
3.10 Comparison between the continuum component of the hybrid model used here and the continuum model published previously by Lobo et al. [69]. The EBID rates were calculated at a distance of ~ 1.5 $\AA$ from of the electron beam axis as a function of electron beam current. Both models show the same decrease in growth rate with increasing current, caused by adsorbate depletion near the beam axis. 34

3.11 (Replicated from David Joy, 1995[88]) The scattering path of an electron with the corresponding step length and angle components. 38
3.12 The original Parametric Model, where all secondary electrons generated are emitted through the closest surface bin. 41
3.13 The first modification to the Parametric Model, where all the secondary electrons generated are distributed evenly over the closest surface bins. . . . 42
3.14 The second modification to the Parametric Model, where all the secondary electrons generated are distribution evenly over the closest surface bins, however the number in each bin is modified by its solid angle to the primary electron trajectory. 42
3.15 Comparison between the Monte Carlo component of the hybrid model used here and the Monte Carlo model CASINO [80]. The dependence of the backscattered electron coefficient on electron beam energy calculated the hybrid model is in excellent agreement with that calculated by CASINO. . 44
3.16 Comparison between the secondary electron yield on a silver surface as a function of electron beam energy taken from literature data and the hybrid continuum-Monte Carlo model. 44
3.17 The secondary electron yield on a silver surface as a function of electron beam tilt. The two distinct trends are consistent with known behaviour of the depth and size of the electron beam interaction volume with beam tilt. 45
3.18 Two EBID simulations were performed for a maximum duration of 13.7 seconds, each plot is shown at different stages within the entire simulation beginning at 25% through to 100% of the maximum duration. (A) Grown without surface evolution the deposit growth direction is upwards over all space; this is confirmed in (B) where the deposit structures have been normalised. (C) Grown with surface evolution the deposit growth is in the surface normal direction; again confirmed in (D). 47
3.19 The electron flux profile (primary electrons only) from an EBID simulation with electron beam projection switched on/off. The simulation maximum duration was 13.7 seconds, and each plot is shown at different stages within the entire simulation beginning at 25% through to 100% of the maximum duration. We observe as the simulation progress the electron flux profile changes in shape to reflect the evolving surface, which would be similar to that shown in Figure 3.18. 48
3.20 The translation between the physical surface A) and the computational surface B). The physical surface is two dimensional in r & z and the computational surface is one dimensional in r, where both surfaces are radially symmetric. The path length between the points along the physical surface is, s. 49
3.21 (Figure adapted from Sobey [94]) Non-Uniform Grid Spacing Schematic. . . 51
3.22 Realistic Non-Uniform Grid Spacing Example, Schematic. 51
3.23 The first two steps of the area remapping process required to ensure correct diffusion behaviour for an evolving surface. (1) calculate the initial surface area between each point along the flat substrate; (2) calculate the surface area between each point along the now evolving nano-structure surface. An example of the calculated area between two points is highlighted in grey. . . 52

3.24 A conical frustum is defined as a cone with the tip removed, where R_1 is the base radii, R_2, is the top radii, h is the height and s is the slant height[95]. 53
3.25 A comparison of the time-evolution of the actual volume of a deposit simulated by the hybrid model, and the volume expected from the total number of molecules dissociated by electrons. The two volumes are in excellent agreement, with only minor differences of <1% observed in the residuals. . 54

4.1 EBIED Simulator Flowchart describing the general flow and order of modules. 56
4.2 Example input text file for the EBIED Simulator. 57
4.3 Example output text file for the EBIED Simulator. 59

5.1 A series of deposits shown as a function of time and temperature, with an insert depicting the typical progression of a deposit's FWHM with time. The point where the FWHM no longer increases with time is considered steady state growth. 79
5.2 Plots of $\ln(\partial h/\partial t)$ versus $1/T$ simulated using electron beam currents in the range of 0.01 pA – 1 nA in the absence of diffusion. Also shown are the corresponding straight line fits used to obtain the activation energies plotted in Figure 5.6. 80
5.3 Precursor coverage (Θ) plotted as a function of pressure (P) and substrate temperature (T) in the limit of zero electron flux ($f \to 0$). The purple region indicates the range of P and T over which the effect of Θ on Arrhenius analysis of EBID rates is negligible. 82
5.4 Activation energy obtained by Arrhenius analysis of the EBID rate ($\partial h/\partial t$) simulated at a number of temperature windows between 250 K and 450 K. The adsorption energy (E_a) of 666 meV is shown as a dashed line. The top axis shows the precursor coverage (Θ) corresponding to each temperature shown on the bottom axis. The activation energy diverges from E_a as $\Theta \to 1$. [Each datapoint was calculated from EBID rates simulated over a temperature window ΔT in the range of 20 to 50 K.] 83
5.5 Prefactors corresponding to the activation energies shown in Figure 5.4. The desorption attempt frequency (k_0) of 10^{13} Hz is shown as a dashed line. The top axis shows the precursor coverage (Θ) corresponding to each temperature shown on the bottom axis. The prefactor diverges from k_0 as $\Theta \to 1$. 84
5.6 Dependence of activation energy on beam current simulated in the absence of diffusion ($D_a = 0$) and in the presence of diffusion ($D_a = D_0 e^{-E_d/(k_B T)}$, where E_d is the diffusion energy). Also shown is a plot of adsorbate depletion ($\Theta_{r\to0}/\Theta_{r\to\infty}$) simulated at the beam axis ($r \to 0$) in the absence of diffusion. The activation energy diverges from the adsorption energy (E_a) of 666 meV as the extent of depletion approaches 1. The precursor pressure was 0.01 Pa and the temperature was varied from 400 to 450 K. 85

6.1 Effects of diffusion on the shapes of deposits grown by EBID. (a) A series of deposits simulated as a function of temperature. At low temperatures, the deposit geometry is unaffected by diffusion, but at elevated temperatures each deposit contains a characteristic 'ring' generated by adsorbates supplied through surface diffusion. (b) Steady state vertical growth rates (R) calculated as a function of distance (r) from the electron beam axis at a number of temperatures (T_n). All simulations were performed using a Gaussian electron beam under conditions of high adsorbate depletion near the beam axis. The normalized electron flux profile $f_N(r)$ is shown as a dashed curve in (b). 92

6.2 (a) Steady state plots of the driving force of diffusion (c) versus distance (r) from the electron beam axis at a number of temperatures (T_n). Each $c(r)$ profile contains two distinct regions corresponding to the source and sink of adsorbates that diffuse along the surface and are consumed in EBID. The sink and source are separated by r_o, shown as a dashed line at 125 nm. (b) Corresponding adsorbate coverage profiles ($\Theta(r)$), and the normalized electron flux profile (($f_N(r)$, dashed grey curve). 93

6.3 (a) The fluence (C) found by integrating $c(r)$ over the sink ($0 \leq r \leq r_o$) shown in Figure 6.2(a), plotted for a number of temperatures T_n. (b,c) Corresponding plots of R_{VD} and R_{VD}/C versus T_n. The Arrhenius analysis method yields good approximations to E_D and D_o at temperatures between ~ 220 K and ~ 275 K. 95

6.4 Arrhenius plots used to generate the data in Figure 6.5(a) at temperatures (T_n) of 150, 200, 250 and 300 K. 96

6.5 (a) Activation energy (red) and pre-exponential factor (blue) obtained by Arrhenius analysis of R_{VD}/C at a number of temperatures T_n. The quantities are approximately equal to E_D and D_o (shown as dashed lines) over the temperature window $220 \lesssim T_n \lesssim 275$ K. (b) Diffusion coefficient (D) versus temperature (dashed black curve), and diffusion coefficients (red diamonds) calculated using the activation energies and pre-exponential factors in (a). . . 97

6.6 The fluence C plotted as a function of electron beam current. 100

6.7 Maximum flux of diffusing adsorbates ($\max[D\nabla^2 N_a(r)]$) plotted at a number of temperatures T_n. The adsorption flux (sF) is shown as a dashed horizontal line. 102

6.8 Activation energy (red) and pre-exponential factor (blue) obtained by Arrhenius analysis of R_{VD}/C at a number of temperatures T_n. The volume $R_{VD}(T_n)$ was estimated by subtracting (a) $R(r, T_{min} = 120$ K) and (a) $R(r, T_{min} = 150$ K) from each $R(r, T_n)$ profile, and integrating the resulting curves over r. 105

List of Tables

3.1 Realistic Non-Uniform Grid Spacing Example. The fifth point was assumed to be repeated in order to calculate the required data. 51

Chapter 1

Introduction

Electron Beam Induced Etching and Deposition (EBIED) enables high resolution, direct write etching and deposition though chemical reactions driven by electron-dissociated precursor molecules. The EBIED technique is becoming an increasingly popular direct write method for the deposition or removal of material. The material specificity and high resolution of the EBIED technique is owed to the multitude of combinations available between substrate and gaseous precursor molecules with the electron beam.

1.1 Project Objectives and Approach

The main objective of this research is to develop a new simulation method, extending those in the literature to provide greater understanding of the EBIED process and further insights into experimental observations. The simulation method developed in this research builds upon two of the most common modelling methods in the literature, Continuum[1–5] and Monte Carlo[6–9] models, discussed in greater detail in Chapter 2, where by combining these two models their respective pitfalls are overcome. This combination of modelling methods is coined the 'hybrid model'. It will later be shown to provide a fast and accurate method for the simulation of EBIED structures.

To provide greater understanding of the EBIED process and further insights into experimental observations the hybrid model is applied to the determination of the precursor

parameters for the two processes, desorption and diffusion. The methods and in-depth investigation discussed in this thesis advance the EBIED field by identifying the dominant conditions needed to probe these precursor parameters and also further our understanding of their role.

1.2 Thesis Outline

In Chapter 2, the background of the EBIED technique is discussed with detailed information of the key processes, and highlights some of the current questions in the literature. Also, discussed is the background to both Continuum and Monte Carlo models highlighting respective advantages and disadvantages.

In Chapter 3, details of the current hybrid model are discussed including the various additions and its concepts, the discussion identifies the need for this hybrid approach. Also included is verification of each component against expected results or if available those from literature.

In Chapter 4, details of how the EBIED simulator works are described, with information on its operation, data flow, and explanations of its inputs, and outputs.

In Chapter 5, a method to calculate the desorption energy and attempt frequency of a precursor molecule is detailed, particularly highlighting a set of conditions needed to ensure desorption is the dominant process under investigation.

In Chapter 6, similar to Chapter 5 a method to calculate the diffusion energy and attempt frequency of a precursor molecule is detailed, again highlighting a set of conditions needed to ensure diffusion is the dominant process under investigation. Also detailed is how the precursor concentration gradient effects diffusion and this is very important in the analysis of diffusion.

In Chapter 7, the project is concluded with final remarks.

1.3 Published Work

The research presented in Chapter 5 of the same title has been published in the Journal of Physical Chemistry C on the 18/06/2015.

- J Cullen, A Bahm, C J Lobo, M J Ford, and M Toth. Localized Probing of Gas Molecule Adsorption Energies and Desorption Attempt Frequencies. *J. Phys. Chem. C*, 119(28):15948–15953, 2015

The research presented in Chapter 6 of the same title has been accepted by ACS Applied Materials & Interfaces on the 4/09/2015.

- Jared Cullen, Charlene J. Lobo, Michael J. Ford, and Milos Toth. Electron-Beam-Induced Deposition as a Technique for Analysis of Precursor Molecule Diffusion Barriers and Prefactors. *ACS Appl. Mater. Interfaces*, September 2015

The Monte Carlo module in the hybrid model contributed to the work by Steven Randolph, et al. published in Applied Physics Letters on the 21/11/2011.

- Steven Randolph, Milos Toth, Jared Cullen, Clive Chandler, and Charlene Lobo. Kinetics of Gas Mediated Electron Beam Induced Etching. *Appl. Phys. Lett.*, 99 (21):213103, 2011

Chapter 2

Literature Review

Electron Beam Induced Etching and Deposition enables high resolution, direct write deposition/etching though chemical reactions driven by electron- dissociated precursor molecules. This technique can be broken up into four key processes: adsorption, desorption, surface diffusion and electron induced dissociation of precursor molecules, and these are depicted in Figure 2.1. These four processes lead to a wide variety of outcomes when the combination of electron-gas and electron-solid interactions change as the deposit evolves over time. The physical processes, experimental applications, and common modelling methods of EBIED are discussed in the subsequent sections.

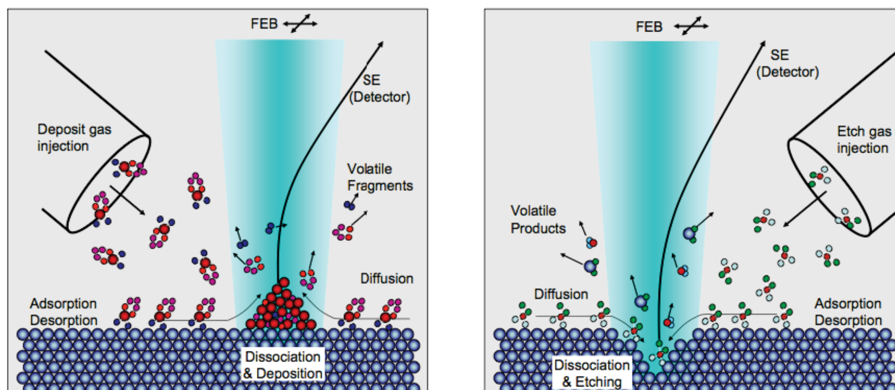

FIGURE 2.1: (Figure Replicated from Hoffmann, et al.[13]) The key concepts of EBIED technique: adsorption, desorption, surface diffusion, and electron beam induced dissociation of precursor molecules. The particular precursor determines whether the resultant dissociation product causes deposition or etching.

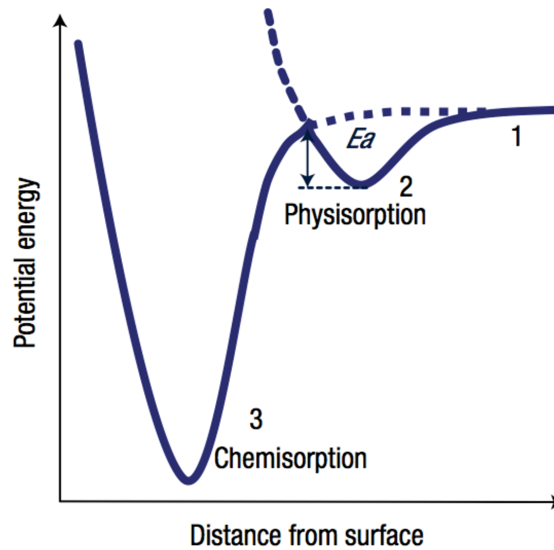

FIGURE 2.2: (Figure Replicated from Li, et al.[16]) Schematic diagram of the potential energy as a function of the distance from the surface.

2.1 Physical Processes

2.1.1 Adsorption

When a gaseous precursor molecule impacts the surface it will be either be reflected and returned to the gas phase or trapped in a physisorption well[14] (van der Waals forces) depending on its kinetic energy. The adsorbed molecule, is free to diffuse on the surface and after a certain amount of time and given enough thermal energy it can escape this physisorption well and return to the gas phase. Nevertheless, if there is sufficient thermal energy the molecule can thermally decompose, resulting in the chemisorption of the molecule onto the surface that has a relatively deep energy well. Chemisorption[14] is typically an undesirable process as deposition/etching occurs wherever the molecule sticks to the surface and is not localised to the electron beam interaction area. In one case, however, chemisorption has been shown to increase growth rates and deposit purity without the de-localised growth[15]. The relative depth of a physisorption and chemisorption energy well is shown in Figure 2.2, where the large energy difference is demonstrated.

As molecules adsorb to the surface the maximum concentration at different partial pressures is the result of the adsorption isotherm. Two common isotherms are Langmuir[17]

and Brunauer-Emmett-Teller[18] (BET), each one describes the rate of increase and maximum concentration with gas pressure. The Langmuir isotherm assumes surface coverage of adsorbed precursors is limited to one mono-layer on a homogeneous surface. The BET isotherm allows for the surface coverage to increase beyond a single mono-layer by forming a number of stacked molecular layers, where each subsequent layer is able to stick to the layer beneath it, thus creating a multi-layered system.

Each isotherm is relevant in different experimental systems, however, the work contained in this thesis assumes the Langmuir isotherm. Multi-layer adsorption is only expected if the substrate surface is much cooler than the precursor gas[19–21]. The functional form defining the Langmuir isotherm is shown in Equation 2.1, where s is the sticking coefficient, F is the precursor flux, and Θ is the surface coverage.

$$sF[1-\Theta] \tag{2.1}$$

The precursor flux is defined as,

$$F = \frac{P}{\sqrt{2\pi m_g k_B T_g}}, \tag{2.2}$$

where P is gas pressure, m_g is the gas molecule mass, k_B is Boltzmann's constant and T_g is gas temperature.

2.1.2 Desorption

When a molecule is adsorbed to the surface given enough time and thermal energy, the molecule can escape the physisorption energy well and return to the gas phase. This process is quantified in Equation 2.3[22], where τ is the mean residence time on the surface with E_{des} is the adsorption energy, τ_o is the attempt frequency for desorption, k_B is Boltzmann constant, and T is the temperature.

$$\tau = \tau_o e^{-E_{des}/k_B T} \tag{2.3}$$

2.1.3 Surface Diffusion

Surface diffusion is where an adsorbed molecule has sufficient energy to jump into an adjacent unoccupied surface site, but this energy must be less than the desorption energy barrier, otherwise desorption would occur. The diffusion equation (Equation 2.4) describes the distance per unit time the molecule can diffuse along the surface, with the Laplacian term defining the coordinate system.

$$\frac{\partial N_a}{\partial t} = D\nabla^2 N_a \tag{2.4}$$

The precursor specific diffusion term, D, is similar to the desorption term, however, the scaling with temperature is the inverse, see Equation 2.5[23] where E_{diff} is the diffusion energy barrier, and D_o is the diffusion attempt frequency.

$$D = D_o e^{-E_{diff}/k_B T} \tag{2.5}$$

2.1.4 Electron Induced Dissociation

When an electron interacts with a precursor molecule, whether it be in the gas phase or adsorbed to the surface, it can break a bond typically resulting in the formation of both volatile and non-volatile products. In EBID these non-volatile products form the deposit structure and the volatile products are removed via gas extraction systems. In EBIE, however, the products formed can react with the surface, resulting in volatile products that are then removed from the system.

The probability of an electron interacting with a precursor molecule is dependent on the electron cross section, and it is a function of the electron energy. Due to the large variation of electron energies between primary, backscattered, secondary electrons and the number of possible reaction pathways the electron cross section is expressed as a net value. This large variation is demonstrated in Figure 2.3 where the electron cross section of water over a number of reaction pathways and energies is shown. In Figure 2.3 a number of reaction products are reported each with its own energy dependent cross section grouped into either dissociative attachment or dissociative ionisation; many other reaction pathways do

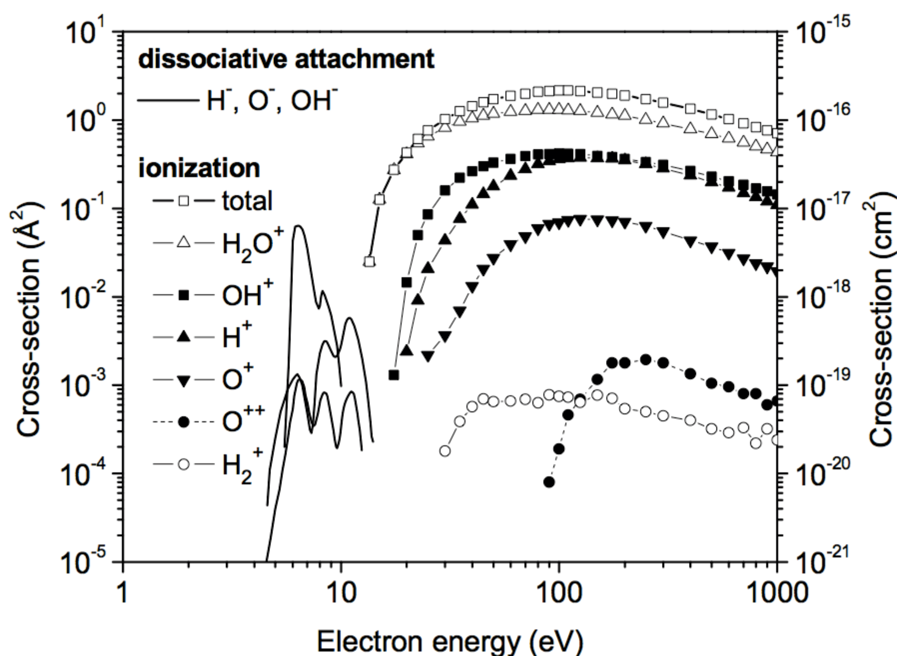

FIGURE 2.3: The electron cross section of gaseous H_2O at various electron energies [24].

exist[25] but these are not discussed here. Dissociative attachment is the process where a low energy electron interacts with a neutral molecule combing to form an excited anion, and subsequently dissociates into neutral and anionic fragments[26], $AB + e^- \rightarrow AB^- \rightarrow A^- + B$. Dissociative ionisation occurs when a high energy electron impacts an electron of a neutral molecule knocking it out of its orbital state. This dissociates the molecule into neutral and cationic fragments[27], $AB + e^- \rightarrow AB^+ \rightarrow A + B^+ + 2e^-$. At this point whether each reaction product contributes to deposition; etching; or is non-reactive and extracted from the system is dependent on the molecule formed.

2.2 Experimental Applications

The experimental origin of EBID can be traced back to the early electron microscopy literature on carbon contamination[28–30] where hydrocarbons adsorbed to the sample surface would decompose under the incoming electron flux. Methods were devised to combat this carbon contamination, but it can be produced intentionally by focusing the electron beam on a single spot for an extended period of time. From an unwanted side

effect, EBID has evolved during the past 30 years into a powerful direct write technique with one of the first applications by Broers et al. in 1976[31].

In the more recent past, EBID has been used to grow a variety of nanostructures including, nanodots and nanowires[32–37], structure patterning[38], dielectric nanoparticles[39], electrical contacts[40], and plasmonic nanostructures[41]. The EBID technique has been advanced by particular research groups, who have focused on the improvement of structure resolution, and also identifying the links between the various experimental parameters.

Extensive literature reviews by Randolph et al.[42], van Dorp et al.[43], and Utke et al.[19] are worthwhile research documents detailing a number of topics which include: the physical processes that occur in EBID/EBIE, effects of various experimental conditions and applications, and also a discussion on the precursor molecules used in EBID/EBIE. Utke et al.[44] have also written and edited work from a number of researchers into a comprehensive book on EBID/EBIE that begins with the history of the technique through to its applications.

2.2.1 The Effect of Various Experimental Parameters

In an experimental system there are many parameters which can be varied, each effecting the size, shape, and growth rate of a deposit/etch pit. These parameters include, electron accelerating voltage (electron energy), electron beam current, temperature, precursor gas pressure, dwell time (growth time), and substrate temperature. The number of combinations between all these parameters makes any prediction quite difficult. Certain trends do appear when one parameter is controlled in isolation. Identification of these trends can be found in EBIED literature[45–52].

Rack et al.[45] in 2003 used EBIE to deduce the link between electron accelerating voltage and the etch rate and the geometry of lattice damage, where the researchers found a linear trend between decreasing etch rate and increasing accelerating voltage. Conversely, Wang et al.[46] showed the opposite trend where increasing accelerating voltage lead to an increase in etch rate. From these two studies, both within the same year, it is not clear what the relationship is, but from the author's research, one would expect the etch rate

to decrease as the electron accelerating voltage increases due to the deeper electron beam interaction volume reducing the number of secondary electrons at the surface.

Later in 2005, Randolph et al.[47] (the same research group as Rack et al.[45]) investigated several factors that effect the etch rate of Silicon Dioxide with Xenon Difluoride. The researchers found that etch rate decreases with increasing electron accelerating voltage, decreasing electron beam current, and with increasing dwell times. In this work, Randolph et al. discussed the cause of the decreasing etch rate with increasing beam energy to be electron stimulated desorption (ESD). ESD is the process by which adsorbed species are removed from a surface by an electron-induced excitation process[48].

Lassiter et al.[2] also discuss the etch rate of Silicon Dioxide with Xenon Difluoride in relation to the precursor surface diffusion rate, etch product residence time, and etch product dissociation probability. The researchers found that the magnitude of these three parameters does effect the etch rate but in particular to the primary etch region; at the centre of the electron beam flux profile or at its fringe.

Roediger et al.[49] and Schoenaker et al.[50] in 2011 showed the same behaviour with the etch rate decreasing with increasing electron accelerating voltage. The researchers attributed the decrease to the decreasing number of secondary electrons contributing to the etching process. Secondary electron yield is known to depend on the electron beam accelerating voltage[53] and peaks at lower energies[53, 54] confirming their hypothesis.

In perhaps a more obvious relationship, the growth rate of nano structures has been found to increase with increasing electron beam current by several authors[47, 49, 50]. The experimental parameters, precursor gas pressure and dwell time (growth time), show an increasing relationship with growth (etch) rate[49, 50, 55]. These three experimental parameters control the supply of both precursor molecules and electrons, as discussed previously, which make up the key processes for growth in EBID/EBIE. From these key processes researchers can conclude that should either process become deficient in its supply it would be the rate limiting step in nano structure growth. This rate limited growth has been presented by Schoenaker et al.[50] (replicated in Figure 2.4) where: as the electron current increased, the etch rate has begun to saturate, showing mass transport limited (by the arrival of precursor molecules) growth. Researchers can relate this to many standard

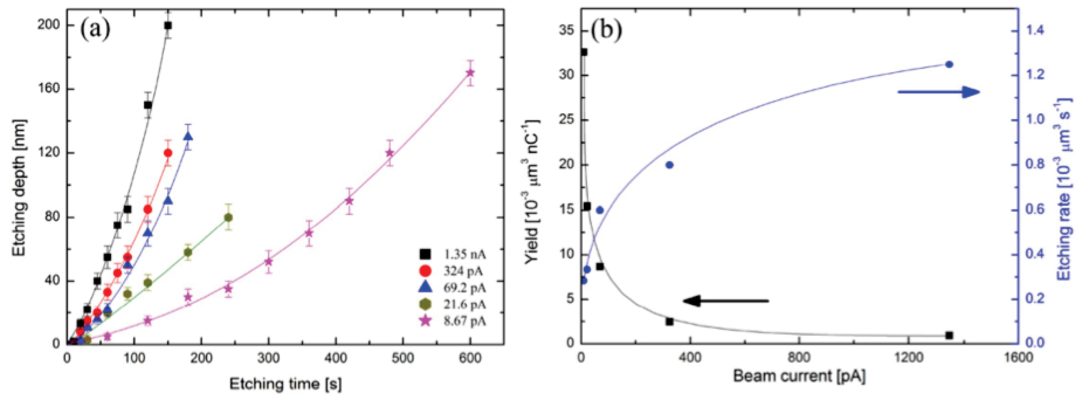

FIGURE 2.4: (Figure Replicated from Schoenaker et al.[50]) (a) Dependence of the etching depth on the etching time for several beam currents. (b) Etching rate and yield for different beam currents. The experiment was performed by using a beam energy of 2 kV. The rate and yield calculations do not take into account the peripheral etching.

chemical reactions where the reaction rate between two species can be limited by either component. In this case the number of precursor molecules or electrons.

Recently, the research by Martin et al.[52] has demonstrated that by cooling the substrate surface the etch rate of a number of precursor/substrate systems increases dramatically, where negligible at room temperature. The increase in etch rate was attributed to the increased residence time of the precursor molecules and also the greater number of available surface sites due to reduced surface diffusion of other molecules. The importance of available surface sites and their effect on etch rates has also been discussed by Martin et al.[51]. The researchers found the electron beam damage (or restructuring) which occurs during EBIE increases the number of active sites as the etch reaction takes place thereby increasing the etch rate.

2.2.2 Resolution of EBID structures

The resolution of EBID structures has consistently been the topic of many literature articles that commonly report lateral widths below 100 nm[56–59] and in one case van Dorp et al.[33] reported an averaged full width at half maximum of 1.0 nm. The key parameters and processes that affect structure resolution are: the rate of precursor arrival[60]; the current and acceleration voltage of the incident electrons[61]; and the influence of secondary electrons[62].

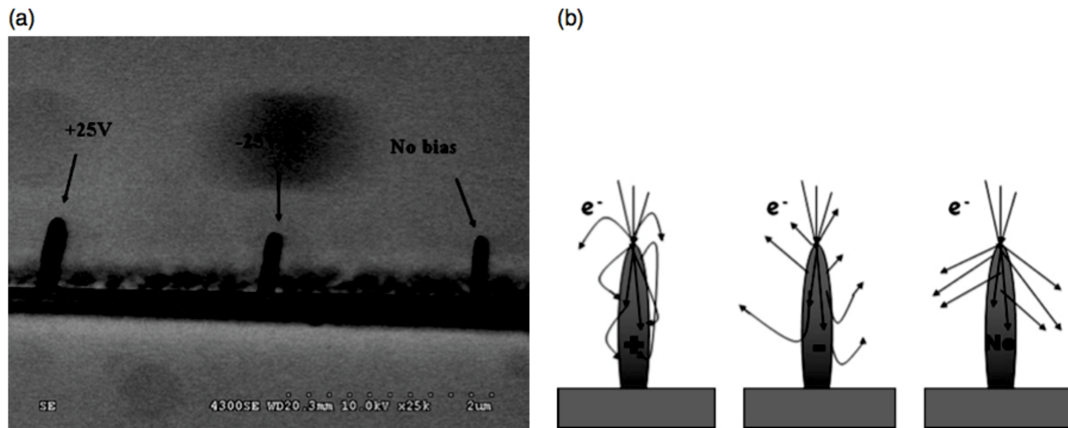

FIGURE 2.5: (Figure Replicated from Choi et al.[62]) (a) Deposition of tungsten pillar under different biased conditions. (b) Scattering of electron (SE, BSE, and PE) on the tungsten pillar under different biases.

Choi et al.[60] showed the growth of SiO_x from TEOS and Tungsten from WF_6 across a range of different precursor pressures at a fixed electron beam current. The researchers found an increasing height growth rate with pressure, and conversely, a decreasing lateral growth rate with pressure (thereby improving resolution). This result is consistent with the rate limiting behaviour discussed earlier; at higher gas pressures the growth regime becomes reaction rate limited (electron limited) resulting in the majority of growth to occur under the primary electron beam area.

Later in 2007, Choi et al.[62] investigated the role of secondary electrons during EBID growth and its effect on resolution. As the deposit structure grows the number of secondary electrons exiting through the side walls increases to a maximum value. It was proposed by Choi et al. that these secondary electrons were the primary cause of lateral growth[62]. To study this effect Choi et al. used substrate biasing to deflect or attract secondary electrons to/away from the deposit structure during growth, see Figure 2.5. Results showed an increase in lateral width with positive bias (attraction of SEs) and no change with negative bias (deflection of SEs); this result is expected but did not improve structure resolution. The role of secondary electrons continues to be studied, but primarily by computer simulation due to the extra control provided, see section 2.3.

2.3 Modelling Methods

EBIED computer simulations are used as a tool to understand and characterise processes and observations recorded in experimental systems where absolute control over conditions are necessary. The simulation methods used in the literature to date, can be split into two techniques, Continuum models[1–4] and Monte Carlo models[6–9]. Utke et al.[19] in 2008 wrote a detailed literature review of Continuum and Monte Carlo modelling methods including a review of different precursor molecules, methods to control and characterise nano-structures, and applications of EBIED. A more recent review by Toth et al.[5] has been published which reviews in great detail Continuum modelling methods for EBIED and Monte Carlo methods for simulation of precursor gas flux.

2.3.1 Continuum Models

Continuum models [1–4], solve differential equations that describe the rates of change of adsorbate concentrations on the substrate surface as a function of time and space. The first Continuum model (see Equation 2.6) was published by Robert W Christy in 1960[1] where he described the rate of deposition of thin films grown via the electron beam irradiation of silicone oil vapour as a function of substrate temperature, electron beam current density, and oil vapour pressure.

$$\frac{dN}{dt} = F - \frac{N}{\tau} - \sigma f N \tag{2.6}$$

In Equation 2.6, N is the number of adsorbed molecules on the surface, F is arrival rate of molecules from the gas phase, τ is mean residence (desorption) time of adsorbed molecules, σ is cross section for electron-molecule interactions, and f is electron flux.

Christy's model has been expanded in recent years by a number of different authors to include such physical processes as multiple precursors for deposition and etching within the same system[3], surface diffusion of adsorbed species[4], precursor flow rate into high aspect ratio etch pits[12], and activated chemisorption of adsorbates[63]. Many other physical processes are discussed in the review by Toth et al[5]. The most common form of the EBID Continuum model is shown in Equation 2.7, where each key process is grouped

and ordered as, adsorption, desorption, electron induced dissociation, and diffusion.

$$\frac{\partial N_a}{\partial t} = sF[1-\Theta] - \frac{N_a}{\tau} - \sigma f N_a + D\nabla^2 N_a \tag{2.7}$$

This Continuum model has been used by Utke et al.[4] to demonstrate how the various input parameters of EBID, that affect the lateral width of structures, can be simplified to a particular growth regime: reaction-rate limited or mass-transport limited. Reaction-rate limited growth (RRL), mentioned earlier, occurs when the EBID reaction is electron limited and mass-transport limited growth (MTL) occurs when the reaction is precursor limited. The relationship between lateral width and the particular growth regime is an excellent way to combine several experimental parameters together. A more in-depth study of each growth regime was completed by Lassiter et al.[2] for structures grown via EBIE with an expansion to include the desorption time of the electron-precursor reaction product. Fowlkes et al.[64] conversely used the two growth regimes RRL & MTL to isolate specific precursor parameters and through simulation determine their value.

Methods to determine precursor parameters has been the focus of many literature articles[1, 4, 15, 20, 52, 64–67], for example, precursor adsorption energy, diffusion coefficient, and electron dissociation cross section. The analysis undertaken by the research groups followed the same general method of isolating each parameter against various physical quantities such as temperature[15, 52, 67], and electron beam current[64, 67]. The research by Fowlkes et al.[64] is particularly interesting as their results showed under particular conditions the determined surface diffusion coefficients were substrate independent. This result is uncharacteristic of conventional thinking (of the author) where the diffusion coefficient is expected to be precursor-substrate dependent.

In a series of articles by Toth et al.[3, 68] and Lobo et al.[69], a Continuum model that included two precursors was proposed where one caused deposition and the other etching of material. The researchers found that the efficiency at which each precursor molecule decomposed relative to the number of incoming electrons lead to either EBID or EBIE becoming dominant. This switching behaviour between deposition and etching is shown in Figure 2.6 replicated from Lobo et al.[69] where as the electron flux increases the transition from deposition to etching at the electron beam axis is observed. The researchers showed

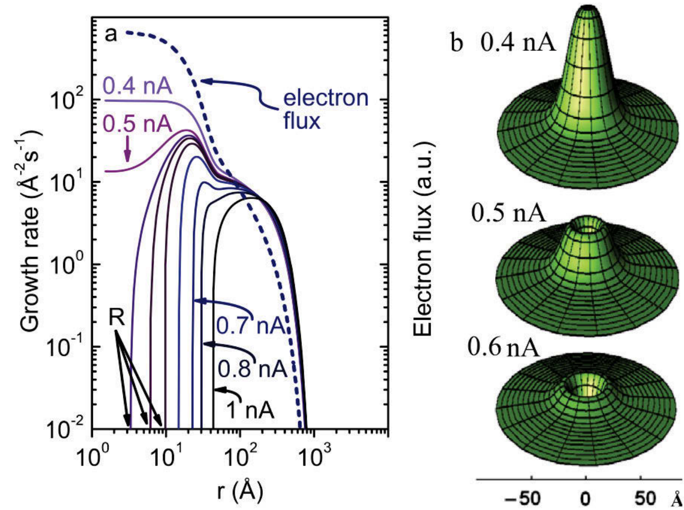

FIGURE 2.6: (Figure Replicated from Lobo, et al. [69]) (a) Growth rate plotted as a function of radius (r) from the beam axis, calculated using currents of 0.4, 0.5, 0.52, 0.53, 0.55, 0.6, 0.7, 0.8 and 1 nA. Also shown is the total electron flux profile from figure 1(d). (b) Surface plots of the 0.4, 0.5 and 0.6 nA deposition rate profiles (linear scale) ($P_d = 10^{-2}$ Pa).

in 2009[68] that this simultaneous deposition and etching can be used to reduce carbon contamination.

Typically the growth rates of EBID deposits decrease with increasing temperature due to desorption. Bishop et al.[63] however demonstrated that at elevated temperatures chemisorption can occur resulting in an increased growth rate. The researchers referred to this process as thermally activated chemisorption. To link this process with their experimental results the researchers extended the standard Continuum model to include the direct chemisorption of precursor molecule from the gas phase and the transition of adsorbed molecules between physisorbed and chemisorbed states. The accuracy of this extended model is demonstrated in Figure 2.7, which is replicated from Bishop et al.'s[63] work. The work by Martin et al.[52] is another case where the researchers have extended the standard Continuum model to include other mechanisms. In this case the researchers included terms to account for the multiple chemical pathways and subsequent reaction products for the etching of Silicon using Nitrogen Fluoride.

VanDorp et al.[70] demonstrated the deposition of nano structures using $W(CO)_6$. The researchers found the deposition rate was much lower than expected and attributed it to

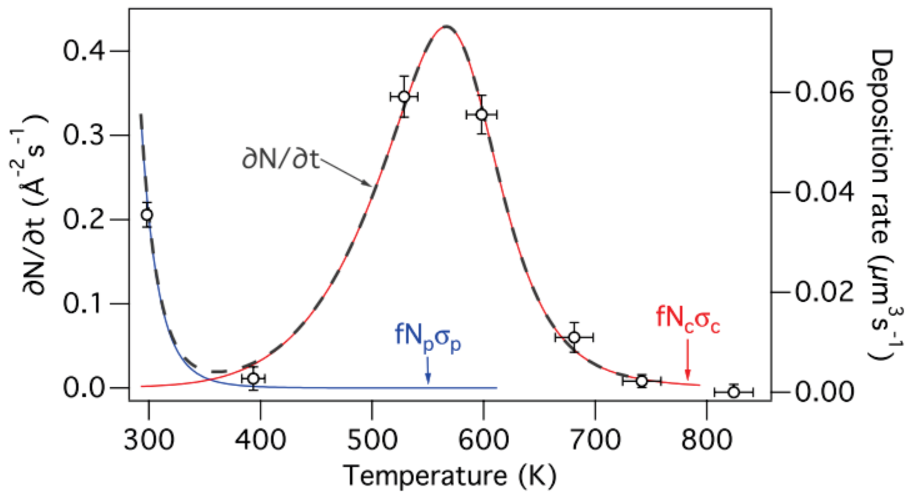

Figure 2.7: (Figure Replicated from Bishop et al.[63]) Steady state adsorbate dissociation rates calculated using Eqs. (4)-(6) as a function of substrate temperature, and the corresponding deposition rates measured using tetraethoxysilane precursor (∘).

electron stimulated desorption. The researchers found this conclusion to be consistent with previous literature[71].

These Continuum models can be used for processes of varying complexity as each model itself is only limited by the mechanisms included. This is demonstrated in many literature articles where the researchers have extended the standard Continuum models to include other physical processes explaining their experimental observations[12, 51, 63, 72, 73]. Nevertheless, a key limitation of the Continuum method is that it cannot incorporate the effects of the evolving sample surface on precursor adsorbates and the electron flux profile responsible for EBIED.

A lack of surface evolution has a significant effect in both the MTL and RRL regimes. In the MTL regime the growth of the surface structure is limited by the arrival of precursor adsorbates, this typically causes structure broadening to occur due to surface diffusion. The electron flux profile is expected to vary greatly with increasing width. The degree of broadening can significantly affect any conclusions made about the key parameter causing lateral growth. In the RRL regime the complex interaction between the structure and the electron beam becomes important, and how the structure grows is now a product of this interaction. Again how the electron flux profile changes with the growing surface becomes very important.

2.3.2 Monte Carlo Models

Monte Carlo models [6–9], involve the explicit simulation of electron interactions with the substrate and time-evolution of the surface. The use of Monte Carlo models in the literature has only featured in the past decade with many authors using it to complement previous or current experimental results. The Monte Carlo models featured in the literature articles discussed below can be grouped into those with mass transport of precursor molecules[6, 74, 75] or those without[7, 9, 76]. In a model without mass transport the growth rate of a deposit (Equation 2.8) is only defined in the reaction-rate limited regime.

$$R(x) = \int_0^{E_0} f(x, E)\sigma_{diss}(E)NdE[76] \tag{2.8}$$

where $R(x)$ is the integral of the number of adsorbed gas molecules (N), the electron flux profile and energy ($f(x, E)$), and the dissociation cross-section of the precursor molecule ($\sigma_{diss}(E)$).

In comparison to a model with mass transport which includes equations to define arrival of precursor molecules from the gas phase (Equation 2.9) and surface diffusion (Equation 2.10).

$$\frac{\Gamma_{gas}}{N_A} = \frac{P}{\sqrt{2\pi MRT}}[75] \tag{2.9}$$

where $\frac{\Gamma_{gas}}{N_A}$ is the precursor flux, N_A is Avogadros number, M is the molecular weight of the precursor, R is the universal gas constant, T is the temperature of the gas, and P is the gas pressure.

$$jumps = \frac{\sqrt{4D_{surf}\tau_e}}{\Delta x^2}[75] \tag{2.10}$$

where $jumps$ is the number of jumps the precursor makes to nearby surface sites per electron, D_{surf} is the surface diffusion coefficient, τ_e is the elapsed time between electrons, and Δx is the voxel separation distance.

Silvis-Cividjian et al.[76] in 2002 used their Monte Carlo model to study why the lateral width of EBID structures is larger than the electron beam diameter. They found the increase in lateral width was due to the high number of secondary electrons leaving through the side walls, which is in agreement with literature discussed above[62]. Perhaps a more

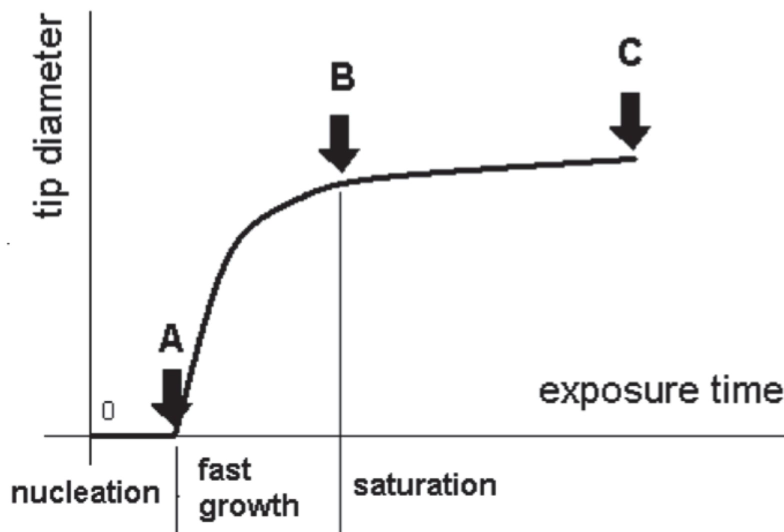

FIGURE 2.8: (Figure Replicated from Silvis-Cividjian et al.[76]) A typical curve showing the evolution of the cone diameter.

unique result from their article was demonstrating the three growth phases every EBID deposit goes through, see Figure 2.8. Later in 2005, Silvis-Cividjian et al.[9] reviewed their previously published articles to provide insights into the most probable deposit lateral width based on the electron beam diameter, the gasous precursor, and the substrate material.

Conversely, in the same year Fowlkes et al.[7] made the statement that EBID growth is not dominated by secondary electrons alone but also by primary electrons, provided growth is in RRL regime where vertical growth is already the primary direction.

Smith et al.[6, 74, 75] developed a comprehensive EBID Monte Carlo model, which included the spatial and temporal coordinates of deposited atoms in addition to the type of electron that induced its deposition. Their first article in 2007 investigated the effect of electron beam energy, mass transport versus reaction-rate-limited growth, and the effects of surface diffusion on the EBID process. One of the key results was in their MTL vs RRL growth study which for the first time demonstrated just how different the lateral width is between the two regimes, see Figure 2.9, and the contribution to growth under the MTL regime from secondary electrons is indeed much greater at 34% compared to 22% under RRL. These results in conjunction with those in a follow up article[6] allowed Smith et al. to come to the conclusion that vertical growth is due to primary and SE(I) electrons, lateral

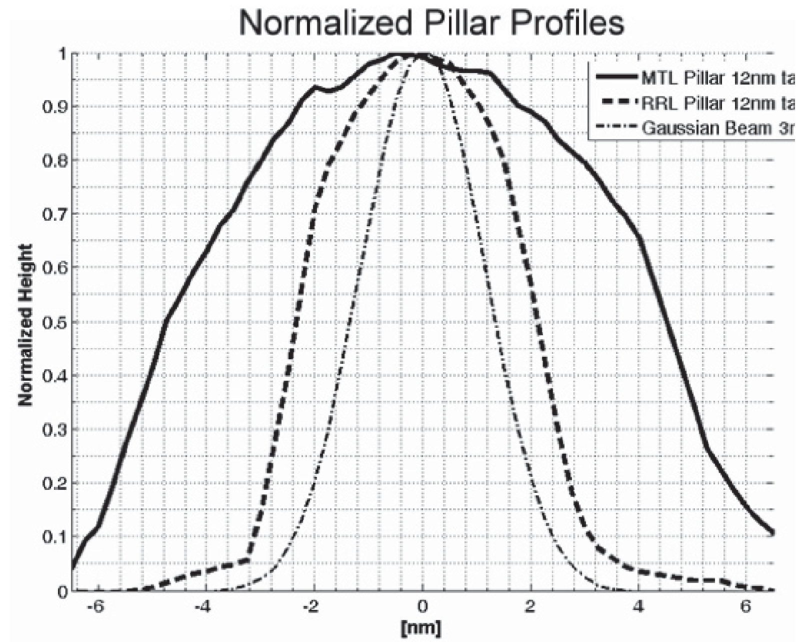

FIGURE 2.9: (Figure Replicated from Smith et al.[74]) Normalized comparison of the MTL pillar shape, RRL pillar shape, and Gaussian beam profiles. These profiles were taken from the 1 keV MTL and RRL pillars at the same height (12.5 nm). 100 000 random Gaussian samples of a 3 nm diameter beam superimposed on the substrate surface to show the beam profile.

growth is due to SE(II) and forward scattered electrons, the width of the nano-pillar is correlated with the size of the electron beam interaction volume and the particular growth regime (MTL or RRL).

Their final paper[75] in the series demonstrated how the particular growth regime can switch from RRL to MTL and visa versa through the magnitude of the precursor diffusion coefficient. This transition was shown in Figure 2.10 where the diffusion coefficient was slowly decreased and eventually set to zero. Further results showed the formation of ring-like structures that occur when the diffusion coefficient is much smaller than the electron beam current; these results correlate with those discussed earlier.

The research by Mitsuishi et al.[77] and Liu et al.[8, 78] features a dynamic Monte Carlo model where they specifically simulate the scattering of electrons through the substrate/deposit material and when an electron crosses the interface to vacuum deposition occurs based on the dissociation cross section. The work presented focused on the effect of electron beam energy and later probe size, substrate thickness, and substrate/deposit material

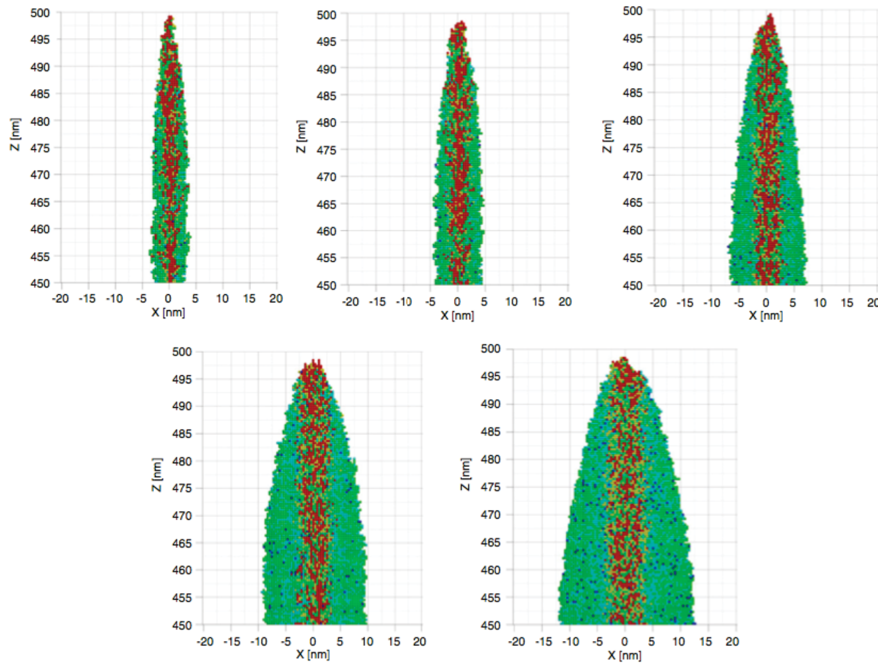

FIGURE 2.10: (Figure Replicated from Smith et al.[75]) Cross sections through the top 50 nm of the pillars at varying diffusion coefficients. Top row, left to right: 1.0×10^{-8}, 1.0×10^{-9}, and 1.0×10^{-10} cm^2s^{-1}. Bottom row, left to right: 1.0×10^{-11}, and 0.0 cm^2s^{-1}.

have on the growth of EBID deposits. Their relatively simple model (in comparison to the one above) was able to directly show the conditions required to form very high resolution EBID deposits.

The computation time needed to simulate the adsorbed molecule transport over the space and time scales typically encountered in experiments is expected to be quite large. This is the limiting factor with using Monte Carlo models for EBID as it is not possible to extend Monte-Carlo methods to deposit sizes typically encountered in experiments. This limitation is prominent when the electron beam interaction volume is greater than the simulation area, and the contribution of precursor adsorbates that could have interacted with secondary electrons no longer occurs.

2.4 Summary

EBIED enables high resolution, direct write deposition though chemical reactions driven by electron-dissociated precursor molecules. The EBIED technique originated as an unwanted side effect of electron beam imaging which has grown to become widely used and researched. To date the simulation of EBIED is achieved with two key modelling methods, Continuum and Monte Carlo models.

Continuum models solve differential equations that describe the rates of change of adsorbate concentrations on the substrate surface as a function of time and space. A key limitation however is that they ignore the effects of the evolving sample surface on precursor adsorbates and the electron flux profile responsible for EBID. Monte Carlo models involve the explicit simulation of electron interactions with the substrate and time-evolution of the surface, but has to date been limited by the computation times needed to simulate the length and time scales typically encountered in experiments.

It is through the combination of these two modelling techniques in our hybrid model that their respective downfalls are overcome, enabling us to investigate structures that occur only at longer times, whilst maintaining the greater experimental accuracy provided by the electron-substrate interactions.

From the above literature we can see that there is a large number of EBIED specific parameters available to the user and each has a similarly diverse effect on the nano structure formed. A number of authors have characterised the relationships between these parameters, however those who have investigated methods to determine the parameter values themselves is small in number. It is here where the work contained within this thesis adds to the literature with methods of how to determine the activation barrier and pre-expontiental factors of desorption and diffusion.

Chapter 3

Hybrid Continuum-Monte Carlo Model

The Hybrid Continuum-Monte Carlo Model calculates precursor concentrations and deposition (or etch) rates using a continuum rate equation approach, and the time-evolution of the electron flux profile by Monte Carlo simulations of electron-solid interactions. Iterative application of these two methods enables efficient simulation of changes in surface geometry with time, as well as the effects of surface evolution on the electron flux and adsorbate concentration profiles.

The Hybrid Continuum-Monte Carlo Model development capitalised on the strong points of each individual method in order to overcome respective pitfalls. These pitfalls were discussed previously but are highlighted with examples from the literature in the subsequent paragraphs. A hybrid model between Continuum and Monte Carlo methods has been previously reported by Rykaczewski et al.[79], however, it was only demonstrated at small simulation times (2.1 sec). The simulation time of 2.1 seconds was used by Rykaczewski et al.[79] as under the three tests cases studied, 'reaction-limited', 'mixed (reaction-diffusion), and 'diffusion-limited' that was the minimum time required to reach steady state growth with the reaction-limited case.

The Hybrid Continuum-Monte Carlo Model developed in this research has a number of advantages over the model developed by Rykaczewski et al.[79]. That model assumes the

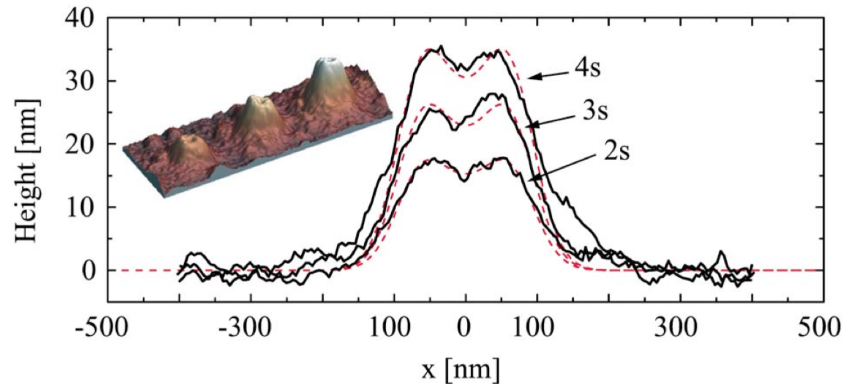

FIGURE 3.1: (Figure Replicated from Utke, et al. [4]) AFM image and line scans of FEB deposits from $Cu(hfac)_2$ precursor. Exposure times are indicated. Indented apex shapes are due to depletion. Dashed lines represent gaussian fits of each deposit.

arrival rate and desorption rate of adsorbed precursor molecules has already reached equilibrium and therefore the related terms are not included. This requires the user to input the initial concentration profile of adsorbed molecules. The robustness of that model to handle complex surface structures observed in diffusion-limited cases[75] is not demonstrated. This may be due to the short simulation time.

Two of the major pitfalls in recent Continuum models is the lack of electron flux and surface evolution. Utke, et al. [4] discussed how the balance between precursor depletion and replenishment determines the resolution inside the local electron irradiated area. With the shape of the grown structures ranging from gaussian (under the RRL regime) to becoming more broad as the amount of molecular replenishment increases due to diffusion (under the MTL regime). While these observations are correct in isolation, when compared to experimental data in Figure 3.1, the pitfalls of Continuum modelling are highlighted. In the AFM line scan at 4s the pillar has begun to broaden at the base, this broadening will continue as the structure grows creating a complex structure morphology that a Continuum model cannot account for. Without the ability to account for this any conclusions drawn about structure resolution and what affects it are questionable.

A major flaw in Monte Carlo modelling is the large computational times required, this limitation typically results in a large restriction on the simulation area. Smith, et al. [74] discussed the growth of high aspect ratio pillars under various conditions: varying beam

energy; mass transport versus reaction-rate-limited growth; and the effects of surface diffusion. The major flaw of Monte Carlo modelling is most prominent when the researchers studied the effect of surface diffusion.

The research by Smith, et al. [74] showed the effect diffusion has on pillar growth quite well, however, the simulation area is significantly smaller than the electron beam interaction volume (simulated with CASINO [80]). The size of the electron beam interaction volume compared to the simulation area is shown in Figure 3.2. The extent of backscatter and secondary electrons leaving the surface is roughly three times that of the simulation area and would result in the adsorbed precursor concentration to be much lower. This would not significantly impact any general conclusion made. For absolute accuracy the simulation area needs to be larger than the electron beam interaction volume and the precursor diffusion length (in MTL regime).

By combining these two modelling methods together this author is able to simulate the growth of nano-structures over the time and space scales of experiments and also provide insights into the physical processes that occur from initial nucleation to steady state growth. This is achieved using a hybrid of Continuum modelling for the growth of the nano-structures and Monte-Carlo modelling for the scattering of electrons through the evolving structure. How each of these modelling methods is combined into a hybrid model and development of further extensions is discussed in following sections.

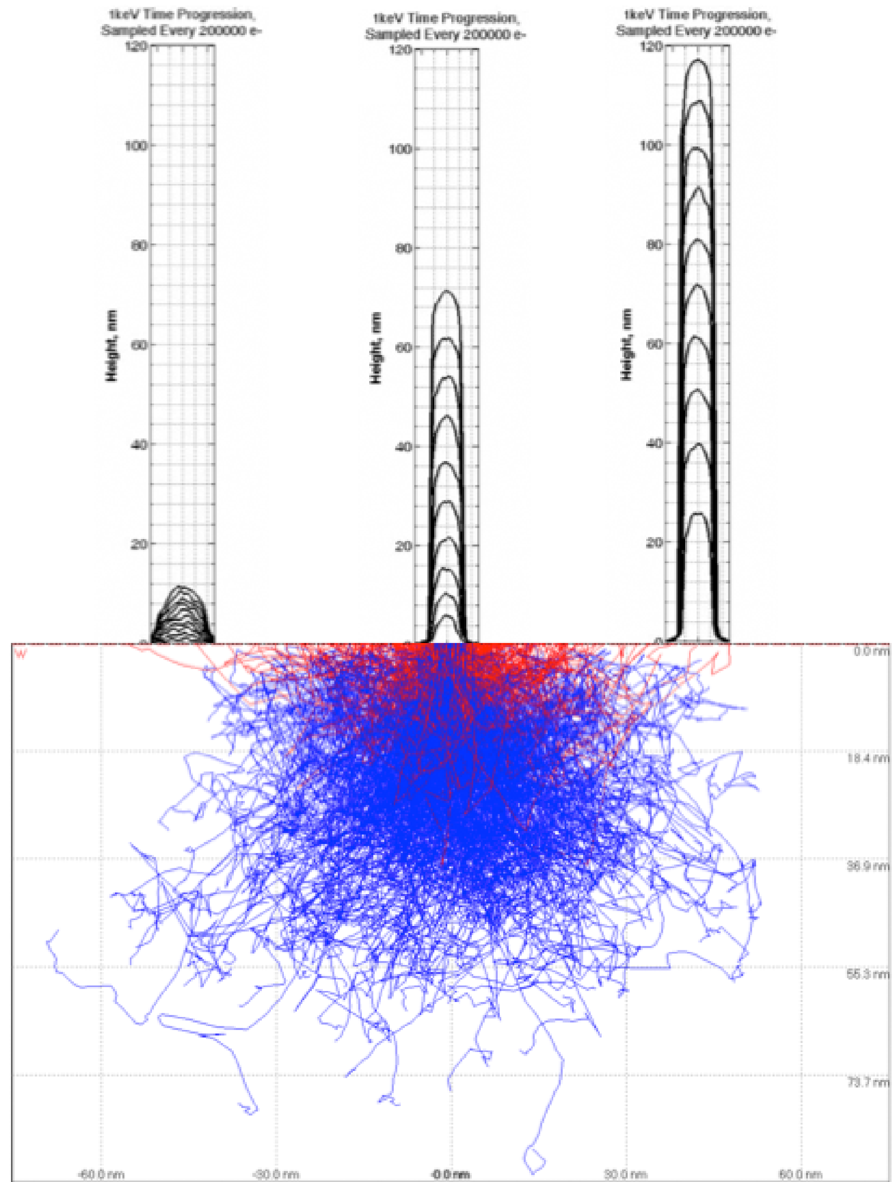

FIGURE 3.2: (Figure Adapted from Smith, et al. [74] with a CASINO Monte Carlo simulation of electron trajectories overlayed.) 2D time-resolved cross-sectional profiles and deposition events (based on electron type) from the gas dynamics simulations. (a) Initial monolayer (ML) present (run 6). (b) Surface diffusion (run 7). (c) Surface diffusion + boundary source (run 8). The normalized sample size is 100 000 electrons.

3.1 Continuum Equations

The Continuum models used in this research (EBID/EBIE/EBIED) were published by Toth et al.[3, 68] and Lobo et al.[69]. The EBID model is shown in Equation 3.1 and

includes the four main components of the EBID process: adsorption; desorption; diffusion; and electron induced decomposition. The EBID model assumes a cylindrical coordinate system where the length scale is the radial distance from the electron beam axis, r. The equation for EBIE is exactly the same with the subscripts changed. It is also similar for the EBIED model except with greater complexity due the addition of another precursor gas. Each of these equations are discretized using the Crank-Nicholson Finite Difference Method [81]; the derivations can be found in Appendix C.

$$\frac{\partial N_d(r,t)}{\partial t} = s_d F_d[1 - N_d(r,t)A_d] - \frac{N_d(r,t)}{\tau_d} + D_d\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right] - \sigma_d f(r) N_d(r,t) \tag{3.1}$$

The EBID model consists of the four processes: adsorption ($s_d F_d[1 - N_d(r,t)A_d]$); desorption ($\frac{N_d(r,t)}{\tau_d}$); electron induced dissociation ($\sigma_d f(r) N_d(r,t)$); and diffusion ($D_d\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right]$), where t is time, N_d is the concentration of adsorbed precursor molecules, s_d is the sticking coefficient, F_d is the precursor flux from the gas phase, A_d is the surface area, τ_d is the desorption(residence) time, D_d is the diffusion coefficent, σ_d is the net electron cross section, and $f(r)$ is the electron flux profile.

The net electron cross section used in the EBID model (including the EBIE and EBIED models) is assumed to be energy independent. This assumption was made due to the shear complexity of recording each individual backscattered and secondary electron's energy as it crossed the surface interface. This unfortunately has the implication that when extracting precursorsolid parameters from EBID/EBIE experiments the values determined are intermixed with the net electron cross section and therefore are energy dependent. This is highlighted in Section 5.2.

From the standard EBID model, the adsorbed precursor concentration can be used to calculate the growing deposit,

$$R = V\sigma f N_a \tag{3.2}$$

where V is the volume of a single adsorbate dissociation product deposited on the surface, and R is the vertical growth rate of the deposit.

The Crank-Nicholson Finite Difference Method is an implicit numerical method[82] that calculates the next time step by taking the central difference in time and the second-order

central difference in space, see Figure 3.3A. As this method requires knowledge of both the previous and next time steps the solution to a tridiagonal matrix of linear equations is required[83]. An explicit numerical method, however, uses the forward difference in time and the second-order central difference in space to directly calculate the next time step[83], see Figure 3.3C.

The major advantage of the Crank-Nicholson over other methods is that it is unconditionally stable[82]; as proven in Appendix C via a Von Neumann stability analysis[84]. Compared to an explicit numerical method that is only stable for specific combinations of spatial and temporal steps[83]. The stability of a numerical method refers to its ability to dampen errors that occur in each iteration (stable) or allow those errors to build (unstable)[83].

To simulate the change in precursor concentration and subsequent surface growth the author re-arranged the EBID equation using the Crank-Nicholson method into a tridiagonal matrix of the form similar to that shown in Equation 3.3, which is then solved via the Thomas algorithm[85]. An excellent walkthrough of this algorithm was written by W. T. Lee[86], where he describes step by step how the algorithm solves the tridiagonal matrix.

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_1 & b_2 & c_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & a_{i-2} & b_{i-1} & c_{i-1} \\ 0 & \cdots & 0 & a_{i-1} & b_i \end{bmatrix} \cdot \begin{bmatrix} x_{n+1} \end{bmatrix} = \begin{bmatrix} e_1 & f_1 & 0 & \cdots & 0 \\ d_1 & e_2 & f_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & d_{i-2} & e_{i-1} & f_{i-1} \\ 0 & \cdots & 0 & d_{i-1} & e_i \end{bmatrix} \cdot \begin{bmatrix} x_n \end{bmatrix} \tag{3.3}$$

3.1.1 Discretization of Time and Space

The spatial step used to discretize the grid representing the nano-structure surface and the time step used to simulate how it evolves with time is quite important. Too small wastes computational time and too large introduces inaccuracies. These in-accuracies are discussed by Richard Ghez[87] in context of how a concentration gradient evolves with time, where the modulus term, λ, or dimensionless diffusion coefficient, D', is defined to

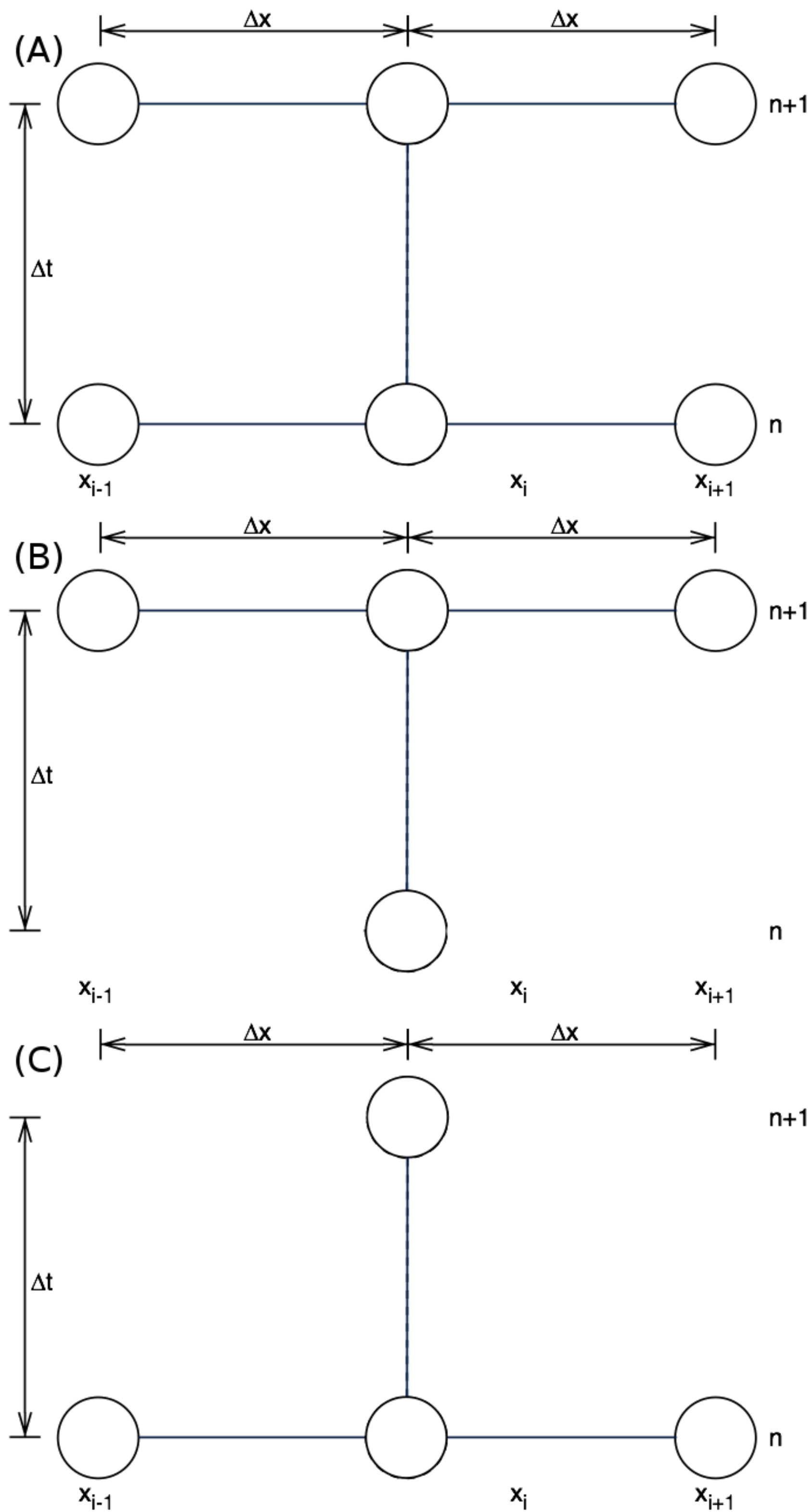

FIGURE 3.3: Representation of three finite difference methods used to solve differential equations in time (n) and space (x): (A) Crank-Nicholson, implicit method. (B) Backward Euler, implicit method. (C) Forward Euler, explicit method.

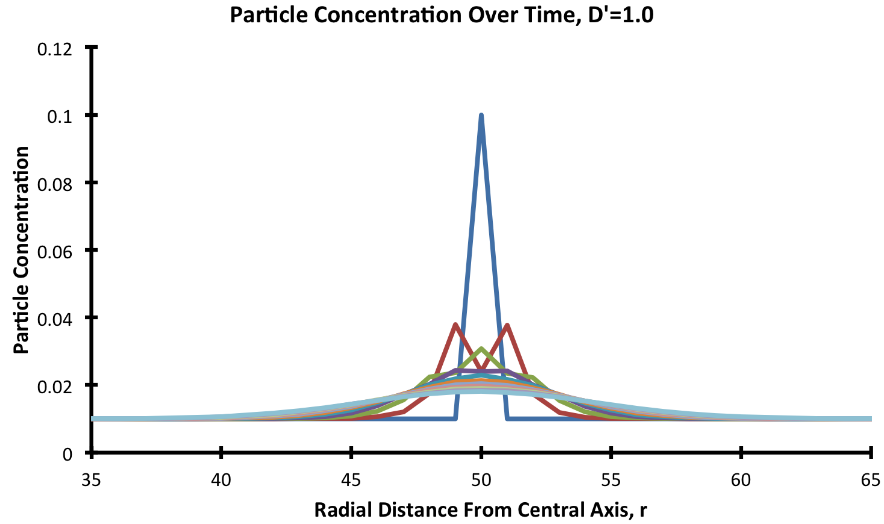

FIGURE 3.4: Example of particle diffusion, where a single spike of concentration spreads over time due to diffusion. Each curve is 10% into the total duration of the simulation except the first curve, which is the starting condition. This example has $D = 1.0 \times 10^8$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 1.0 \times 10^{-8}$ s with a total simulation duration of 1.0×10^{-7} seconds.

represent the ratio between the spatial grid step and time step (Equation 3.4). The correct range for the dimensionless diffusion coefficient is reported to be, $0 < D' < \frac{1}{2}$[87].

$$D' = \frac{D\Delta t}{\Delta x^2} \tag{3.4}$$

To investigate an appropriate range of spatial and temporal steps for EBID/EBIE/EBIED simulations the diffusion of a single spike of precursor concentration is observed over time, where all other terms are either removed or set to either zero or infinity to isolate the diffusion process.

In the first simulation the following input parameters were used, $D = 1.0 \times 10^8$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 1.0 \times 10^{-8}$ s resulting in a dimensionless diffusion coefficient of 1.0 (outside the correct range). The precursor concentration is shown to evolve over time in Figure 3.4 and a problem in how the diffusion initially spread is immediately evident. In the first time step (red curve) the particle concentration is dipped below the adjacent points. This is not the correct behaviour.

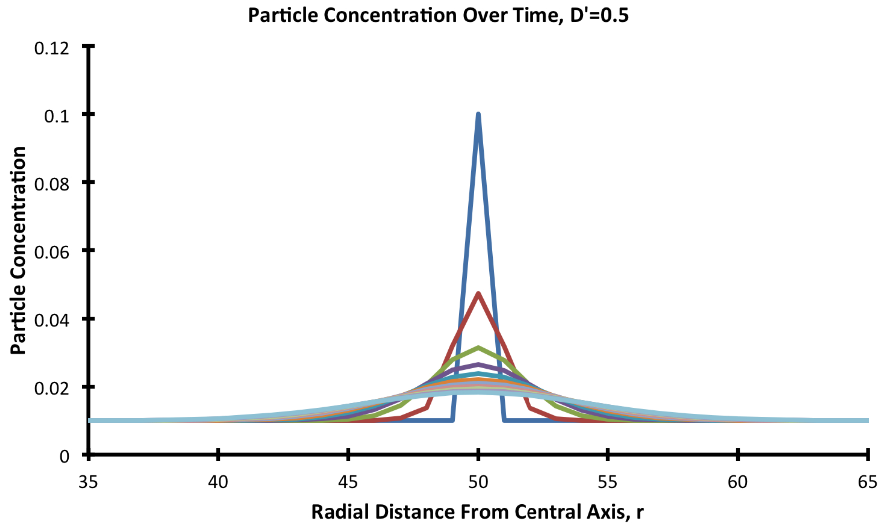

FIGURE 3.5: Example of particle diffusion, where a single spike of concentration spreads over time due to diffusion. Each curve is 10% into the total duration of the simulation except the first curve, which is the starting condition. This example has $D = 1.0 \times 10^{8}$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 5.0 \times 10^{-9}$ s with a total simulation duration of 1.0×10^{-7} seconds.

Two more simulations are shown in Figures 3.5 and 3.6, where D' has been set to 0.5 (so called marginal stability [87]) and 0.1 respectively. The problem initially found in Figure 3.4 is no longer evident even in the marginally stable case following refinement by the author.

Figure 3.7 depicts the combination of spatial and temporal steps needed to ensure a dimensionless diffusion coefficient below 0.5 (green). It is clear that very small time steps are required to ensure nano-scale spatial steps. The author has observed, however, in the hybrid model simulator when the time step is increased by 10% with each iteration that as long as the initial time step is within the green region no errors occur.

3.1.2 Verification

To verify the EBID/EBIE and EBIED Continuum equations two tests are devised to evaluate each key process against known behaviour, and analytical equations. The first test begins with a simulation with zero initial coverage and zero electron beam current to confirm that the maximum coverage is equal to that expected by Equation 3.5 for EBID/EBIE

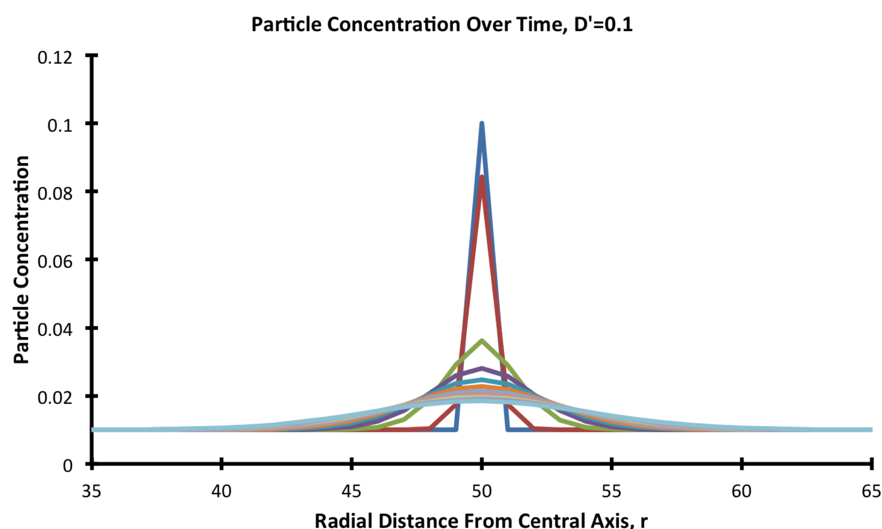

FIGURE 3.6: Example of particle diffusion, where a single spike of concentration spreads over time due to diffusion. Each curve is 10% into the total duration of the simulation except the first curve, which is the starting condition. This example has $D = 1.0 \times 10^8$ Å/s, $\Delta x = 1.0$ Å, and $\Delta t = 1.0 \times 10^{-9}$ s with a total simulation duration of 1.0×10^{-7} seconds.

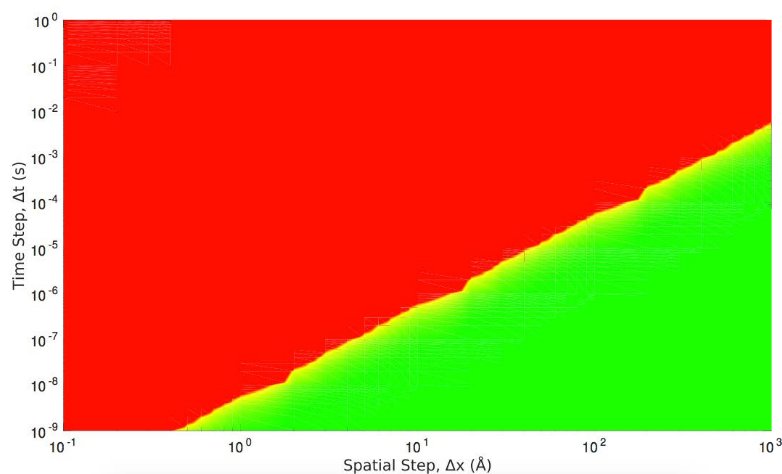

FIGURE 3.7: The dimensionless diffusion coefficient, D', as a function of spatial step, Δx, and time step, Δt, for a diffusion coefficient of 1.0×10^8 Å/s. The green region defines the parameter space where the dimensionless diffusion coefficient is below 0.5 and the red above 0.5.

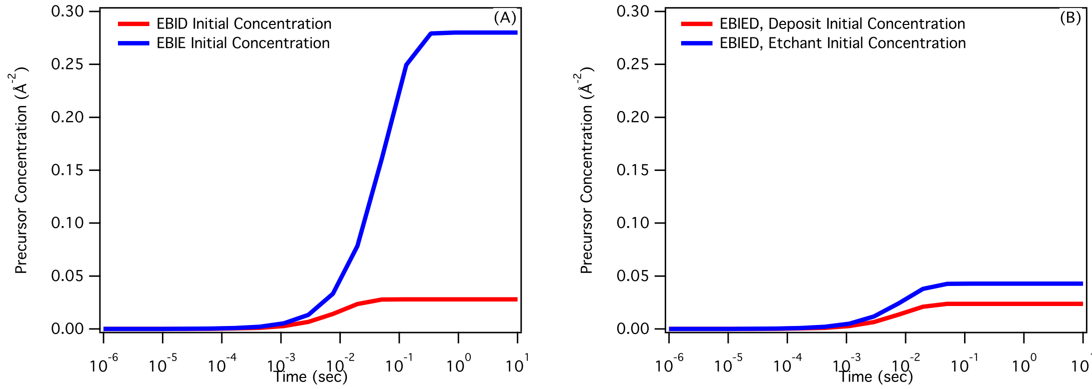

FIGURE 3.8: The initial precursor concentration as a function of time, (A) The concentration from two separate simulations where one is EBID and the other EBIE, (B) The precursor concentration over time from a EBIED simulation.

and known behaviour for EBIED. This test was designed to verify the adsorption and desorption processes. Equation 3.5 calculates the precursor concentration at infinite time (N_∞) given no depletion by the electron beam.

$$N_\infty = \frac{sF}{sFA + \frac{1}{\tau}} \tag{3.5}$$

The initial concentration of precursor molecules from three different simulations each featuring either EBID, EBIE, or EBIED systems is shown in Figure 3.8. The initial concentrations in Figure 3.8A match the calculated values when using Equation 3.5 with: $s = 1$, $F_d = 1.10773$, $F_e = 1.99374$, $A_d = 35.7$, $A_e = 3.57$, $\tau = 6057.97$ (Note: the two different surface areas and the resulting difference in concentration). The initial concentration in Figure 3.8B cannot be compared to an analytical solution due to the competition between surface sites of the two precursors but the values are as expected with the concentration of the smaller etchant molecule limited by the larger deposition molecule.

The second test is based upon the results by Lobo, et al. [69] replicated in Figure 3.9 to verify the growth of a nano structure using the EBIED model. The test case was designed to show the same behaviour where as the electron beam current is varied and the result of the competition between deposition and etching changes.

The results by Lobo, et al. [69] (Figure 3.9), showed that the relative strength of each process, deposition or etching, could be controlled via the magnitude of the electron flux.

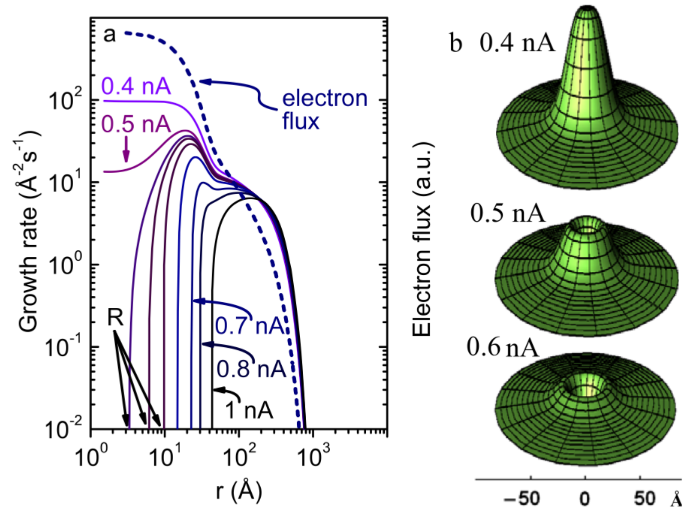

FIGURE 3.9: (Figure Replicated from Lobo, et al. [69]) (a) Growth rate plotted as a function of radius (r) from the beam axis, calculated using currents of 0.4, 0.5, 0.52, 0.53, 0.55, 0.6, 0.7, 0.8 and 1 nA. Also shown is the total electron flux profile from figure 1(d). (b) Surface plots of the 0.4, 0.5 and 0.6 nA deposition rate profiles (linear scale) ($P_d = 10^{-2}$ Pa).

At low electron flux the deposition process, being more efficient, causes deposit growth to occur. At high electron flux, deposition molecules become depleted causing the etch process to be dominant, etching the deposit. The Gaussian shape of the electron beam, however, results in the growth of ring-like structures. Due to the high electron flux beam centre and low electron flux beam tail. It is this electron flux controlled switching of each process that will be shown to verify the EBIED model.

To compare the models, the growth rates of a series of deposits are simulated as a function of electron beam current. Figure 3.10 shows that the vertical growth rate calculated at $r \sim 1.5$ $\mathring{A}$ decreases with increasing current. The decrease in growth rate is caused by adsorbate depletion near the beam axis. The excellent agreement between the two models confirms consistent implementation of all terms in our EBIED model.

3.2 Electron Trajectory Simulation

Monte Carlo methods [80, 88] are used to model the electron-solid interactions needed to calculate the backscattered and secondary electron contributions to the total electron flux profile $f(r)$. The electron-solid interactions are modelled using the single scattering and

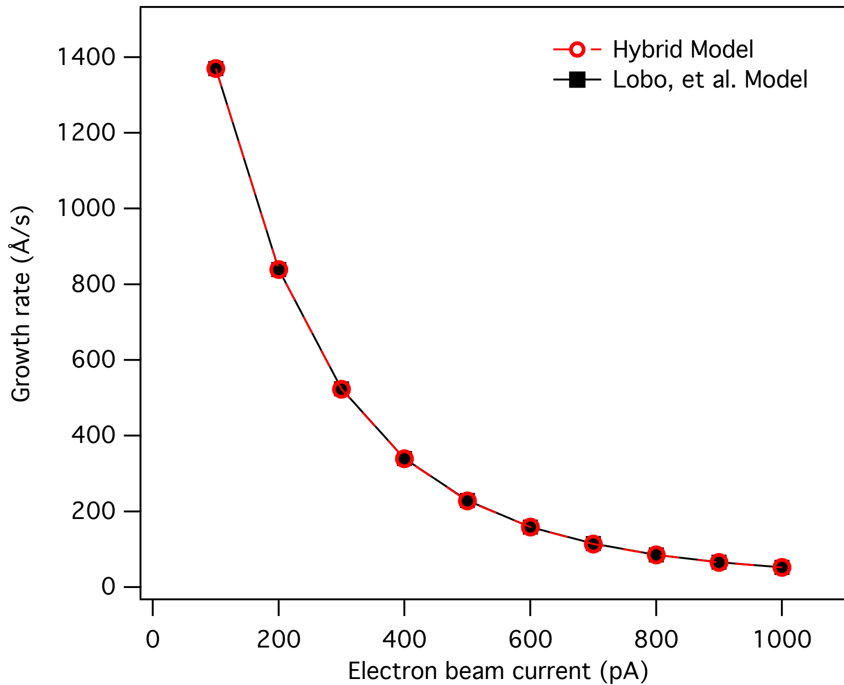

FIGURE 3.10: Comparison between the continuum component of the hybrid model used here and the continuum model published previously by Lobo et al. [69]. The EBID rates were calculated at a distance of ~ 1.5 $\mathring{A}$ from of the electron beam axis as a function of electron beam current. Both models show the same decrease in growth rate with increasing current, caused by adsorbate depletion near the beam axis.

parametric models developed by David Joy[88], with extensions made to work with an evolving surface.

The single scattering model simulates the interaction of electrons by calculating its path taken through the material from one scattering event to another. The scattering events are limited to elastic scattering events only, as the interaction between the electron and the atomic nucleus causes angular deflections of 5°to 180°, compared to inelastic scattering events of about 0.1°[88]. The elastic scattering events are described using Rutherford cross sections, and by ignoring the inelastic scattering events a continuous slowing down approach is used whereby it is assumed that the energy lost in each scattering event is lost evenly over the entire scattering path.

The trajectories of forward and backscattered electrons are calculated using a line-line intersection method[89]. It involves calculating the intersection point of the current electron scattering event and each point on the surface. When only one intersection point lies on

the surface the electron is determined to have backscattered through this point and left the solid. When two intersection points lie on the surface the electron is determined to have forward scattered out of the surface at the first intersection point (e.g. a pillar sidewall), and into the surface at the second intersection point (e.g. a horizontal substrate region adjacent to the pillar).

Secondary electrons are not simulated explicitly but instead calculated using a modified version of the parametric model designed by Joy[88]. The parametric model approximates the secondary electron yield at a single point on a flat surface and enables accurate modelling of secondary electron generation without requiring details of its generation or scattering cascades. In the author's implementation, the parametric model is modified to enable SE generation from curved surfaces. The first modification is to evenly distribute the generated secondary electrons along the entire primary electron path between elastic scattering events. Second, the author assumed isotropic SE generation, and calculated the fraction that intersects each point making up the discretized surface.

The extensions made enable the Monte Carlo methods to work with the evolving surface structure, particularly with the number and correct location of backscattered and secondary electrons, and forward scattering of electrons back into the surface.

3.2.1 Single Scattering Model

The single scattering model describes the scattering events of an electron within a material. This model was developed by David Joy where a complete write up is found in "Monte Carlo Modeling for Electron Microscopy and Microanalysis"[88]. A summary of the model with the assumptions made and the key equations are reported here.

There are two major assumptions made in the Single Scattering model, only elastic scattering events are considered and a continuous slowing down approach is used. Elastic scattering events are those where the electron is attracted to the positive nucleus and, subsequently, repulsed by the surrounding electron cloud. This concept could be described

as a collision between billiard balls. These elastic scattering events result in angular deflections of 5°to 180°compared to inelastic scattering events of about 0.1°. This large difference enables us to ignore the inelastic scattering events

The continuous slowing down approach assumes that the electron is losing energy continuously over its scattering path due the number of inelastic scattering events. Although these inelastic scattering events are not continuous they can be averaged to occur over the entire path.

The key equations in the single scattering model are used to describe the size, direction, and subsequent energy loss of each scattering event. The size of scattering event comes from the calculation of the electron's mean free path. It is a function of the elastic cross section. The elastic cross section, σ_E, is calculated from the total screened Rutherford elastic cross section[90], Equation 3.6.

$$\sigma_E = 5.21 \times 10^{-21} \frac{Z^2}{E^2} \frac{4\pi}{\alpha(1+\alpha)} \left(\frac{E+511}{E+1024} \right)^2 \tag{3.6}$$

Where E is the electron energy, Z is the atomic number of the material, α is an approximation for the electron not observing all of the nuclear charge[91], as defined in Equation 3.7.

$$\alpha = 3.4 \times 10^{-3} \frac{Z^{0.67}}{E} \tag{3.7}$$

From the elastic cross section the author calculates the mean free path, λ, of the electron in Equation 3.8.

$$\lambda = \frac{A}{N_A \rho \sigma_E} \tag{3.8}$$

where N_A is Avogadro's number, ρ is the density of the material, and a is the atomic weight of the material. This mean free path is an average, and therefore, the actual distance, s, that the electron scattered is given in Equation 3.9 including a random variation, RND.

$$s = -\lambda Log_e(RND) \tag{3.9}$$

The angle of the scattering event, ϕ, is also calculated through Equation 3.10.

$$\cos\phi = 1 - \frac{2\alpha RND}{1+\alpha-RND} \tag{3.10}$$

With this angle the azimuthal scattering angle, ψ, can be calculated which defines the possible scattering cone of the electron.

$$\psi = 2\pi RND \tag{3.11}$$

To calculate the new position of the electron from the previous position in the Cartesian coordinate system the following series of equations are used.

$$xn = x + s \times ca \tag{3.12}$$

$$yn = y + s \times cb \tag{3.13}$$

$$zn = z + s \times cc \tag{3.14}$$

These directional angles, ca, cb, and cc are calculated using Equations 3.15 to 3.17

$$ca = cx\cos\phi + V1V3 + cyV2V4 \tag{3.15}$$

$$cb = cy\cos\phi + V4(czV1 - cxV2) \tag{3.16}$$

$$cc = cz\cos\phi + V2V3 - cyV1V4 \tag{3.17}$$

where,

$$V1 = AN\sin\phi \tag{3.18}$$

$$V2 = ANAM\sin\phi \tag{3.19}$$

$$V3 = \cos\psi \tag{3.20}$$

$$V4 = \sin\psi \tag{3.21}$$

$$AN = -\frac{cx}{cz} \tag{3.22}$$

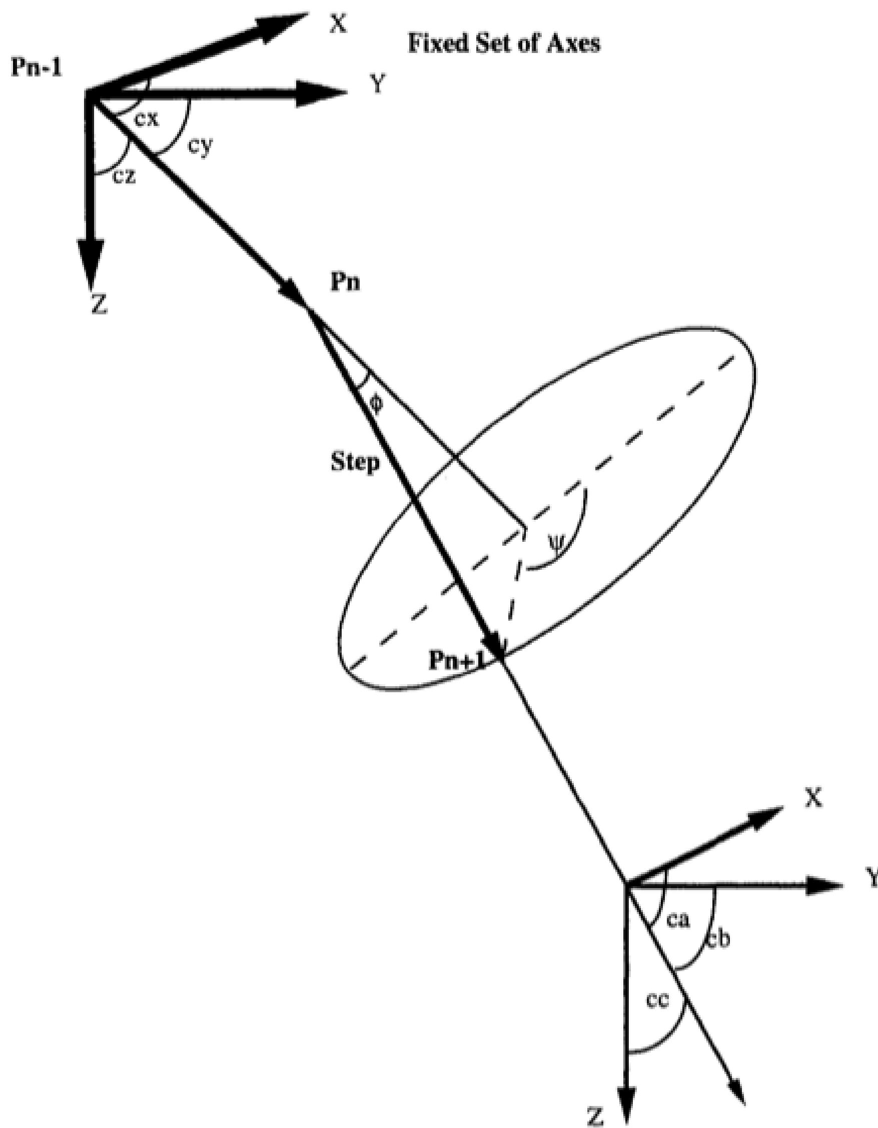

FIGURE 3.11: (Replicated from David Joy, 1995[88]) The scattering path of an electron with the corresponding step length and angle components.

$$AM = \frac{1}{\sqrt{1 + AN^2}} \tag{3.23}$$

These equations calculate the distance and direction the electron travels in each scattering event, the equations are represented in Figure 3.11 replicated from "Monte Carlo Modeling for Electron Microscopy and Microanalysis"[88].

The final equation defines the amount of energy lost by the electron over the scattering event. To calculate this energy loss a modified Bethe equation[92] in Equation 3.24 is used, where the rate of energy loss, dE/dS, is a function of electron energy, material

atomic number and weight, and the mean ionisation potential[93] in Equation 3.25. The mean ionisation potential, J, approximates the energy lost per interaction of the electron with the material.

$$\frac{dE}{dS} = -78500\frac{Z}{AE}\log_e\left(\frac{1.166(E+0.85J)}{J}\right) \tag{3.24}$$

$$J = \left[9.76Z + \frac{58.5}{Z^{0.19}}\right] \times 1 \times 10^{-3} \tag{3.25}$$

3.2.2 Parametric Model

The Parametric Model describes the secondary electron yield at each point along a flat surface and was designed by David Joy[88] to accurately model the experimental behaviour of secondary electrons without the details of its generation or scattering cascade. The parametric model can be spilt up into two key equations, the first describes the generation of the secondary electrons and the second its transport to the surface.

The number of secondary electrons generated from a primary electron scattering event is directly proportional to the stopping power of the electron at that point over the scattering length of that event. This is described in Equation 3.26, where ϵ is the material dependent energy required to generate a secondary electron.

$$N_{SE} = -\frac{1}{\epsilon}\cdot\frac{dE}{ds} \tag{3.26}$$

The transport of these secondary electrons to the surface is assumed to follow the "straight-line approximation", whereby the probability of the secondary electron escaping along a direct path to the surface is based upon its depth and its mean free path. This is described in Equation 3.27, where λ is the material specific mean free path of a secondary electron and A is a constant of 0.5 since only half of the secondary electrons will move towards the surface.

$$p(z) = A\cdot\exp^{-\frac{z}{\lambda}} \tag{3.27}$$

From Equations 3.26 & 3.27, the number of secondary electrons arriving at each point along the surface is found and as a result the desired secondary electron yield.

3.2.3 Model Extensions

3.2.3.1 Backscattered & Forward Scattered Electrons

For a flat substrate an electron is determined to backscatter once it travels above zero. This simple definition cannot be applied to an evolving surface. To determine whether an electron backscatters a new method is defined that evaluates the electron scattering event for leaving the surface or not, and whether the electron re-enters the surface, subsequently.

The first step is to check if the electron has travelled above the minimum z-axis point of the surface. This eliminates any electron that is deep within the surface from needing to complete the next step.

The next step is to calculate the line-line intersection between the scattering event and each surface section. While every scattering event and surface section would intersect at one point in infinite space, a check that the intersection point lies within the bounds of the current section is made. From these conditions it can be determined that there are three possible cases: no intersection; 1 intersection; and 2 intersections. In detail these are:

- No intersection, the electron is still within the surface and scattering can continue
- 1 intersection, the electron is determined to have backscattered
- 2 intersections, the electron is determined to have forward scattered and its scattering event modified to accommodate the re-entry into the surface.

3.2.3.2 Secondary Electrons

The Parametric Model described in section 3.2.2 is designed for flat surfaces only. Where the number of secondary electrons generated from each primary electron scattering event is to leave through the closest surface site. In order to model the secondary electron yield correctly two modifications are made. Conceptual diagrams of the original model and the two modifications are shown in Figures 3.12, 3.13 and 3.14 respectively.

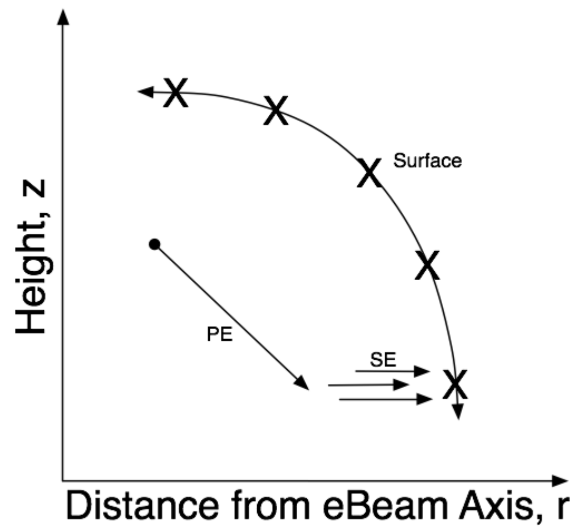

FIGURE 3.12: The original Parametric Model, where all secondary electrons generated are emitted through the closest surface bin.

The first modification involves changing the secondary electron generation at the end point of the primary scattering event to evenly over the entire scattering path. This modification more realistically simulates what happens experimentally. The primary electrons scatter elastically and in-elastically generating secondary electrons. The even distribution of the secondary electrons is achieved by finding the two closest points on the surface at the start and end points of the primary electron scattering event and then dividing the total number of secondary electrons over those points and any in-between, taking into account the change in distance to each point along the surface.

The second modification extends the first by modifying the constant A in Equation 3.27 to match correctly the angular distribution of the secondary electrons path to the surface. The same initial steps are taken to the first modification but the angle of the secondary electrons to the surface normal direction of each bin is taken into account.

With these modifications the Parametric Model is able to calculate the number of secondary electrons at each surface site even with an evolving surface.

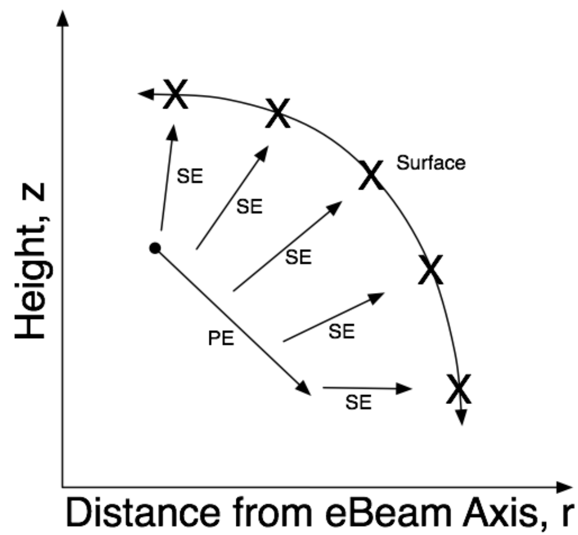

FIGURE 3.13: The first modification to the Parametric Model, where all the secondary electrons generated are distributed evenly over the closest surface bins.

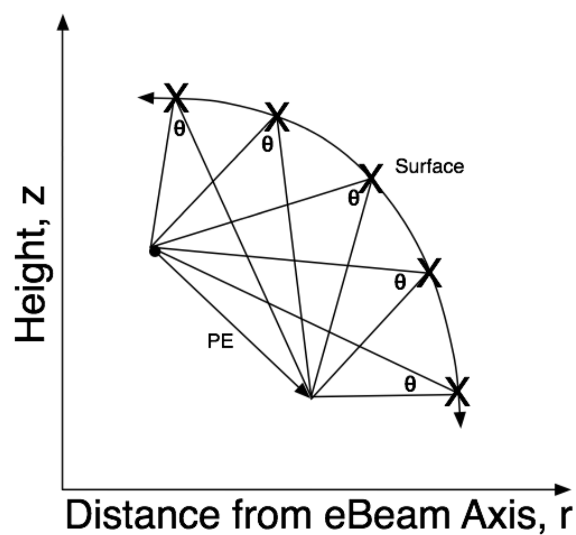

FIGURE 3.14: The second modification to the Parametric Model, where all the secondary electrons generated are distribution evenly over the closest surface bins, however the number in each bin is modified by its solid angle to the primary electron trajectory.

3.2.4 Verification

To verify the Electron Trajectory Simulator a series of test cases were devised and compared with either CASINO[80] or experimental data. These tests included:

- The backscattered electron coefficient, as a function of incident beam energy.
- The secondary electron yield, as a function of incident beam tilt.
- The secondary electron yield, as a function of incident beam energy.

CASINO, a well known Monte Carlo electron trajectory simulator (used to simulate electron interactions) is therefore chosen as an adequate comparison. The experimental data for the secondary electron yield is collected from a series of sources[54]. The simulations are conducted on a silver substrate with an electron beam tilt ranging from 0° to 85°and incident beam energy from 0.03 keV to 30.0 keV using 100 000 electrons.

The first test case compared the backscattered electron coefficient as a function of incident beam energy. The simulation result in Figure 3.15 matched almost perfectly with the Rutherford Cross Section result, as expected.

The second test case compared the secondary electron yield as a function of incident beam energy. The simulation results in Figure 3.16 show a similar trend with a very low yield at low beam energies and increasing to a peak at approximately 1 keV then decreasing. There is a difference at the lower beam energies in comparison to the experimental data, but it is as expected, as the Rutherford cross sections used in the model are known to not work very well at these energies. The remaining data matches quite well, considering the large variation in values gathered from the literature.

The final test case compared the secondary electron yield as a function of electron beam tilt. The results in Figure 3.17 show two distinct trends with increasing tilt, at 1 keV the yield remains largely unchanged until high tilt angles which is expected as the interaction volume is almost completely contained within the secondary electron escape depth. At 5 keV and 10 keV the secondary electron yield increases with larger tilt angles as the

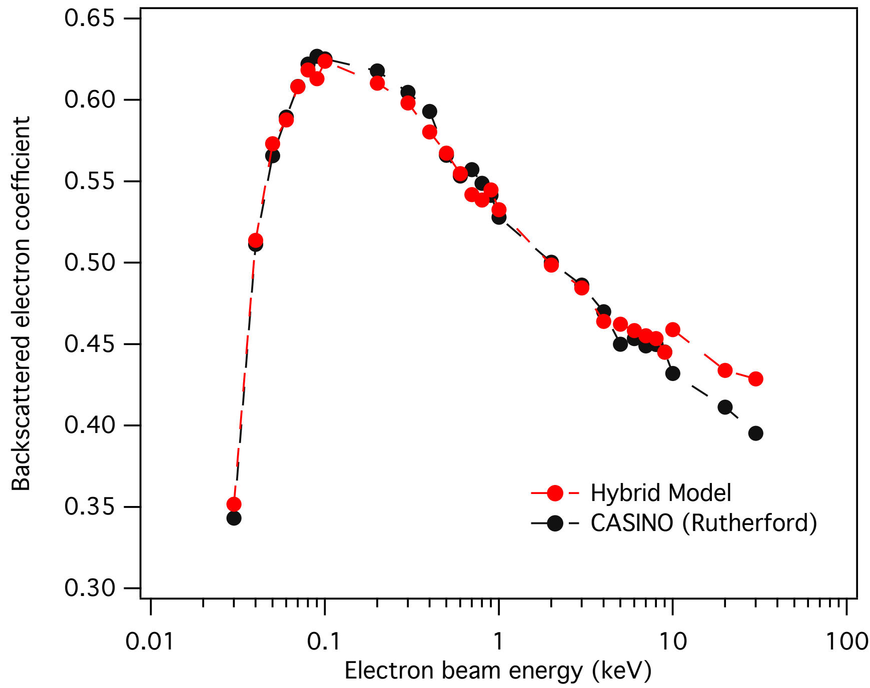

FIGURE 3.15: Comparison between the Monte Carlo component of the hybrid model used here and the Monte Carlo model CASINO [80]. The dependence of the backscattered electron coefficient on electron beam energy calculated the hybrid model is in excellent agreement with that calculated by CASINO.

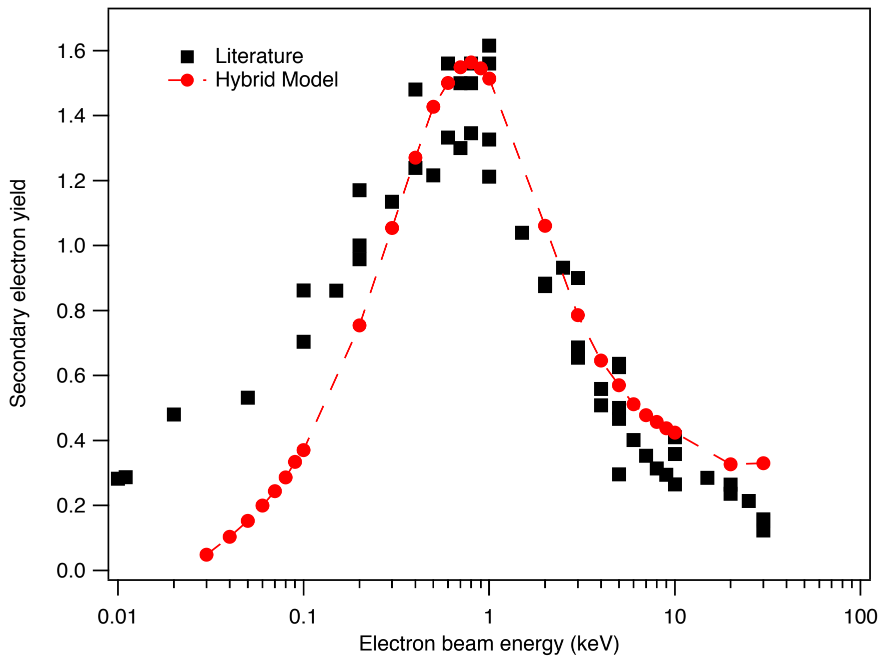

FIGURE 3.16: Comparison between the secondary electron yield on a silver surface as a function of electron beam energy taken from literature data and the hybrid continuum-Monte Carlo model.

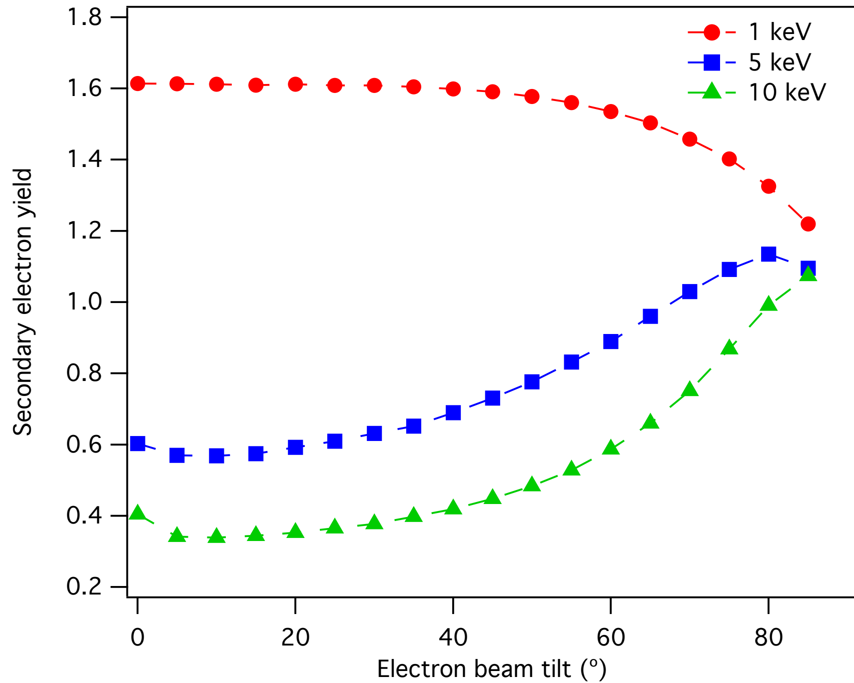

FIGURE 3.17: The secondary electron yield on a silver surface as a function of electron beam tilt. The two distinct trends are consistent with known behaviour of the depth and size of the electron beam interaction volume with beam tilt.

relatively deep interaction volume becomes more contained within the escape depth. Both of these observations are expected from known behaviour.

From these three test cases it is determined that the Electron Trajectory Simulator is working and has shown to replicate data gathered from a well known simulation program and experimental data.

3.3 Model Extensions

The combination of both Continuum and Monte Carlo modeling methods do overcome many of the respective pitfalls. Nevertheless, when compared to an experimental system some pitfalls still exist. Thus, a number of extensions are developed to address these remaining pitfalls.

3.3.1 Surface Evolution & Electron Beam Projection

The first of these pitfalls is how the growing structure evolves over time. In previous Continuum models[3, 68, 69] the growing deposit surface is evolved directly upwards/ downwards. In an experiment the surface would evolve in a surface normal direction as molecules are deposited or removed. The surface normal direction can be calculated from the vector joining the two points adjacent to the point to be moved. Due to statistical noise from the Monte Carlo model, in the Hybrid Model a more complex approach is needed. The calculation of the surface normal direction needs to be done over many points, allowing the surface to grow much smoother and also to prevent sharp spikes or other computationally undesirable effects occurring. The difference between evolving directly upwards and with the surface normal direction is shown in Figure 3.18.

The second pitfall involves how the primary electron beam interact with the evolving surface. For a flat surface all electrons impacting a specific point interact with the surface, but in the case of a vertical wall those same electrons would pass by instead. The surface area seen by the electron beam at these sharp wall-like regions becomes significantly reduced compared to the flat surface. Therefore the amount of primary electron flux is modified to reflect the surface area at each point. This is coined the electron beam projection onto the surface. It is important to note that the backscattered and secondary electron flux profiles are not modified in this manner as they are calculated explicitly by the Monte Carlo algorithm. An example of how electron beam projection effects the primary electron flux profile is shown in Figure 3.19. We observe as the simulation progresses the electron flux profile changes in shape to reflect the evolving surface, which would be similar to that shown in Figure 3.18. The electron flux at approximately 700 nm remains unchanged throughout the simulation due to that area remaining relatively flat.

3.3.2 Surface Diffusion Modelling

As the surface evolves over time, the physical process of surface diffusion becomes increasingly more difficult to solve. For a flat surface, the process of surface diffusion is well defined in the radial coordinate system, with the laplacian operator as defined in Equation

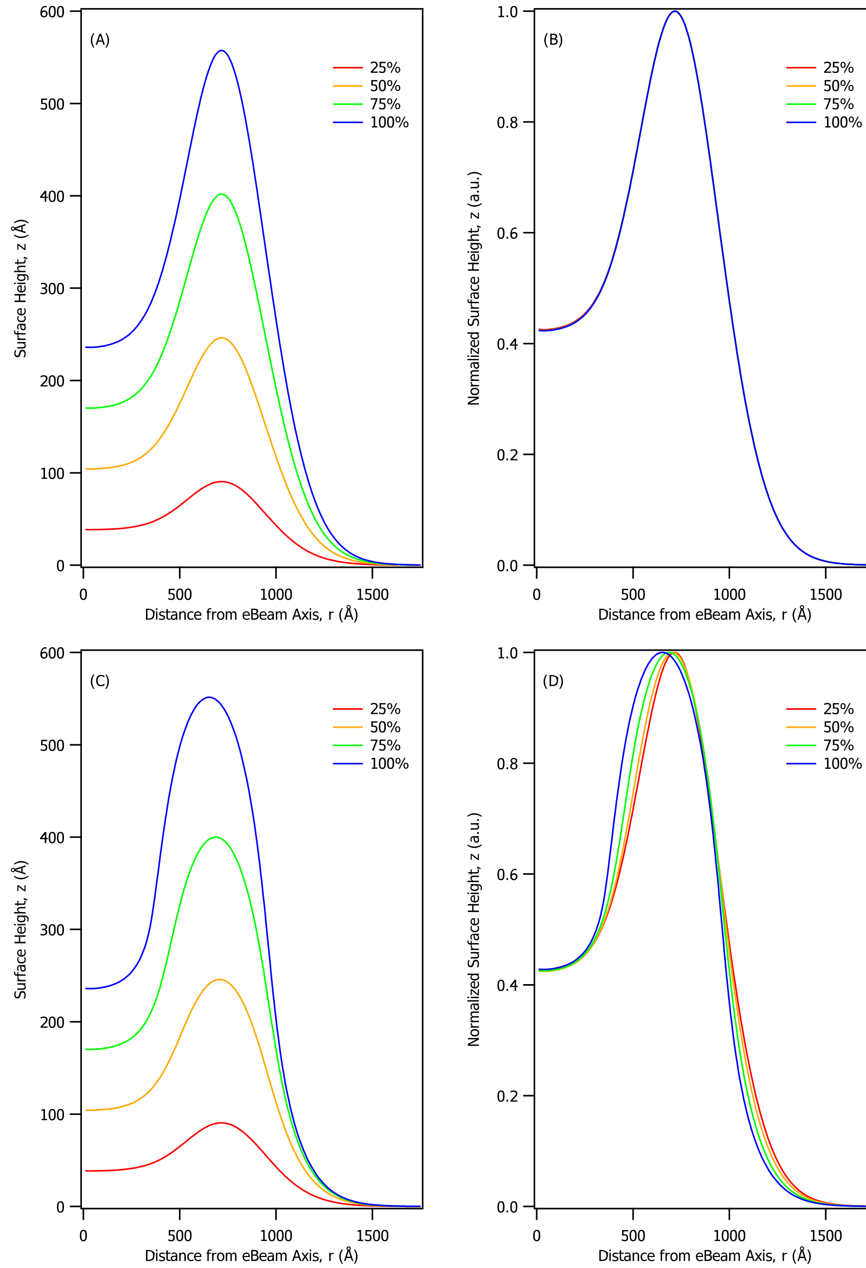

FIGURE 3.18: Two EBID simulations were performed for a maximum duration of 13.7 seconds, each plot is shown at different stages within the entire simulation beginning at 25% through to 100% of the maximum duration. (A) Grown without surface evolution the deposit growth direction is upwards over all space; this is confirmed in (B) where the deposit structures have been normalised. (C) Grown with surface evolution the deposit growth is in the surface normal direction; again confirmed in (D).

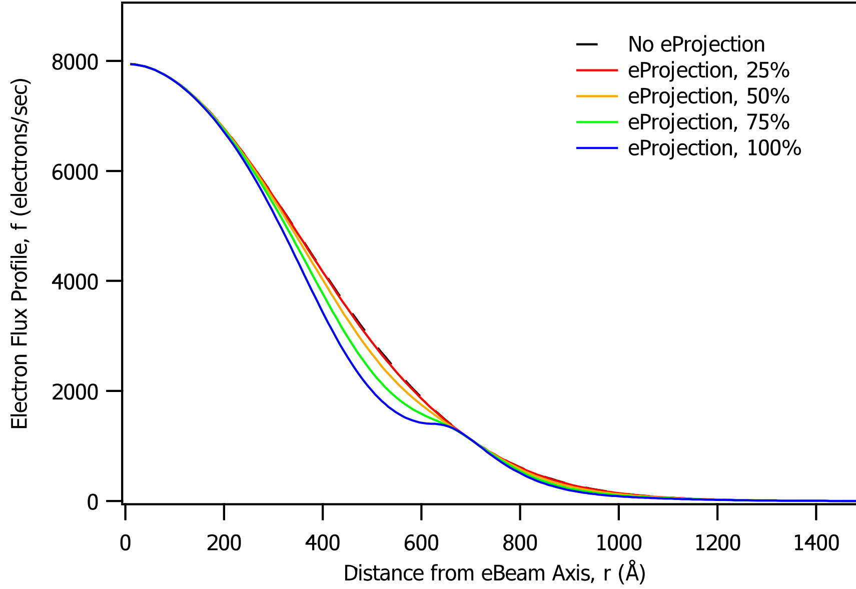

FIGURE 3.19: The electron flux profile (primary electrons only) from an EBID simulation with electron beam projection switched on/off. The simulation maximum duration was 13.7 seconds, and each plot is shown at different stages within the entire simulation beginning at 25% through to 100% of the maximum duration. We observe as the simulation progress the electron flux profile changes in shape to reflect the evolving surface, which would be similar to that shown in Figure 3.18.

3.28. As the surface evolves a question arises, is this equation still valid and if not how does one account for any deviation.

$$\nabla_r^2 \equiv \left(\frac{1}{r} \frac{\partial}{\partial r} + \frac{\partial^2}{\partial r^2} \right) \tag{3.28}$$

$$D \nabla_r^2 N_a \tag{3.29}$$

Equation 3.29 defines the diffusion of molecules in the radial coordinate system by collapsing these annuli into a 1D string of points where the change in area of each annuli as it moves further away from the central axis is taken into account.

As the surface evolves a 2D string of points is formed and how these collapse down into the form required by Equation 3.28 is unclear. Figure 3.20 depicts the translation from the physical surface to the computational surface that needs to be understood and defined.

Two possible solutions are proposed to solve this problem: either the points are moved normal to the surface and the spacing between them varies, so called Free Form Movement;

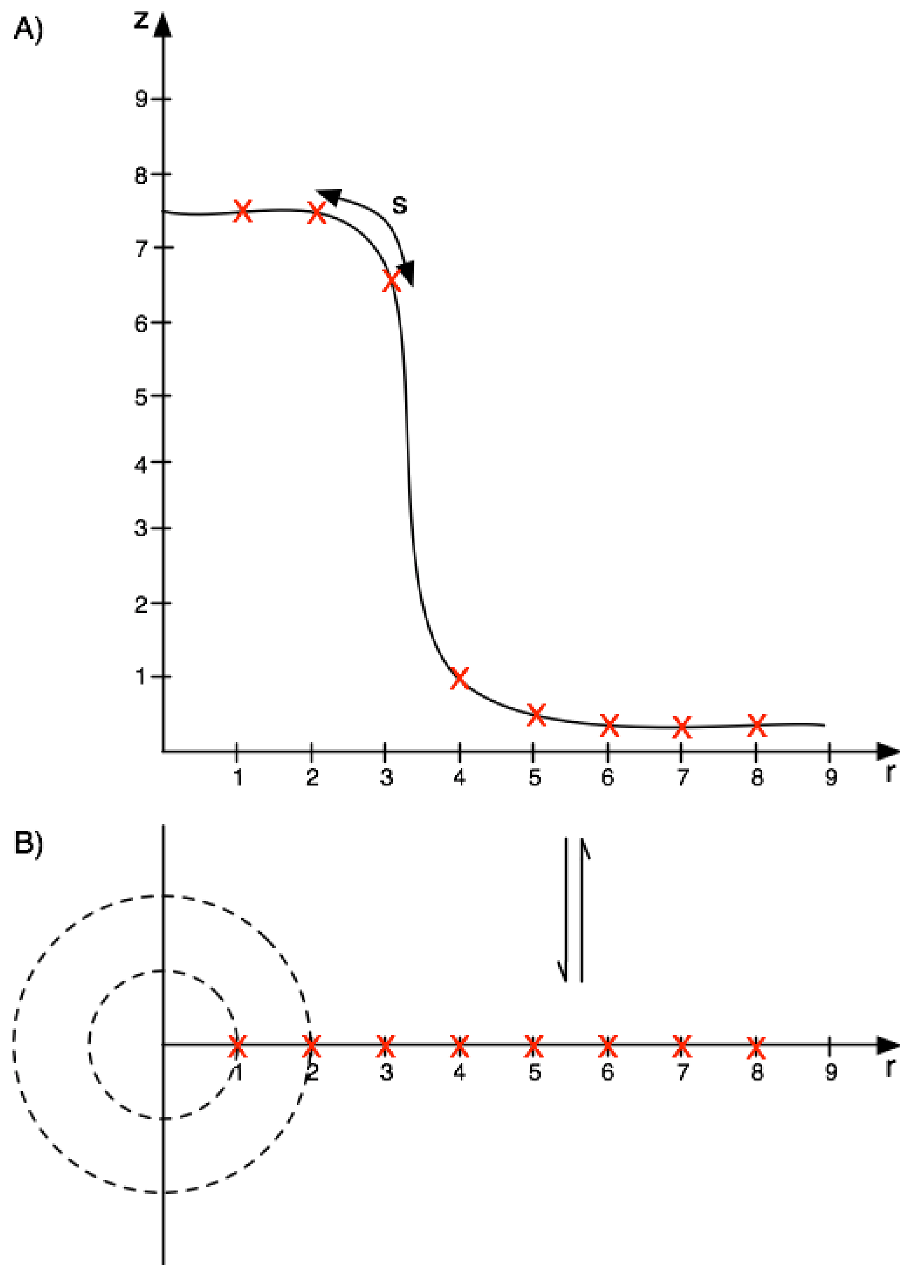

FIGURE 3.20: The translation between the physical surface A) and the computational surface B). The physical surface is two dimensional in r & z and the computational surface is one dimensional in r, where both surfaces are radially symmetric. The path length between the points along the physical surface is, s.

resulting in a non-uniform 1D grid, or the points are moved normal to the surface but then remapped in such a way that the area of each annulus remains equal to that of a flat surface, a uniform 1D grid. The first solution is simpler in that each point is allowed to move, however, the method to deal with the non-uniform grid spacings is untested (see Section 3.3.2.1). The second solution is computationally more expensive due the remapping of the points but the method (see Section 3.3.2.2) for uniform grid spacings is well defined and understood. Through extensive testing it is determined that area remapping provided the most reliable method to dealing with surface diffusion on an evolving surface.

3.3.2.1 Free Form Movement

The process of solving the partial differential equations derived in Appendix C for free form movement, involves many of the ideas presented by Sobey [94], where the transient diffusion equation in 1D is derived for a non-uniform Cartesian grid. The key idea is that the asymmetric grid spacing between data points can be accounted for by including an asymmetry parameter, A_i, in the second spatial derivative of $\frac{\partial^2 C}{\partial x^2}$, where A_i is defined as $(x_i - x_{i-1} - \Delta x)/\Delta x$, Δx as $(x_{i+1} - x_{i-1})/2$, and C is a scalar quantity representing a molecular concentration. A schematic of this asymmetric grid is shown in Figure 3.21. As a real world example a series of points are made at increasing distances apart from one another as shown in Figure 3.22, and the exact distances are presented in Table 3.1. From these points the Δx is changing and yet the asymmetry parameter, A_i, is roughly the same for the central three points, showing that no matter the separation between points the possible asymmetry values will remain within an allowed range. This range is limited to $-1 < A_i < 1$, it is evident when studying Equation C.44, where in order to prevent a divide by zero this specific range must be maintained. Also, from a purely conceptual point of view, if there is no asymmetry then A_i is equal to zero (data point 5 in Table 3.1), and if there is complete asymmetry then A_i would be equal to ± 1, this cannot happen as the point would lie on an adjacent point.

With the concepts presented above, the partial differential equations could be solved for any variation in data point separation.

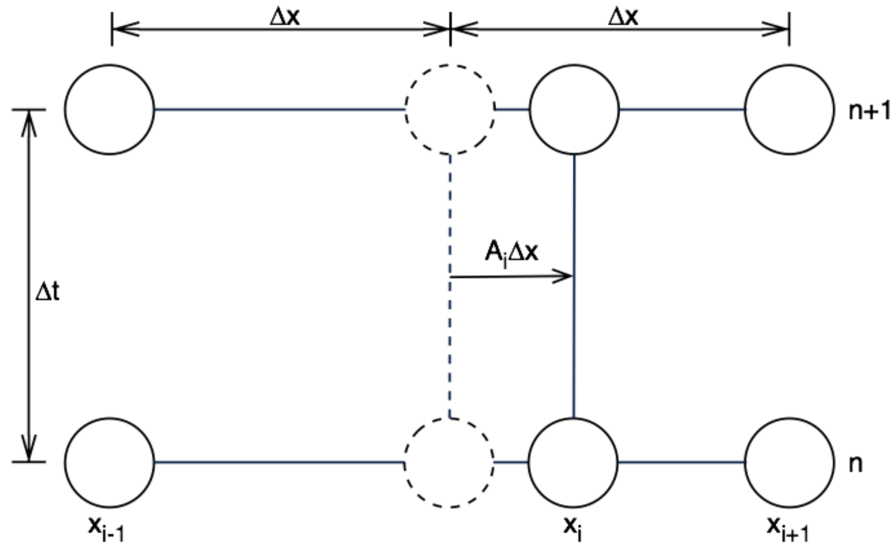

FIGURE 3.21: (Figure adapted from Sobey [94]) Non-Uniform Grid Spacing Schematic.

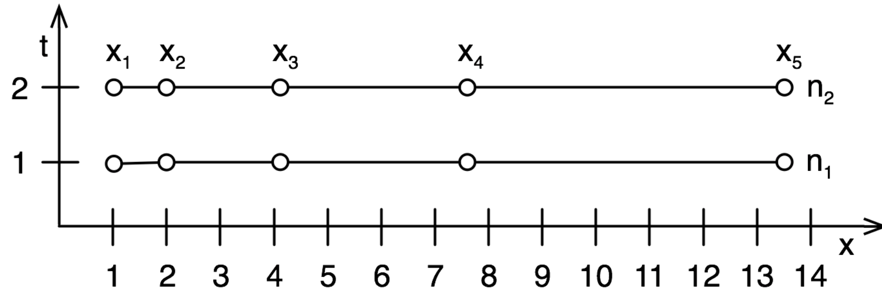

FIGURE 3.22: Realistic Non-Uniform Grid Spacing Example, Schematic.

Point, x_i	Position	Δx_i	A_i
1	1.010	0.994	0.016
2	1.988	1.547	-0.368
3	4.104	2.797	-0.243
4	7.582	4.689	-0.258
5	13.481	5.899	0.000

TABLE 3.1: Realistic Non-Uniform Grid Spacing Example. The fifth point was assumed to be repeated in order to calculate the required data.

3.3.2.2 Area Remapping

The area remapping process can be divided into three steps: (1) calculate the initial surface area between each point along the flat substrate; (2) calculate the surface area between each point along the now evolving nano-structure surface; and (3) reposition each point to match the initial area if required. The first two steps are depicted in Figure 3.23 with

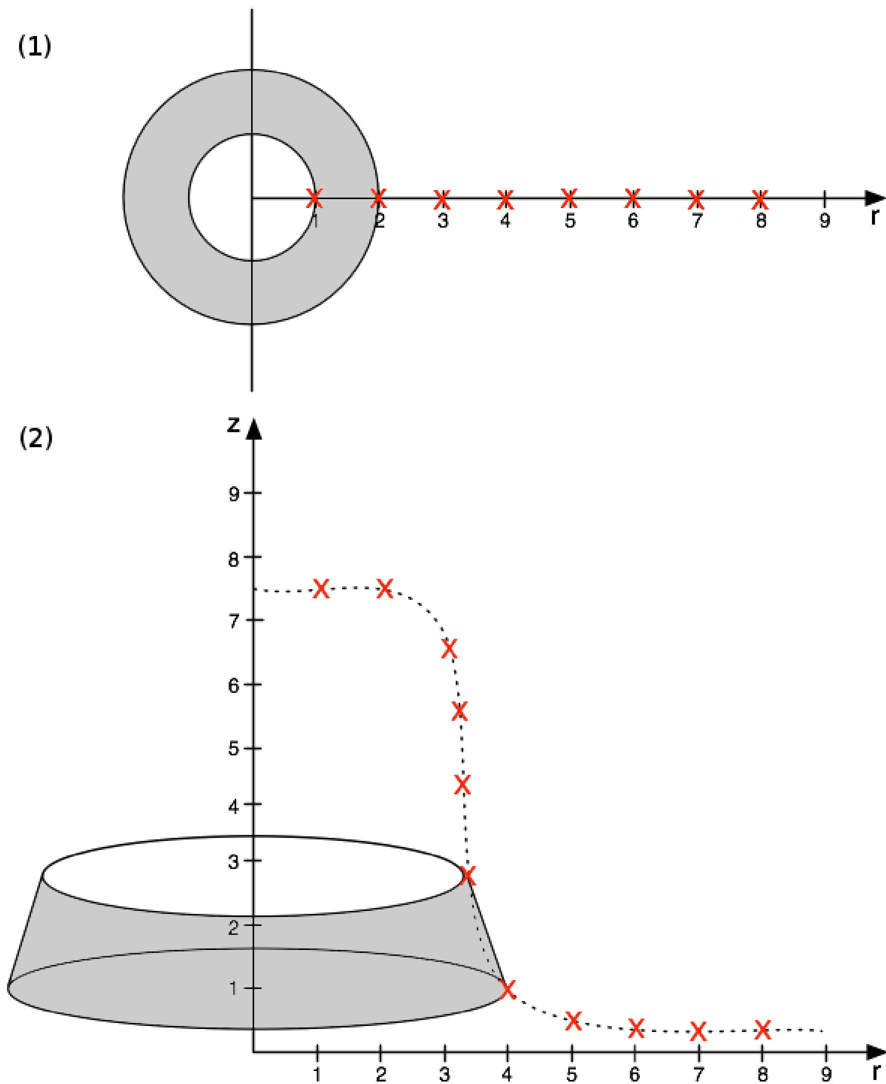

FIGURE 3.23: The first two steps of the area remapping process required to ensure correct diffusion behaviour for an evolving surface. (1) calculate the initial surface area between each point along the flat substrate; (2) calculate the surface area between each point along the now evolving nano-structure surface. An example of the calculated area between two points is highlighted in grey.

an example of the area between two points highlighted in grey. By performing these three steps, the form of Equation 3.28 is maintained no matter the surface topography, and therefore surface diffusion will behave correctly.

It was found that a conical frustum is able to approximate the geometry of both the initial flat substrate and the evolving nano-structure surface, see Figure 3.24. In Figure 3.24, the initial flat substrate can be defined with the height, h, of the conical frustum set to zero.

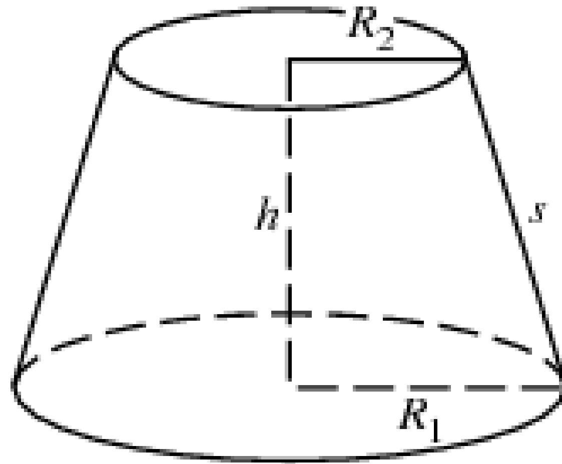

FIGURE 3.24: A conical frustum is defined as a cone with the tip removed, where R_1 is the base radii, R_2, is the top radii, h is the height and s is the slant height[95].

3.3.3 Verification

To test the time-evolution of surfaces generated by the hybrid continuum-Monte Carlo model, the author compared the volumes of deposits calculated by integration of the simulated surfaces ('actual volumes') to the corresponding total volumes of molecules deposited by electrons ('expected volumes'). The expected volumes are given by the product of deposited molecule volume and the total number of molecules dissociated by electrons which is generated automatically by the continuum component of the model.

As shown in Figure 3.25, the time-evolutions of the actual and expected volumes are in excellent agreement. The residual error, also shown on the plot, increases with simulation time. The error is controlled by the number of electrons simulated in each iteration of the Monte Carlo model, and the magnitude of the time step used to propagate the hybrid model in time.

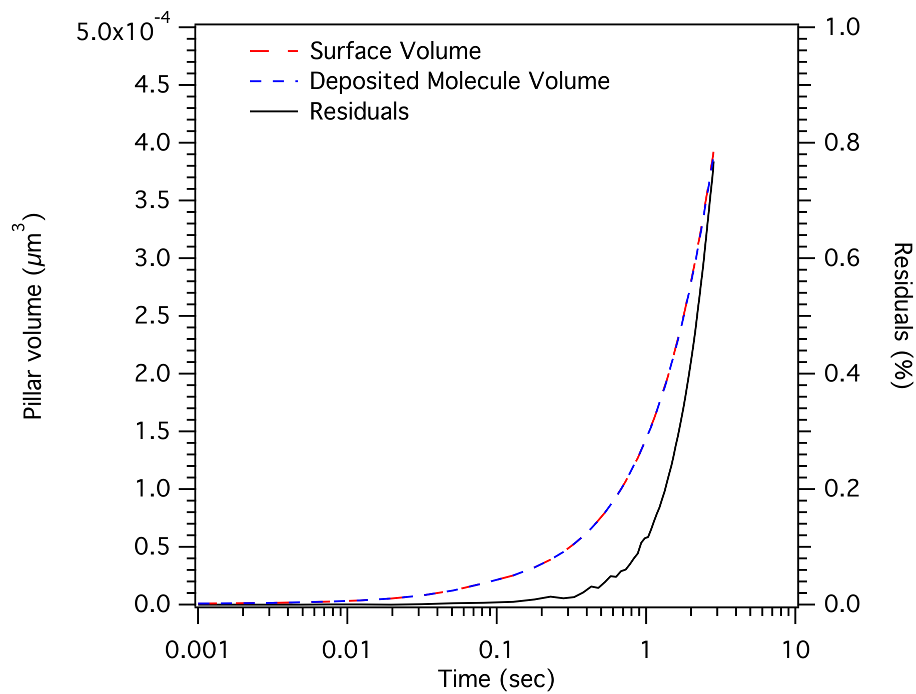

FIGURE 3.25: A comparison of the time-evolution of the actual volume of a deposit simulated by the hybrid model, and the volume expected from the total number of molecules dissociated by electrons. The two volumes are in excellent agreement, with only minor differences of <1% observed in the residuals.

Chapter 4

Hybrid Model Simulator Details

The hybrid model simulator implements the Continuum and Monte Carlo models with the extra extensions discussed in Chapter 3. Each of the modelling methods and extensions form the core modules within the simulator. All code is available in Appendix B.

The hybrid model simulator is simple in operation but does require an understanding of its inputs and outputs. The general flow of the simulator is shown in Figure 4.1. The input parameters are read into the simulator from disk, initial calculations are performed to setup arrays, allocate memory, etc. The main loop of the simulator begins with the Monte Carlo electron trajectories and electron flux profile modules to update the primary electron flux profile with the contribution of backscattered and secondary electrons for a flat surface. The Continuum model and Surface Evolution modules are run to evolve the surface and finally the output data is saved to disk. This main loop continues until the maximum number of time steps has been reached.

The remaining sections in this chapter detail how to use the simulator with an explanation of the various inputs and outputs to the simulator.

4.1 User Operation

The user operation of the EBIED Simulator is broken down into three stages, inputs, running and outputs. The input parameters and the choice of what modules to run are all

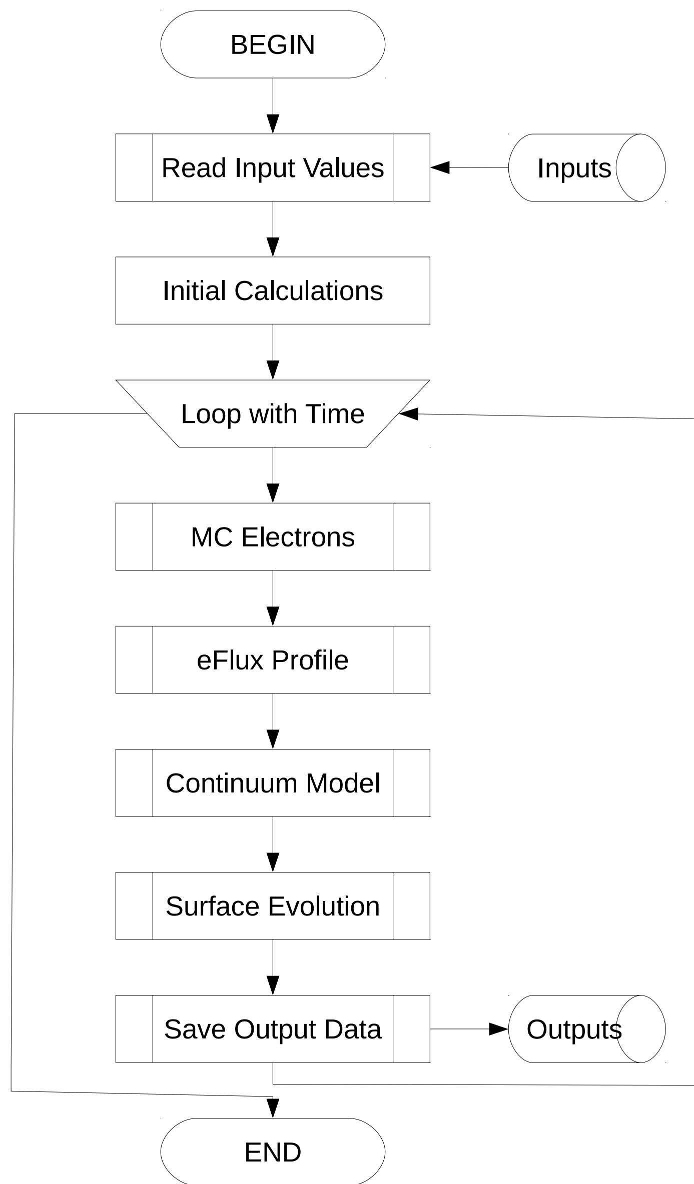

FIGURE 4.1: EBIED Simulator Flowchart describing the general flow and order of modules.

```
 1 number_of_electron_trajectories = 100000 //number of electrons trajectories calculated
 2 number_of_surface_bins = 2000 //number of surface bins * surface unit size, gives surfa
 3 number_of_simulation_time_steps = 200 //number of simulation time steps * time step siz
 4 length_of_electron_energy_deposited_and_electron_maximum_z_depth_array = 2000001 //numb
 5 delta_r = 5.0 //surface step size, angstrom.
 6 delta_t = 1.0E-8 //time step size, sec.
 7 electron_beam.cutoff_energy = 0.05 //cutoff energy in keV in bulk case.
 8 electron_beam.top_hat_abruptness = 10.0 //abruptness of top hat electron beam profile.
 9 electron_beam.diameter = 1000.0 //electron beam diameter in A.
10 electron_beam.energy = 5.0 //input electron beam energy in keV.
11 electron_beam.current = 1000.0 //electron beam current, in pA.
12 electron_beam.tilt = 0.0 //tilt of the electron beam, in degrees.
13 layered_material_interface_depth = 0.0 //layered material interface depth, in A.
14 upper_material.atomic_number = 78.0 //upper material atomic number (Pt).
15 upper_material.atomic_weight = 195.084 //upper material atomic weight (PtC5).
16 upper_material.density = 21.45 //upper material density, in g/cm3 (PtC5).
17 upper_material.epsilon = 0.03 //effective energy required to produce an SE, in keV.
18 upper_material.lambda = 5.0 //effective SE escape depth, in A.
19 lower_material.atomic_number = 14.0 //lower material atomic number.
20 lower_material.atomic_weight = 28.085 //lower material atomic weight.
21 lower_material.density = 2.329 //lower material density, in g/cm3.
22 lower_material.epsilon = 0.09 //effective energy required to produce an SE, in keV.
23 lower_material.lambda = 27.0 //effective SE escape depth, in A.
24 gas_temperature = 290.0 //gas temperature, in K.
25 substrate_temperature = 250.0 //substrate temperature, in K.
```

Figure 4.2: Example input text file for the EBIED Simulator.

done via the one input text file, see Figure 4.2. Each of the input parameters are labelled with a short description and, if required, physical units. Each of the module options are also labelled for what each on/off flag does. It is noted that there is no error checking for the input file, such that all numbers need to maintain its integer or double precision data types and due to the multi-node nature of the code, for example, the number of electron trajectories needs to be divisible by the number of nodes.

The EBIED Simulator has only been run on machines with OSX or Linux. There is no foreseeable issue with running on a Windows machine. The code is run inside the terminal and requires OpenMPI (http://www.open-mpi.org/) to be installed. A full tutorial from initial installation to running a simulation is detailed in Section 4.2.

The first time the code is run, or if any changes are made to the simulator code, it will need to be (re)complied, using the mpicc command. Once complied, the EBIED Simulator is run via the mpirun command, here is also where the number of nodes are specified (e.g. "mpirun -n 4 ./simulator.o" for four nodes). Initially the simulator will print to terminal the input parameters read from the input text file followed by the current simulation percentage and finally an estimated time until completion.

There are many output text files for the EBIED simulator that contain the following data:

- A log file where all text printed to the terminal window is recorded

- Backscattered electron coefficient and secondary electron yield
- Locations where backscattered electron left the surface and a count
- Locations where secondary electron left the surface and a count
- Amount of electron flux at each point on the surface
- Growth rate at each point along the surface
- Deposited molecule concentration at each point on the surface
- Adsorbed precursor molecule concentration at each point on the surface, one for etchants and one for deposition
- Z height of each point on the surface

The form of each output file is a series of data arranged in columns for easy analysis/plotting, see an example in Figure 4.3.

4.2 Tutorial: How to run the EBIED simulator

To run the EBIED simulator from a clean install to viewing an output file the user needs to follow this tutorial.

4.2.1 Mac OS X

1. Goto https://github.com/kennethreitz/osx-gcc-installer/ download and install the correct package for your OSX version.
2. Confirm the installation is successful by running the command gcc in a Terminal window (Applications/ Utilities) and getting the return message of no input files. A return message of command not found means the installation is not successful.
3. Goto http://www.open-mpi.org/software/ download the latest stable release, e.g. openmpi-1.6.5.tar.bz2.

Surface Height				
Time(sec) 1.1e-08		Time(sec) 3.641e-08		Time(sec) 6
r(A)	z(A)	r(A)	z(A)	r(A)
5	0.000137755	5	0.000294104	
9.999	0.000122803	9.999	0.000275101	9.99
14.998	0.000115481	14.998	0.000264036	14.99
19.997	0.00011189	19.997	0.000257416	19.99
24.996	0.000110172	24.996	0.00025343	24.99
29.995	0.000109358	29.995	0.000251109	29.99
34.994	0.000108986	34.994	0.000249598	34.99
39.993	0.000108779	39.993	0.000248647	39.99
44.992	0.000108593	44.992	0.000247958	44.99
49.991	0.000108434	49.991	0.000247366	49.99
54.99	0.000108235	54.99	0.000246767	54.9
59.989	0.000108046	59.989	0.000246174	59.98
64.988	0.000107823	64.988	0.000245507	64.98
69.987	0.000107584	69.987	0.00024486	69.98
74.986	0.000107327	74.986	0.000244225	74.98
79.985	0.000107055	79.985	0.000243567	79.98
84.984	0.00010678	84.984	0.000242852	84.98
89.983	0.000106524	89.983	0.000242103	89.98
94.982	0.000106252	94.982	0.000241264	94.98
99.981	0.000105956	99.981	0.000240386	99.98
104.98	0.000105603	104.98	0.000239526	104.9

Figure 4.3: Example output text file for the EBIED Simulator.

4. Extract the file. Navigate to the extracted folder in a Terminal window with the command cd, e.g. cd ~/Downloads/openmpi-1.6.5.

5. Enter the command ./configure

6. Once complete enter the command make all install.

7. Confirm the installation is successful by running the command mpicc and getting the return message of no input files. A return message of command not found means the installation is not successful.

8. Navigate to the location of the EBIED simulator, e.g. cd ~/Downloads/EBIED.

9. Compile the EBIED simulator with the command mpicc simulator.c -o simulator.o, assuming no error messages the compilation was successful.

10. Open the input parameters text file in the inputs directory, ~/Downloads/EBIED/inputs and modify the input parameters required for your simulation, see section 4.3 for details about each parameter.

11. Run your simulation with the command mpirun -n X ./simulator.o, where X is the number of physical cores of your system.

12. Your simulation will now print various outputs to the Terminal window as well as to the log file in the outputs directory, ~/Downloads/EBIED/outputs.

13. Once your simulation is complete you will find the output data in the outputs directory, see section 4.4 for details about each output file.

4.2.2 Ubuntu - Linux

1. Open a terminal window and enter the command mpicc, note the packages required to install.

2. Enter the command mpirun, note the packages required to install.

3. Enter the command sudo apt-get install X, where X is all noted packages.

4. Confirm the installation is successful by running the command mpicc and getting the return message of no input files. A return message of command not found means the installation is not successful.

5. Navigate to the location of the EBIED simulator, e.g. cd ~/Downloads/EBIED.

6. Compile the EBIED simulator with the command mpicc simulator.c -o simulator.o -lm, assuming no error messages the compilation was successful.

7. Open the input parameters text file in the inputs directory, ~/Downloads/EBIED/inputs and modify the input parameters required for your simulation, see section 4.3 for details about each parameter.

8. Run your simulation with the command mpirun -n X ./simulator.o, where X is the number of physical cores of your system.

9. Your simulation will now print various outputs to the Terminal window as well as to the log file in the outputs directory, ~/Downloads/EBIED/outputs.

10. Once your simulation is complete you will find the output data in the outputs directory, see section 4.4 for details about each output file.

4.3 Summary of Parameters within the EBIED Simulator Input File

This section describes each parameter within the input_parameters.txt text file for the EBIED simulator, with a general description of the parameter and its allowed value. Each parameter is of the double data type unless otherwise specified as the integer data type. The large list of input parameters is divided into the subsections, Simulation Parameters, Electron Beam Parameters, Material Parameters, Precursor Parameters, Module Toggles, and Miscellaneous Parameters for easy lookup.

4.3.1 Simulation Parameters

- number_of_electron_trajectories
  - The number of electrons to be simulated during each call of the Monte Carlo module.
  - Only integer values greater than 1 are accepted where the number inputted must be divisible by the number of physical cores with no remainder.
- number_of_surface_bins
  - The number of bins that the Continuum model will use to store the surface, this number multiplied by the 'delta_r' parameter will give the maximum length in angstroms.
  - Only integer values greater than 2 are accepted.
- number_of_simulation_time_steps

  - The number of time steps the simulator will iterate over.
  - Only integer values greater than 999 are accepted.

- length_of_electron_energy_deposited_and_electron_maximum_z_depth_array
  - The size of the array that stores the electron energy deposited into each Z slice and each electron's maximum Z depth in the simulation area both above and below zero.
  - Only odd integer values greater than 1 are accepted. It is recommended that a very large number is used to prevent simulation errors, e.g. 1000001.

- delta_r
  - The size of each surface bin, in angstroms, this number multiplied by the 'number_of_surface_bins' parameter will give the maximum surface length.
  - Only values greater than 0.0 are accepted.

- delta_t
  - The size of each time step, in seconds.
  - Only values greater than 0.0 are accepted.

4.3.2 Electron Beam Parameters

- electron_beam.cutoff_energy
  - The energy, in kilo-electron volts, where an electron scattering within the material will be said to have stopped scattering.
  - Only values greater than 0.0 are accepted.

- electron_beam.top_hat_abruptness
  - The abruptness of the top hat electron beam profile, see the excel file 'electron beam shape generator.xlsx' for the effect this parameter has on the electron beam profile.
  - Only values greater than 0.0 are accepted.

- electron_beam.diameter
  - The diameter of the electron beam, in angstroms.
  - Only values greater than 0.0 are accepted.
- electron_beam.energy
  - The accelerating voltage of the electron beam, in kilo electron volts.
  - Only values greater than 0.0 are accepted.
- electron_beam.current
  - The current of the electron beam, in pico Amperes.
  - Only values greater than 0.0 are accepted.
- electron_beam.tilt
  - The tilt angle of the electron beam, in degrees.
  - Only values greater than 0.0 and less than 89.9 are accepted.

4.3.3 Material Parameters

The EBIED simulator assumes that there are two materials making up the structure being simulated, for example the deposition of carbon on a gold substrate or the etching of a silicon dioxide layer on pure silicon. The atomic composition of these two materials are input by the user into the upper_material and lower_material parameter structures. The Z depth where the transition between the two materials occurs is contained within the layered_material_interface_depth parameter, it is noted that the upper material is always above the lower material. A single material system can be used if both the upper and lower material parameters are the same.

Only the parameters for the upper material are listed here as they are same for the lower material.

- layered_material_interface_depth

  - The transition Z depth between the upper and lower material, in angstroms.
  - Any value is accepted.

- upper_material.atomic_number
  - The atomic number of the upper material.
  - Only values greater than 0.0 are accepted.
- upper_material.atomic_weight
  - The atomic weight of the upper material.
  - Only values greater than 0.0 are accepted.
- upper_material.density
  - The density of the upper material, in grams/centimetre cubed.
  - Only values greater than 0.0 are accepted.
- upper_material.epsilon
  - The effective energy required to produce a secondary electron in, kilo electron volts.
  - Only values greater than 0.0 are accepted.
- upper_material.lambda
  - The effective secondary electron escape depth, in angstroms.
  - Only values greater than 0.0 are accepted.
- substrate_temperature
  - The temperature of the growing structure/substrate material, in kelvin.
  - Only values greater than 0.0 are accepted.

4.3.4 Precursor Parameters

The EBIED simulator accepts parameters for both a deposition and etching precursors where each one can be turned off by simply lowering respective partial pressures to zero. Listed here are the parameters for the deposition precursor only, the etch precursor accepts the same type of parameters.

- gas_temperature
  - The temperature of the arriving precursor gas(es), in kelvin.
  - Only values greater than 0.0 are accepted.
- deposit_precursor.gas_partial_pressure
  - The partial pressure of the deposition precursor gas molecules, in pascal.
  - Only values greater than 0.0 are accepted, where a value of 0.0 will turn the gas off.
- deposit_precursor.reactive_product_molecular_mass
  - The molecular mass of the reactive product formed when the dissociation of the deposition precursor occurs, in kilograms.
  - Only values greater than 0.0 are accepted.
- deposit_precursor.surface_area
  - The surface area the deposition precursor occupies when adsorbed to the substrate surface, in angstroms squared.
  - Only values greater than 0.0 are accepted.
- deposit_precursor.desorption_energy
  - The size of the energy barrier required for a physisorbed deposition precursor molecule to desorb from the substrate surface, in electron-volts.
  - Only values greater than 0.0 are accepted.
- deposit_precursor.desorption_attempt_frequency

  - The frequency where an adsorbed deposition precursor molecule attempts to get over the energy barrier required for desorption to occur, in seconds.
  - Only values greater than 0.0 are accepted.

- deposit_precursor.diffusion_energy
  - The size of the energy barrier required for the diffusion of an adsorbed deposition precursor molecule across the substrate surface, in electron-volts. Note, if the calculated diffusion coefficient from this value is less than 1.0, diffusion for both the deposition and etch precursor will be turned off.
  - Only values greater than 0.0 are accepted.

- deposit_precursor.diffusion_attempt_frequency
  - The frequency where an adsorbed deposition precursor molecule attempts to get over the energy barrier required for diffusion to occur, in seconds. Note, if the calculated diffusion coefficient from this value is less than 1.0, diffusion for both the deposition and etch precursor will be turned off.
  - Only values greater than 0.0 are accepted.

- deposit_precursor.sticking_coefficient
  - The sticking coefficient for the deposition precursor to physisorb to the substrate surface.
  - Only values greater than 0.0 are accepted.

- deposit_precursor.PE_electron_cross_section
  - The electron cross section for the primary electrons incident on the growing structure/substrate, in angstroms squared.
  - Only values greater than 0.0 are accepted.

- deposit_precursor.BSE_electron_cross_section
  - The electron cross section for the backscattered electrons leaving the growing structure/substrate, in angstroms squared.

  - Only values greater than 0.0 are accepted.

- deposit_precursor.SE_electron_cross_section
  - The electron cross section for the secondary electrons leaving the growing structure/substrate, in angstroms squared.
  - Only values greater than 0.0 are accepted.
- deposit_pinned_reaction_electron_cross_section
  - The reaction electron cross section of pinned molecules, in angstroms squared.
  - Only values greater than 0.0 are accepted.
- deposit_precursor_reaction_electron_cross_section
  - The reaction electron cross section of the deposition precursor molecules, in angstroms squared.
  - Only values greater than 0.0 are accepted.

4.3.5 Module Toggles

- toggle.electron_trajectory_simulator
  - Turns the Monte Carlo module on or off.
  - An integer value of 0 turns the module off and an integer value of 1 turns the module on.
- toggle.electron_trajectory_tracking
  - Turns the recording of every electron scattering event on or off and saves it to disk.
  - An integer value of 0 turns the tracking off and an integer value of 1 turns the tracking on.
- toggle.electron_beam_projection
  - Turns the electron beam projection module on or off.

  - An integer value of 0 turns the module off and an integer value of 1 turns the module on.

- toggle.electron_beam_shape
  - Switches the electron beam shape between a top hat or a gaussian profile.
  - An integer value of 0 is for a gaussian profile and an integer value of 1 is for a top hat profile.

- toggle.surface_evolution
  - Turns the surface evolution module on or off.
  - An integer value of 0 turns the module off and an integer value of 1 turns the module on.

- toggle.previous_simulation
  - Causes the EBIED simulator to load from a previous simulation file or not.
  - An integer value of 0 does not load a previous simulation and an integer value of 1 does.

- toggle.coverage
  - Causes the EBIED simulator to begin the simulation with monolayer surface coverage or no surface coverage.
  - An integer value of 0 is for no surface coverage and an integer value of 1 is for monolayer surface coverage.

4.3.6 Miscellaneous Parameters

The following section of parameters contains those needed by the various modules to perform specific tasks during the simulation.

- number_of_points
  - The number of points where the polynomial fit is computed for the surface evolution.

  - Only odd integer values greater than 1 are accepted, note the number of points must be greater than the order parameter.

- seed
  - The seed used by the random number generator.
  - Only integer values greater than 0 are accepted, a value of -1 will cause the simulator to use a random value.

- precursor_diffusion_tolerance
  - The tolerance percentage that the absorbed precursor concentration in the final surface bin is allowed to change before warning the user.
  - Only values greater than 0.0 are accepted.

4.4 Summary of Outputs from the EBIED Simulator

The section contains a summary of each output file for the EBIED simulator.

- logfile.txt, contains the same output that is presented to the user in the Terminal window as the EBIED simulator runs.
- output_bksct_se.txt, contains two columns listing the backscattered electron and secondary electron yield at each time the output data is saved.
- output_e_BSE.txt, contains the number of backscattered electrons exiting each surface bin as a function distance from electron beam axis for each time the output data is saved.
- output_e_SE.txt, contains the number of secondary electrons exiting each surface bin as a function distance from electron beam axis for each time the output data is saved.
- output_electron_flux_profile.txt, contains the electron flux profile as a function distance from electron beam axis for each time the output data is saved.

- output_growth.txt, contains the growth rate of the deposited molecular concentration as a function distance from electron beam axis for each time the output data is saved.
- output_N_big_D.txt, contains the deposited molecule concentration as a function distance from electron beam axis for each time the output data is saved.
- output_N_little_d.txt, contains the deposition precursor concentration as a function distance from electron beam axis for each time the output data is saved.
- output_N_little_e.txt, contains the etch precursor concentration as a function distance from electron beam axis for each time the output data is saved.
- output_surface.txt, contains the surface height as a function distance from electron beam axis for each time the output data is saved.
- output_track.txt, contains the coordinates of each scattering event each electron made within the growing structure/substrate. Note, a custom Matlab script was written to parse this data as it contains the x, y, z coordinates for each scattering event separated by zeros for each electron.
- input_previous_simulation.txt, contains the height of each surface bin and the distance each surface bin is from the electron beam axis, including the concentrations of the deposit, etch, and deposited molecules. These values allow a simulation to be restarted at a later time.

4.5 Summary of Warning and Error Messages within the EBIED Simulator

This section contains a summary of each warning/error message within the EBIED simulator to help the user with any issues that may arise. The warning messages do not stop the running of the simulator but act to inform the user that there maybe an issue. The error messages do stop the running of the simulator as these are problems that cannot be overcome.

- Warning (Code n): An electron travelled outside the defined depth array. Increase the size of input parameter, length_of_electron...maximum_z_depth_array, to prevent this warning.
  - This warning message tells the user to increase the size of the quoted input parameter so the electron maximum z depth and deposited energy can be recorded. The numbered code is for the software developer to know where in the EBIED simulator code this warning occurred.
- Warning! n electron(s) escaped the simulation area. Increase the size of input parameter, number_of_surface_bins, to prevent this warning.
  - This warning message tells the user that an electron has escaped the bounds of the simulation area and to increase the quoted input parameter so it can be recorded.
- Warning: The deposition precursor molecule diffusion coefficient is less than 1.0! Turning diffusion off!
  - This warning message tells the user the input diffusion related parameters for the deposition precursor resulted in a diffusion coefficient of less than 1.0, and therefore, the Continuum model will proceed with no diffusion. If the Continuum model is to run as normal using a diffusion coefficient of less than 1.0 there would be an error.
- Warning: The etch precursor molecule diffusion coefficient is less than 1.0! Turning diffusion off!
  - This warning message tells the user the inputted diffusion related parameters for the etch precursor resulted in a diffusion coefficient of less than 1.0, and therefore, the Continuum model will proceed with no diffusion. If the Continuum model is to run as normal using a diffusion coefficient of less than 1.0 there would be an error.
- Warning (Deposit Gas): The dimensionless diffusion coefficient, D'=n, does not obey the inequality, $0 < D' < 0.5$, surface diffusion may no longer behave correctly.

Increase the size of input parameter, delta_r, or decrease the size of input parameter, delta_t, to prevent this warning.

  - This warning message tells the user the dimensionless diffusion coefficient for the deposition precursor does not obey the inequality, see Section 3.1.1, and how to fix the issue.

- Warning (Etch Gas): The dimensionless diffusion coefficient, D'=n, does not obey the inequality, $0 < D' < 0.5$, surface diffusion may no longer behave correctly. Increase the size of input parameter, delta_r, or decrease the size of input parameter, delta_t, to prevent this warning.

  - This warning message tells the user the dimensionless diffusion coefficient for the etch precursor does not obey the inequality, see section 3.1.1, and how to fix the issue.

- Warning: Deposit precursor molecules have diffused out of final surface bin, surface diffusion may no longer behave correctly. Increase the size of input parameter, number_of_surface_bins, to prevent this warning.

  - This warning message tells the user that enough deposition precursor molecules have diffused out of the final surface bin to cause diffusion to behave not quite correctly and how to fix this issue.

- Warning: Etchant precursor molecules have diffused out of final surface bin, surface diffusion may no longer behave correctly. Increase the size of input parameter, number_of_surface_bins, to prevent this warning.

  - This warning message tells the user that enough etch precursor molecules have diffused out of the final surface bin to cause diffusion to behave not quite correctly and how to fix this issue.

- Error: Tridiagonal Solver Failed, the simulation is now exiting! This is difficult error to solve, suggested solutions are to increase the frequency of the Monte Carlo module or reduce the time step.

  - This error message tells the user that the Crank-Nicholson solver used in the Continuum model failed and provides suggestions to fix this issue in future simulations. Unfortunately this error crashes the simulator.

- Error: The electron beam tilt is greater than 89.9 degrees, the simulator is now exiting! Decrease the electron beam tilt and restart the simulation.
  - This error message tells the user the input electron beam tilt is too high and to reduce this value.

Resolving issues in the EBIED simulator is addressed by warning and error messages. Not all issues can be captured by standard messages, however, and it is the responsibility of the researcher to overcome these on a case by case basis.

Chapter 5

Localized Probing of Gas Molecule Adsorption Energies and Desorption Attempt Frequencies

5.1 Introduction

Electron beam induced etching (EBIE) and deposition (EBID) can be used as analysis techniques for the characterisation of adsorbates at surfaces, enabling the measurement of properties that include the gas molecule adsorption energy (E_a), desorption attempt frequency (k_0), diffusion coefficient, and electron dissociation cross-section [1, 4, 20, 52, 63, 64, 67, 70, 71]. Unlike complimentary conventional techniques such as thermally programmed desorption (TPD)[96, 97] the use of EBIE and EBID as analysis techniques offers high spatial resolution, and the ability to perform measurements on adsorbates in the steady state at various pressures and temperatures.

Characterisation of adsorbates by EBID and EBIE typically involves the measurement of deposition or etch rates as a function of one or more control parameters (e.g. electron beam flux, substrate temperature, precursor vapor pressure). The rates are analysed using methods based on models of EBIE/EBID rate kinetics to extract the adsorbate properties of interest. There has been a few prior studies[19, 52] using EBIE/EBID rate kinetics, the

interpretation is not clear due to very different experimental methodology from standard surface science techniques (e.g. TDP).

A specific example discussed in the literature is the measurement of E_a realized by Arrhenius analysis of electron beam induced deposition rates. A number of groups have reported that the value of E_a measured by EBID is ~ 2.5 to 5 times lower than expected [1, 70, 71], and the surprising result that E_a scales inversely with the electron beam current used to perform EBID.[71] These observations are attributed to electron stimulated desorption[70, 71].

In the following chapter an Arrhenius analyses of EBID rates yield activation energies and pre-exponential factors that are expected to scale with electron beam current (and other parameters). Explanation of these dependencies using established models of EBID rate kinetics is discussed. A series of conditions are defined where the activation energies and prefactors correspond to E_a and k_0, respectively. The results are also applicable to EBIE and illustrate that: (i) EBID and EBIE are indeed expected to enable meaningful, quantitative characterization of adsorbates, and (ii) the results of such analyses require careful interpretation in the context of mechanisms behind EBIE/EBID rate kinetics.

The results in this chapter use the Hybrid Continuum-Monte Carlo model as defined earlier in Chapter 3 to simulate EBID with cyclopentadienyl trimethyl platinum as the precursor gas. The specific conditions used were:

Electron Beam

- Diameter, 10.0 nm
- Energy, 5.0 keV
- Current, 10 pA to 10 nA

Precursor Molecule

- Atomic number, 78.0 (Pt)
- Atomic weight, 195.084 amu (PtC_5)

- Density, 21.45 g/cm^3 (PtC_5)
- Energy required to produce a SE, 0.03 keV (Pt)[98]
- SE escape depth,[98] 5.0 Å
- Pressure, 10 mPa
- Surface area,[55] 35.7 Å^2
- Sticking coefficient,[55] 1.0
- Net electron dissociation cross section, 1.0 Å^2
- Desorption energy,[99] 666 meV
- Desorption prefactor,[22] 10^{13} Hz
- Diffusion energy,[99] 0.114 eV
- Diffusion prefactor,[55] 4.16×10^9 A^2/s

5.2 Arrhenius analysis of deposition rates

The precursor molecule adsorption energy (E_a) and attempt frequency (k_0) can, in principle, be obtained by Arrhenius analysis of the growth rates of deposits made by EBID. This approach is valid only if the growth rates scale exponentially with reciprocal substrate temperature ($1/T$), and the scaling is caused solely by the temperature-dependence of the adsorption time (τ):

$$\tau = \frac{1}{k_0} exp\left(\frac{E_a}{k_B T}\right). \tag{5.1}$$

EBID growth rates can be calculated by solving equations for the rate of change of concentration of precursor gas adsorbates at the substrate surface ($\partial N_a/\partial t$)[19, 44], referred here as Equation 5.2, and in Chapter 3 as Equation 3.1:

$$\frac{\partial N_a}{\partial t} = \Lambda - \frac{N_a}{\tau} - \frac{\partial N_\alpha}{\partial t} + D_a \nabla^2 N_a, \tag{5.2}$$

where t is time and a and α represent gas molecule adsorbates and its fragments generated by electron induced dissociation, respectively. $\frac{\partial N_a}{\partial t}$ is given by a sum of fluxes representing precursor adsorption (Λ), desorption ($\frac{N_a}{\tau}$), electron induced dissociation ($\frac{\partial N_\alpha}{\partial t}$) and surface diffusion ($D_a \nabla^2 N_a$). N is number density at the surface, τ is the adsorbate residence time at the surface, and D_a is the diffusion coefficient. Adsorbate surface coverage (Θ) is typically assumed to be limited to 1 monolayer by the Langmuir isotherm:

$$\Lambda = sF(1-\Theta), \tag{5.3}$$

$$\Theta = AN_a \tag{5.4}$$

where s and F are the gas molecule sticking coefficient and flux, respectively, and A is the area of a single surface site. The rate of change of concentration of the fragments α is given by:

$$\frac{\partial N_\alpha}{\partial t} = n\sigma f N_a, \tag{5.5}$$

where f is electron flux, σ is the effective cross-section [100] for electron-induced dissociation of the adsorbates a, and n is the number of fragments generated per adsorbate. The etch or deposition rate scales with $\frac{\partial N_\alpha}{\partial t}$. In the case of deposition, the vertical growth rate is given by:

$$\frac{\partial h}{\partial t} = V_\gamma \sigma f N_a, \tag{5.6}$$

where h is the deposit height, and V_γ is the volume of a single molecule added to the substrate per adsorbate dissociated in the deposition reaction.

In cylindrical coordinates the following parameters are functions of the radial distance (r) from the electron beam axis: N_a, N_α, Λ, Θ, f and h. The gas molecule flux (F) is typically assumed to be independent of r and t over the area irradiated by electrons (exceptions are discussed in [55, 101, 102]):

$$F = \frac{P}{\sqrt{2\pi m_g k_B T_g}}, \tag{5.7}$$

where P is gas pressure, m_g is the gas molecule mass, k_B is Boltzmann's constant and T_g is gas temperature.

Referring to standard EBID equations, 5.2 and 5.6, Arrhenius analysis of growth rates therefore yields E_a and k_0 only if:

(1) the adsorption flux Λ is independent of T over the range of T used to perform the analysis (i.e. $\Theta \approx 0$, $ds/dT \approx 0$, and $dF/dT \approx 0$, so that $\Lambda \approx sF$), referred to as the 'athermal adsorption flux condition',

(2) the thermal desorption rate is much greater than the adsorbate dissociation rate (i.e. $\tau^{-1} \gg \sigma f$), referred to as the 'reaction-rate limited growth condition',

(3) net transport of adsorbates through diffusion is negligible (i.e. $\nabla^2 N_a \approx 0$), referred to as the 'negligible diffusion condition', and

(4) the growth rate is measured in the steady state (i.e. $\partial N_a/\partial t \approx 0$), referred to as the 'steady state growth condition'.

Under these conditions, Eqns. 5.2 and 5.6 reduce to:

$$0 = sF - \frac{N_a}{\tau} - \sigma f N_a, \tag{5.8}$$

$$\frac{\partial h}{\partial t} = \frac{V_\gamma \sigma f s F}{\tau^{-1} + \sigma f}, \tag{5.9}$$

$$\approx V_\gamma \sigma f s F \tau, \tag{5.10}$$

and Arrhenius analysis of $\partial h/\partial t$ yields E_a/k_B and $V_\gamma \sigma f s F k_0^{-1}$ (i.e. both the adsorption energy E_a and desorption attempt frequency k_0 can be deduced if the quantity $V_\gamma \sigma f s F$ is known).

As an example of the number of deposits an experimentalist would need to collect, Figure 5.1, contains a small sample of EBID deposits each grown for increasing time and temperature. In Figure 5.1 an observation is that as time progresses the deposit size increases and once the FWHM of the deposit no longer increases (see insert example) steady state is reached. Another observation is as temperature increases the size of the deposits decreases and becomes dependent on temperature. This behaviour is needed in order to study desorption.

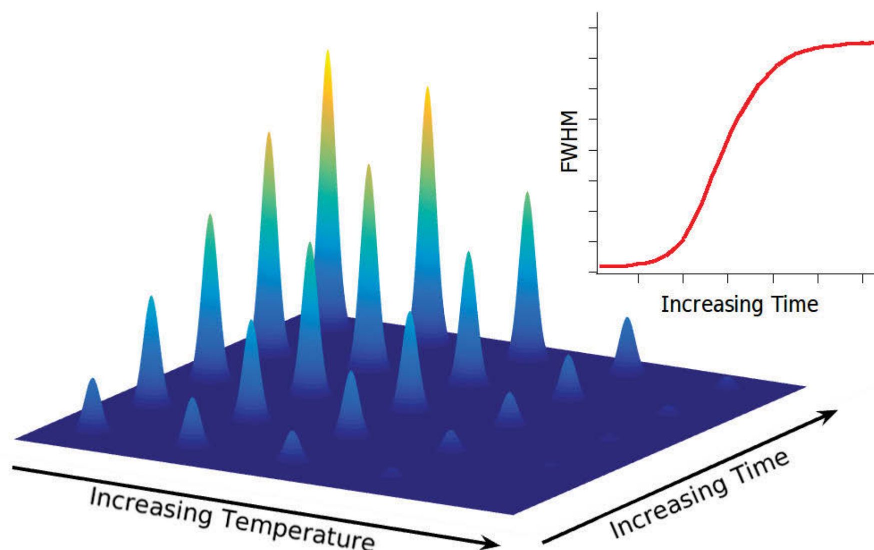

Figure 5.1: A series of deposits shown as a function of time and temperature, with an insert depicting the typical progression of a deposit's FWHM with time. The point where the FWHM no longer increases with time is considered steady state growth.

Each of the subsequent Figures in this Chapter all required an Arrhenius analysis to calculate the desired quantity. As an example of the Arrhenius analysis performed, Figure 5.2 contains a series of plots illustrating the dependence of $\ln(\partial h/\partial t)$ on $1/T$ is linear when the four conditions defined are satisfied. The dependence of $\ln(\partial h/\partial t)$ on $1/T$ can become sub-linear if one or more of the conditions is violated. Specifically, the extent of the deviation from linearity scales with the rate of change of the magnitude of the underlying process with reciprocal temperature (e.g. the rate of change of coverage caused by depletion with $1/T$ is greatest when the electron beam current is in the range of ~ 50 to 500 pA).

In the following sections, the effect conditions 1-4 has on the activation energies and pref-actors obtained by Arrhenius analysis of EBID rates is shown. The author demonstrates that if any one of these conditions is not satisfied then Equation 5.10 is invalid and the values yielded by Arrhenius analysis diverge from E_a and k_0.

In the following, the term 'activation energy' and 'pre-factor' are used to mean the values obtained by Arrhenius analysis of EBID growth rates, irrespective of whether or not they correspond to the adsorption energy and the desorption attempt frequency, respectively.

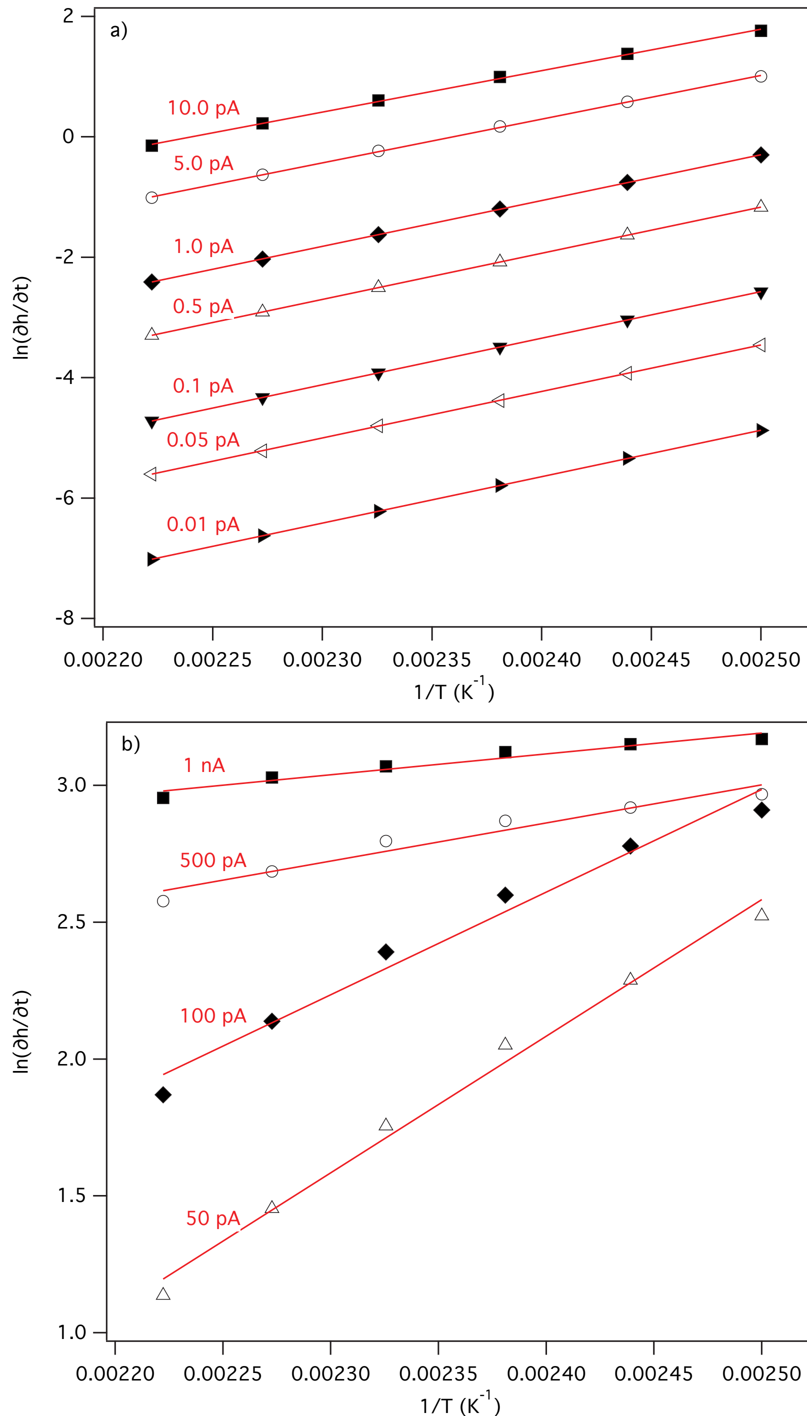

FIGURE 5.2: Plots of $\ln(\partial h/\partial t)$ versus $1/T$ simulated using electron beam currents in the range of 0.01 pA – 1 nA in the absence of diffusion. Also shown are the corresponding straight line fits used to obtain the activation energies plotted in Figure 5.6.

5.2.1 Athermal adsorption flux condition

Gas molecules incident onto the substrate either adsorb to or reflect from the surface. The fraction that adsorb is given by the product $s(1-\Theta)$ in Equation 5.3, and the fraction that reflect is given by $s\Theta$. Hence, if the coverage Θ changes with T, the rate at which gas molecules are reflected from occupied surface sites affects the temperature dependence of the adsorption flux Λ which, in turn, affects $N_a(T)$, $\partial h/\partial t(T)$ and the activation energy and prefactor yielded by Arrhenius analysis of EBID rates.

The surface coverage Θ increases with P and $1/T$ as shown in Figure 5.3 for the platinum precursor considered in the simulations. The purple area indicates the range of P and T corresponding to the 'athermal adsorption flux condition', where $\Theta \approx 0$. In this part of the parameter space, the effect of $\Theta(T)$ on Arrhenius analysis of deposition rates is negligible. Outside this region, $\Theta \gg 0$ and the activation energy and prefactor obtained by Arrhenius analysis of $\partial h/\partial t$ both approach zero as $\Theta \to 1$. This is illustrated in Figure 5.4 and 5.5 by plots of the activation energy and prefactor obtained as a function of T (using a current density of 1.27×10^{-7} nA/nm^2 and a precursor pressure of 10 mPa). The shapes of the curves in Figure 5.4 and 5.5 are governed by the functional form of the Langmuir isotherm used in Equation 5.3 (i.e. the dependence of Λ on Θ). Other isotherms will change the shapes of the curves, but will not be independent of Θ, except for the idealized special case of unlimited multilayer adsorption of non-interacting adsorbates in which $\Lambda = sF$.

The primary practical implication of the athermal adsorption flux condition is that Arrhenius analysis of EBID rates must be performed under conditions of negligible surface coverage because $\partial N_a/\partial T \to 0$ as $\Theta \to 1$ due to reflection of gas phase precursor molecules from occupied surface sites. If $\Theta > 0$ then both E_a and k_0 will be underestimated by an amount that scales with Θ in a manner defined by the adsorption isotherm.

We note that since $\Lambda = sF(1-\Theta)$, s and F must also not vary with T in order for the athermal adsorption flux condition to be satisfied. The sticking coefficient can typically be assumed to be independent of T (as has been done in the simulations). This assumption is not valid in some cases, such as that of activated sticking encountered in chemisorption [63]. The gas molecule flux F (given by Equation 5.7) is independent of substrate temperature T if it is varied independently of the gas temperature (T_g). This requirement is satisfied

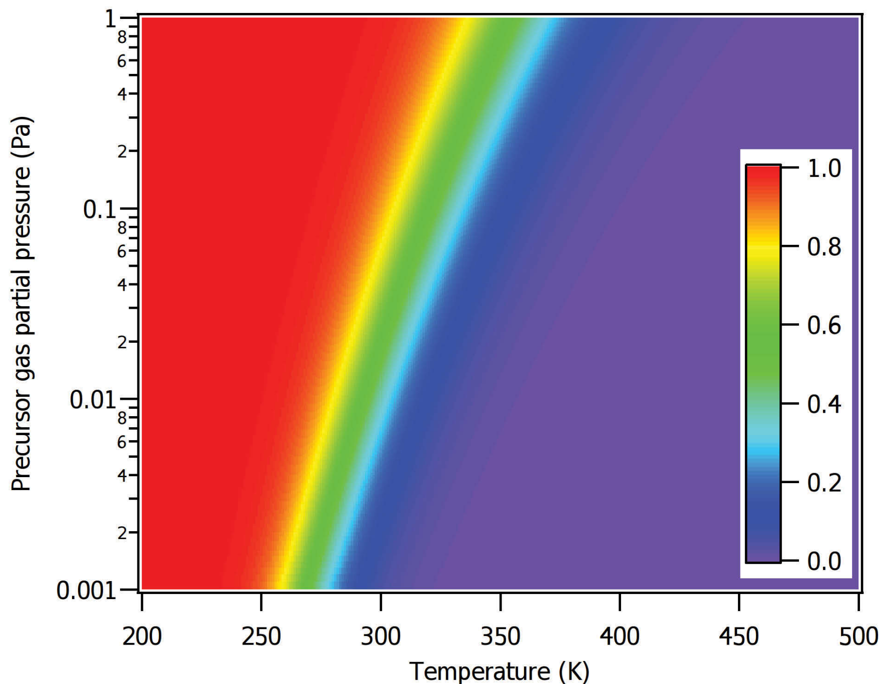

FIGURE 5.3: Precursor coverage (Θ) plotted as a function of pressure (P) and substrate temperature (T) in the limit of zero electron flux ($f \to 0$). The purple region indicates the range of P and T over which the effect of Θ on Arrhenius analysis of EBID rates is negligible.

in most EBID experiments because standard EBID is a cold-wall deposition technique in which T_g is dominated by the temperature of the capillary used to deliver the precursor gas.

5.2.2 Reaction-rate limited growth condition

The reaction-rate limited growth condition is violated (i.e. $\tau^{-1} \not\gg \sigma f$) if the rate at that adsorbates are consumed in the deposition reaction (σf) is significant relative to the thermal desorption rate. In this situation, Equation 5.10 is not a good approximation of Equation 5.9, and the activation energies and prefactors yielded by Arrhenius analyses of EBID rates diverge from E_a and k_0, respectively. This is illustrated in Figure 5.6 by a plot of the activation energy obtained as a function of beam current in the absence of diffusion (the simulation is performed with D fixed at zero in order to delineate the effects of σf from the additional effects of diffusion that are discussed below). An increase in beam current causes an increase in the electron flux (f) that, in turn, causes an increase in the adsorbate

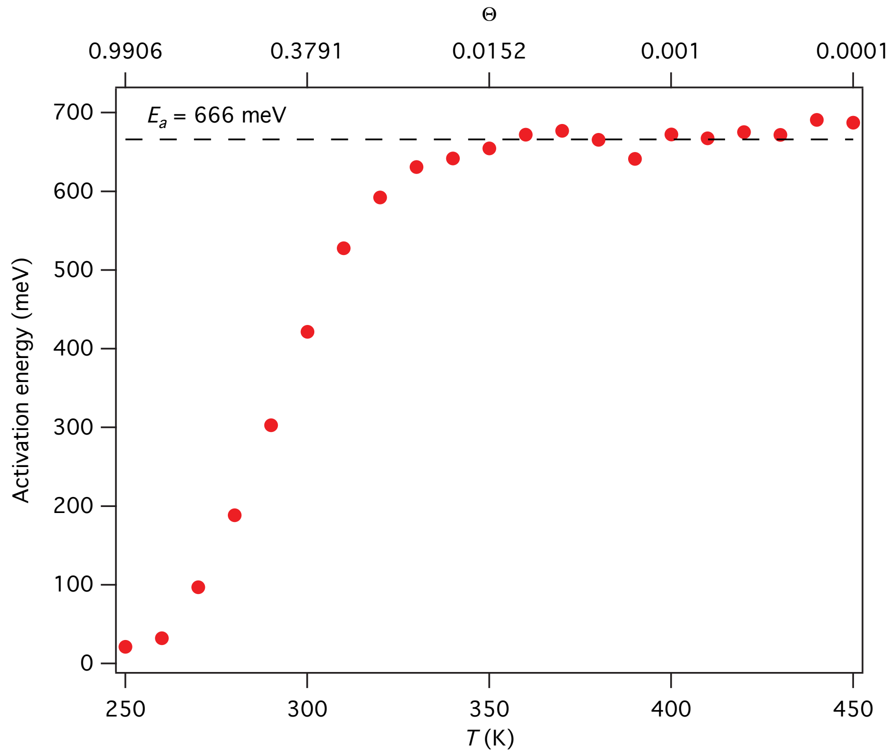

FIGURE 5.4: Activation energy obtained by Arrhenius analysis of the EBID rate ($\partial h/\partial t$) simulated at a number of temperature windows between 250 K and 450 K. The adsorption energy (E_a) of 666 meV is shown as a dashed line. The top axis shows the precursor coverage (Θ) corresponding to each temperature shown on the bottom axis. The activation energy diverges from E_a as $\Theta \to 1$. [Each datapoint was calculated from EBID rates simulated over a temperature window ΔT in the range of 20 to 50 K.]

consumption rate in the deposition reaction (σf). If σf is significant relative to τ^{-1}, the denominator of Equation 5.9 is dominated by the temperature-independent σf. Hence, the activation energies (and prefactors) yielded by Arrhenius analyses of EBID rates approach zero as $\sigma f \to \infty$. More specifically, in the absence of diffusion, the activation energy is directly proportional to the extent of depletion (given by $\Theta_{r\to 0}/\Theta_{r\to\infty}$ since $f = 0$ at $r = \infty$). This is illustrated in Figure 5.6 by a plot of $\Theta_{r\to 0}/\Theta_{r\to\infty}$, superimposed on the plot of activation energy versus beam current.

The primary practical consequence of the reaction-rate limited growth condition is that EBID must be performed in the so-called reaction rate limited growth regime [44] defined by $\tau^{-1} \gg \sigma f$ (i.e. in the regime where electron irradiation does not cause significant depletion of precursor adsorbates).

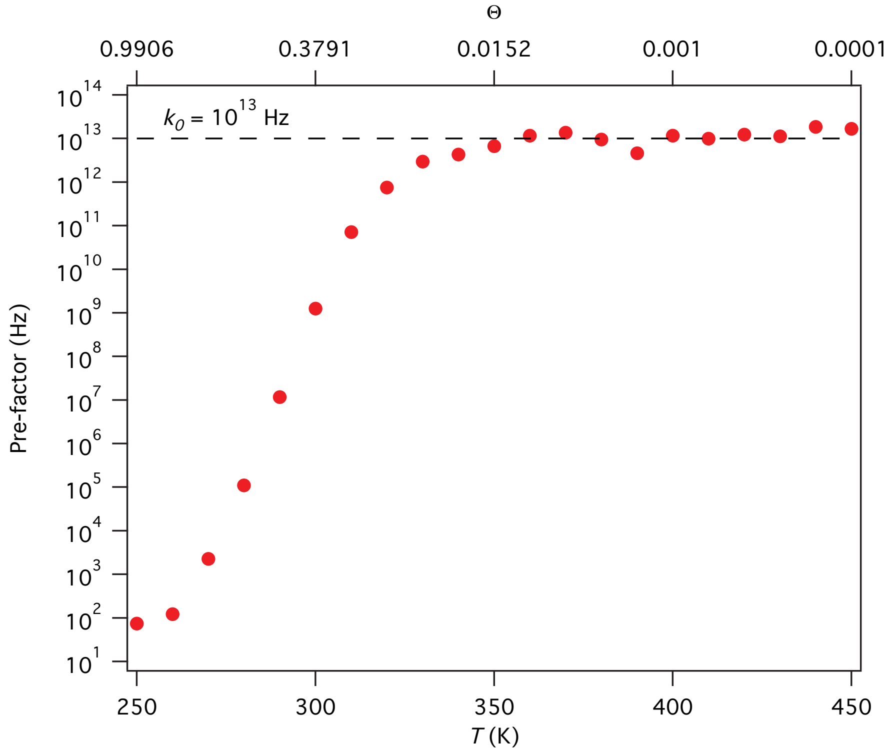

FIGURE 5.5: Prefactors corresponding to the activation energies shown in Figure 5.4. The desorption attempt frequency (k_0) of 10^{13} Hz is shown as a dashed line. The top axis shows the precursor coverage (Θ) corresponding to each temperature shown on the bottom axis. The prefactor diverges from k_0 as $\Theta \to 1$.

5.2.3 Negligible diffusion condition

If the concentration gradient, $\nabla^2 N_a \not\approx 0$, adsorbates are replenished not only through adsorption from the gas phase, but also via diffusion along the substrate surface [44]. This is significant because any adsorbate replenishment mechanism that is distinct from adsorption alters the temperature-dependence of the coverage Θ and hence the activation energy (and prefactor) yielded by Arrhenius analysis of deposition rates.

In EBID, adsorbate replenishment through diffusion is negligible in the reaction rate limited growth regime [44] because net adsorbate transport through diffusion requires an adsorbate concentration gradient. Hence, if the reaction-rate limited growth condition is satisfied (i.e. $\tau^{-1} \gg \sigma f$), then the negligible diffusion condition is automatically satisfied (i.e. $\nabla^2 N_a \approx 0$). This is illustrated in Figure 5.6 by the plot of activation energy versus beam current (simulated with diffusion activated in the model) it shows that the activation energy approaches E_a as the extent of depletion approaches zero.

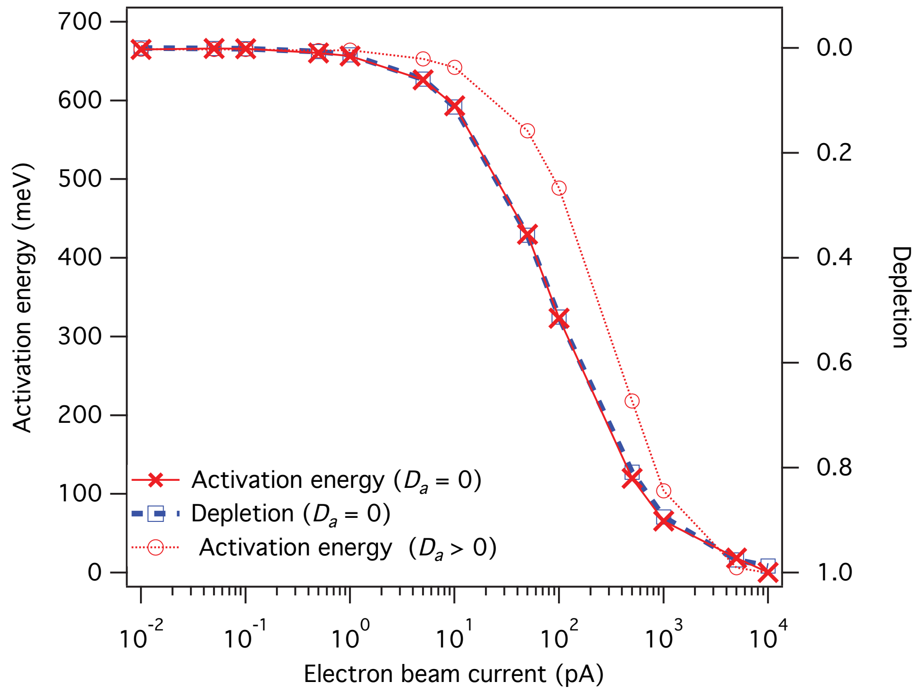

FIGURE 5.6: Dependence of activation energy on beam current simulated in the absence of diffusion ($D_a = 0$) and in the presence of diffusion ($D_a = D_0 e^{-E_d/(k_B T)}$, where E_d is the diffusion energy). Also shown is a plot of adsorbate depletion ($\Theta_{r\to 0}/\Theta_{r\to\infty}$) simulated at the beam axis ($r \to 0$) in the absence of diffusion. The activation energy diverges from the adsorption energy (E_a) of 666 meV as the extent of depletion approaches 1. The precursor pressure was 0.01 Pa and the temperature was varied from 400 to 450 K.

In Figure 5.6, the difference between the curves of activation energy versus beam current simulated with and without diffusion show that the net effect of diffusion is to decrease the discrepancy between the activation energy and E_a. This is expected since the net flow through diffusion acts to replenish adsorbates consumed in EBID, and therefore, alleviates depletion.

5.2.4 Steady state growth condition

In order to obtain E_a and k_0, the Arrhenius analysis must be performed in the steady state (whereby $\frac{\partial N_a}{\partial t} \approx 0$ and $\partial h/\partial t$ is constant). If steady state has not been attained, the growth rate changes during the time interval where $\partial h/\partial t$ is measured by an amount that is different at each substrate temperature, thereby altering the activation energies and prefactors obtained by the Arrhenius analysis of EBID growth rates. In practice, this is problematic only if the reaction rate limited growth condition is not satisfied, the growth

time is too short, or some process such as electron beam damage modifies the deposit height while it is being grown by EBID. Experimentally, the condition of steady state growth can be verified by measuring deposit heights at each temperature versus time, and by showing that the rate of change of height is constant.

5.3 General implications for the determination of adsorbate properties

The discussion and conclusions drawn from this research can also be applied to other research projects. In particular those focused on other precursor adsorbates. In the following section how the details of this research can be applied by others is discussed.

The overarching qualitative implication of the first three conditions (athermal adsorption flux, reaction-rate limited growth and negligible diffusion) is that the activation energy measured by EBID is lower than E_a if any one of the conditions is violated. The discrepancy occurs if the increase in adsorbate concentration with reciprocal temperature is smaller than expected from the thermal desorption rate τ^{-1}. Violations of one or more of these conditions likely account for the fact that activation energies reported in the EBID literature are often lower than E_a, but never greater than E_a [1, 70, 71].

General conclusions are also applicable to EBIE, as it is analogous to EBID, except that Equation 5.6 represents the etch pit depth rather than the deposit height. The above analysis is strictly applicable only to systems where adsorption is described by a single potential well. In cases such as activated chemisorption, described by multiple potential wells and one or more adsorption barriers [63, 72, 103], the above analysis must be re-done using appropriate rate equations in order to determine the correct meaning and scaling of activation energies and prefactors obtained in different temperature regimes. Care must also be taken to ensure that mechanisms that are not accounted for by Equation 5.2 do not alter the temperature dependence of N_a. For example, adsorbate-adsorbate interactions are not included and also, in EBIE, a number of such special cases have been identified[2, 3, 12, 51, 52]. In these instances, the meaning of the measured activation

energies and prefactors must be evaluated on a case-by-case basis, using the appropriate rate equations.

It should be noted that, in the present work, Arrhenius analysis is applied to vertical deposition rates at the beam axis (i.e. $|\partial h/\partial t|_{r\to 0}$). The conclusions are, applicable to Arrhenius analysis of all other measures of growth rate such as the deposit volume [71] and mass [70]. In these cases, the steady state growth condition is more stringent because steady state is attained only once the deposit shape stops changing due to the time-evolution of the electron interaction volume inside a growing deposit (see, for example, the discussions of the time-evolution of deposit shapes and base diameters in [6, 60, 74]).

Qualitatively, the decrease in activation energy with increasing beam current seen in Figure 5.6 is observed experimentally[71]. It is attributed to ESD[70, 71] rather than the phenomenon of adsorbate depletion that is inherent to EBID. ESD causes the adsorbate concentration (N_a) to decrease at a rate $\sigma_E f$, where σ_E is the cross-section for ESD[48, 97, 104, 105]. It can be incorporated into Equation 5.2 in the form of the ESD flux $\sigma_E f N_a$:

$$\frac{\partial N_a}{\partial t} = \Lambda - \left(\frac{N_a}{\tau} + \sigma_E f N_a\right) - \frac{\partial N_\alpha}{\partial t} + D_a \nabla^2 N_a, \tag{5.11}$$

and therefore alters the denominator of Equation 5.9:

$$\frac{\partial h}{\partial t} = \frac{V_\gamma \sigma f s F}{\tau^{-1} + \sigma_E f + \sigma f}. \tag{5.12}$$

Consequently, the ESD rate $\sigma_E f$ affects Arrhenius analysis by contributing to adsorbate depletion in the same manner as the dissociation rate σf. The extent of ESD is typically negligible since, for most adsorbates, σ_E lie in the range[97] $10^{-7}\mathring{A}^2$ to $10^{-2}\mathring{A}^2$ (i.e., in general, $\sigma_E \ll \sigma$). Thus the decrease in activation energy with beam current observed by Li *et al.* [71] is expected from the net effect of $\sigma f + \sigma_E f$ even if σ_E is negligible. It is caused by an increase in the extent of depletion of precursor adsorbates with increasing electron flux (i.e. violation of the reaction rate limited growth condition), irrespective of whether or not ESD plays a significant role in EBID.

5.4 Conclusion

Development of a hybrid continuum-Monte Carlo model of electron beam induced etching and deposition enables accurate calculation of the growth of nano- and micro-structures over the length and time scales used in experiments. The model was used to simulate the dependencies of EBID rates on experimentally controlled growth parameters, and to interpret the physical meaning of activation energies and pre-exponential factors obtained by Arrhenius analysis of EBID rates. The activation energies are shown to correspond to precursor molecule adsorption energies, and the prefactors to desorption attempt frequencies, provided that EBID is performed under specific conditions. It is shown how deviations from these conditions affect the Arrhenius analysis and explained the observed trends as being caused by changes in adsorbate concentration with key EBID experimental parameters.

Chapter 6

Electron Beam Induced Deposition As a Technique for the Analysis of Precursor Molecule Diffusion Barriers and Pre-Factors

6.1 Introduction

Continuing the analysis outlined in the prior chapter, the same model was used to develop a procedure for the determination of the diffusion barrier (E_D) and pre-exponential factor (D_o), by exploiting the fact that the precursor molecules consumed in EBID are replenished through two pathways – adsorption from the gas phase and diffusion along the surface. It is shown that diffusion gives rise to a growth rate component which can be isolated, and that Arrhenius analysis of this growth rate component can be used to deduce E_D and D_o. The technique is compelling relative to conventional diffusion analysis techniques [106, 107] because it yields both E_D and D_o and can be combined with the analysis in Chapter 5 to obtain the corrugation factor (defined as the ratio of the diffusion energy and the adsorption energy) from a single set of self-consistent data with nanoscale spatial resolution.

Is it noted that Utke et al. [4] and Szkudlarek et al.[65] previously used the deposit geometry to estimate the diffusion coefficient of $Cu(hfac)_2$. Fowlkes et al.[64] estimated the diffusion coefficient of $W(CO)_6$ by an analysis of the growth rates of pillars fabricated by EBID using a pulsed electron beam. In this chapter E_D and D_o are probed by measuring the dependence of diffusion rate on substrate temperature.

The results presented use the Hybrid Continuum-Monte Carlo model as defined earlier in Chapter 3 to simulate EBID with cyclopentadienyl trimethyl platinum using a 5 keV, 1 nA, stationary, Gaussian electron beam with a diameter of 100 nm, a diffusion energy and pre-exponential factor of 114 meV and 4.16×10^9 Å^{2}s^{-1}, respectively, and substrate temperatures in the range of 120 to 350 K. The diffusion energy and pre-exponential factor used here were respectively reported in the work by Shen et al. [99] and Winkler et al.[55](the values were assumed to be the same on the growing deposit and the substrate). All other model input parameters are identical to those used in Chapter 5. Langmuir adsorption is assumed at all temperatures, hence precursor condensation that occurs at low temperatures is ignored by the model. This point is discussed further below.

6.2 Roles of Desorption and Diffusion in EBID

It is shown in Chapter 5 that Arrhenius analysis of EBID rates can be used to obtain the adsorption energy and desorption attempt frequency. Specifically, under appropriate conditions the deposit growth rate, R, can be approximated by:

$$R \approx V\sigma fsF\tau \tag{6.1}$$

$$\approx V\sigma fsF\, \tau_o exp\left(\frac{E_a}{k_BT}\right) \tag{6.2}$$

and a plot of $ln(R)$ vs $1/T$ is linear, and has a slope of $\frac{E_a}{k_BT}$ and a y-intercept of $V\sigma fsF\tau_o$. Equation 6.1 is valid only in the so-called *reaction rate limited growth regime* [5] where adsorbate depletion caused by the electron beam is negligible. In this regime, diffusion plays a negligible role in the replenishment of adsorbates consumed in EBID[10]. However, in the opposite extreme of high depletion (encountered at high electron beam current densities), diffusion can make a very significant contribution to EBID [4, 5, 64]. This is

illustrated in Figure 6.1(a) which shows cross-sectional slices through deposits simulated at a number of temperatures. At the lowest temperature shown in the figure, the deposit has a flat-top geometry because D is negligible, and precursor adsorbates under the Gaussian electron beam are highly depleted – i.e. within the flat region of the deposit, the vast majority of molecules adsorbing through sF are consumed in EBID through $\sigma f N_a$. At elevated temperatures, each deposit contains a characteristic 'ring' generated by adsorbates supplied through surface diffusion (i.e. through the term $D\nabla^2 N_a$ in Equation 3.1). The ring is also seen in the steady state vertical deposition rates $R(r)$ shown in Figure 6.1(b) for a number of temperatures T_n. In the following sections, an Arrhenius analysis of the volumetric growth rate of this ring is used to obtain E_D and D_o.

6.3 Adsorbate Transport Through Diffusion

The diffusion flux in the EBID model (Equation 3.1) yields the net transport of adsorbates across the surface. It consists of a diffusion coefficient, D, multiplied by the driving force of diffusion, $\nabla^2 N_a$. The latter relates the net flow of adsorbates across the surface to the adsorbate concentration gradient at each point on the surface. In cylindrical coordinates, the driving force of diffusion, represented from hereon by c, is given by:

$$c = \nabla^2 N_a = \frac{\partial^2 N_a}{\partial r^2} + \frac{1}{r}\frac{\partial N_a}{\partial r} \tag{6.3}$$

Since c and D appear as a product in Equation 3.1, the dependence of c on r and T must be understood if we are to develop an Arrhenius analysis technique for the determination of E_D and D_o.

Figure 6.2(a) shows $c(r)$ profiles corresponding to the $R(r)$ profiles in Figure 6.1(b) at a number of temperatures T_n, simulated under conditions of significant depletion caused by the electron beam. The extent of depletion is illustrated in Figure 6.2(b) by plots of the surface coverage $\Theta(r)$ and the normalized electron flux profile $f_N(r)$.

The plots of $c(r)$ show that it is comprised of two distinct regions separated by r_o: one positive (at $r < r_o$, where in this particular case $r_o \sim 125$ nm) and the other negative (at

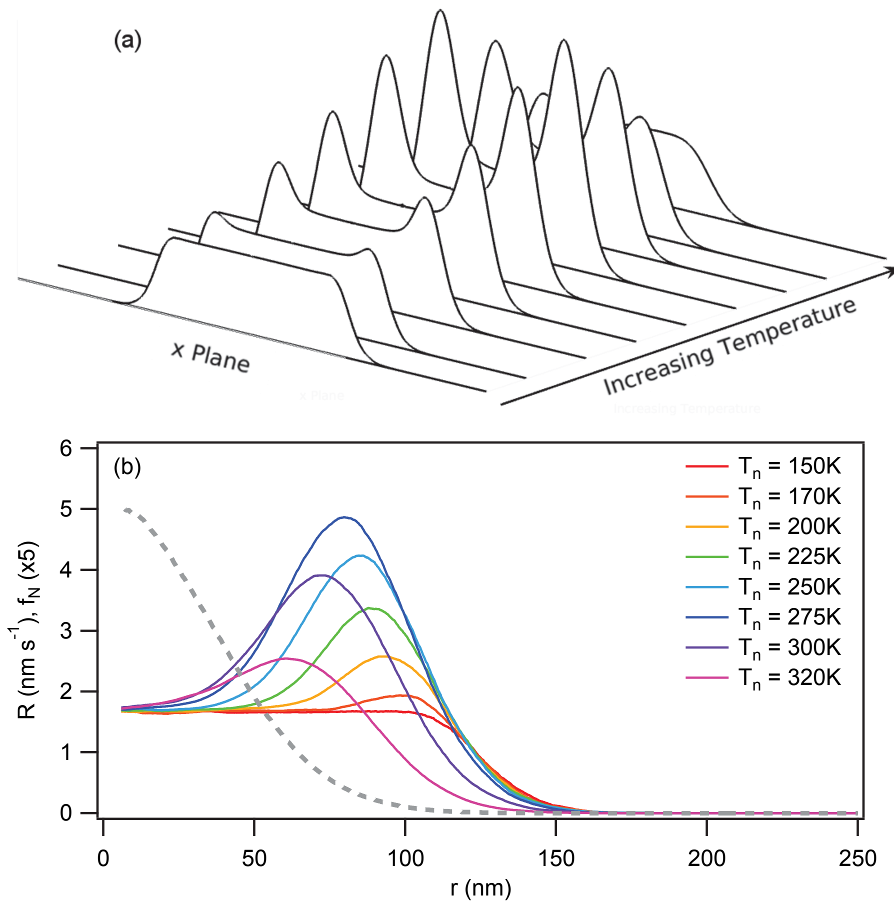

FIGURE 6.1: Effects of diffusion on the shapes of deposits grown by EBID. (a) A series of deposits simulated as a function of temperature. At low temperatures, the deposit geometry is unaffected by diffusion, but at elevated temperatures each deposit contains a characteristic 'ring' generated by adsorbates supplied through surface diffusion. (b) Steady state vertical growth rates (R) calculated as a function of distance (r) from the electron beam axis at a number of temperatures (T_n). All simulations were performed using a Gaussian electron beam under conditions of high adsorbate depletion near the beam axis. The normalized electron flux profile $f_N(r)$ is shown as a dashed curve in (b).

$r > r_o$). To explain the dependence of c on r, we must consider how adsorbates are replenished during EBID. When electrons irradiate the substrate, an adsorbate concentration gradient (i.e. the coverage gradient seen in Figure 6.2(b)) forms as precursor molecules are consumed in the deposition reaction. The consumed adsorbates are replenished through adsorption from the gas phase and diffusion along the surface. The diffusing adsorbates originate in the negative part of $c(r)$, labelled 'source' in Figure 6.2(a), and are dissociated in the positive region, labelled 'sink'. The integral of $c(r)$ over the sink region ($0 \leq r \leq r_o$) is a fluence C (in units of Å^{-2}) corresponding to in-diffusing adsorbates that are dissociated by electrons and give rise to the formation of the rings seen in Figure 6.1. CD is

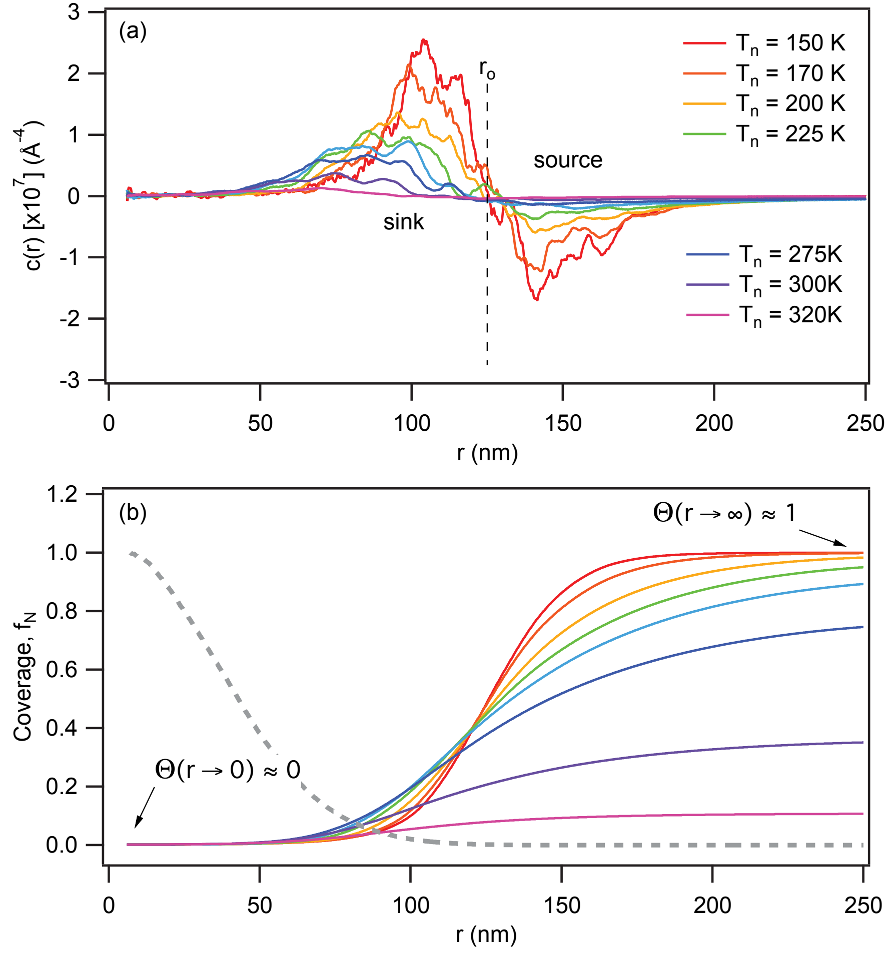

FIGURE 6.2: (a) Steady state plots of the driving force of diffusion (c) versus distance (r) from the electron beam axis at a number of temperatures (T_n). Each $c(r)$ profile contains two distinct regions corresponding to the source and sink of adsorbates that diffuse along the surface and are consumed in EBID. The sink and source are separated by r_o, shown as a dashed line at 125 nm. (b) Corresponding adsorbate coverage profiles ($\Theta(r)$), and the normalized electron flux profile (($f_N(r)$, dashed grey curve).

therefore the corresponding growth rate, and the volumetric deposition rates of the rings (in units of $\text{Å}^3\text{s}^{-1}$) are therefore given by:

$$R_{VD} = VCD \tag{6.4}$$

$$= VC\ D_o \exp\left(\frac{-E_D}{k_B T}\right) \tag{6.5}$$

Equations 6.4 and 6.5 are analogous to Equation 6.1 and 6.2, and as shown below, a plot of $ln(R_{VD}/C)$ versus $1/T_m$ can be used to obtain E_D and D_o, provided the quantity C is known at each of the temperatures T_m used to generate the Arrhenius plot. The Arrhenius

analysis must be performed on the quantity R_{VD}/C (rather than R_{VD}) because C is a function of T, as shown in Figure 6.3(a). Corresponding plots of $R_{VD}(T)$ and $R_{VD}/C(T)$ are shown in parts (b) and (c) of the figure, and discussed below.

The above arguments, however, are valid only over a particular temperature window, shown in Figure 6.3, as is discussed in detail below.

Experimentally, C can be obtained at any given temperature T_n by measuring the deposition rate $R(r)$ and rearranging Equation 3.2 to give $N_a(r)$:

$$N_a = R/V\sigma f \tag{6.6}$$

which can then be substituted into Equation 6.3 to obtain $c(r)$. Integration of $c(r)$ over $0 \leq r \leq r_o$ gives C. Hence, at a given temperature T_n, a plot of $ln(R_{VD}/C)$ versus $1/T_m$ can be obtained from the measured experimental growth rates ($R(r, T_m)$ and $R_V(T_m)$, where T_m represents a set of temperatures centered on T_n used to generate the Arrhenius plot (see, for example, the simulated Arrhenius plots shown in Figure 6.4). The quantities V, σ and $f(r, T)$ must also be known because they appear in Equation 6.6. The electron flux profile $f(r, T)$ is a function of T because it includes electrons emitted from the deposit at each temperature used to generate the Arrhenius plot (e.g. see how the deposit geometry changes with T in Figure 6.1). It can be obtained using Monte-Carlo simulations of electron-solid interactions [80, 88, 108] and the geometries of deposits grown by EBID. If the dissociation cross-section σ is not known, it can be approximated by an analysis of EBID growth rates [64].

6.4 Extraction of Diffusion Energies and Pre-Exponential Factors

Figure 6.5(a) shows plots of activation energies and pre-exponential factors obtained by Arrhenius analyses of R_{VD} at a number of temperatures T_n. The corresponding diffusion coefficients are shown in Figure 6.5(b). Each point n was generated as follows:

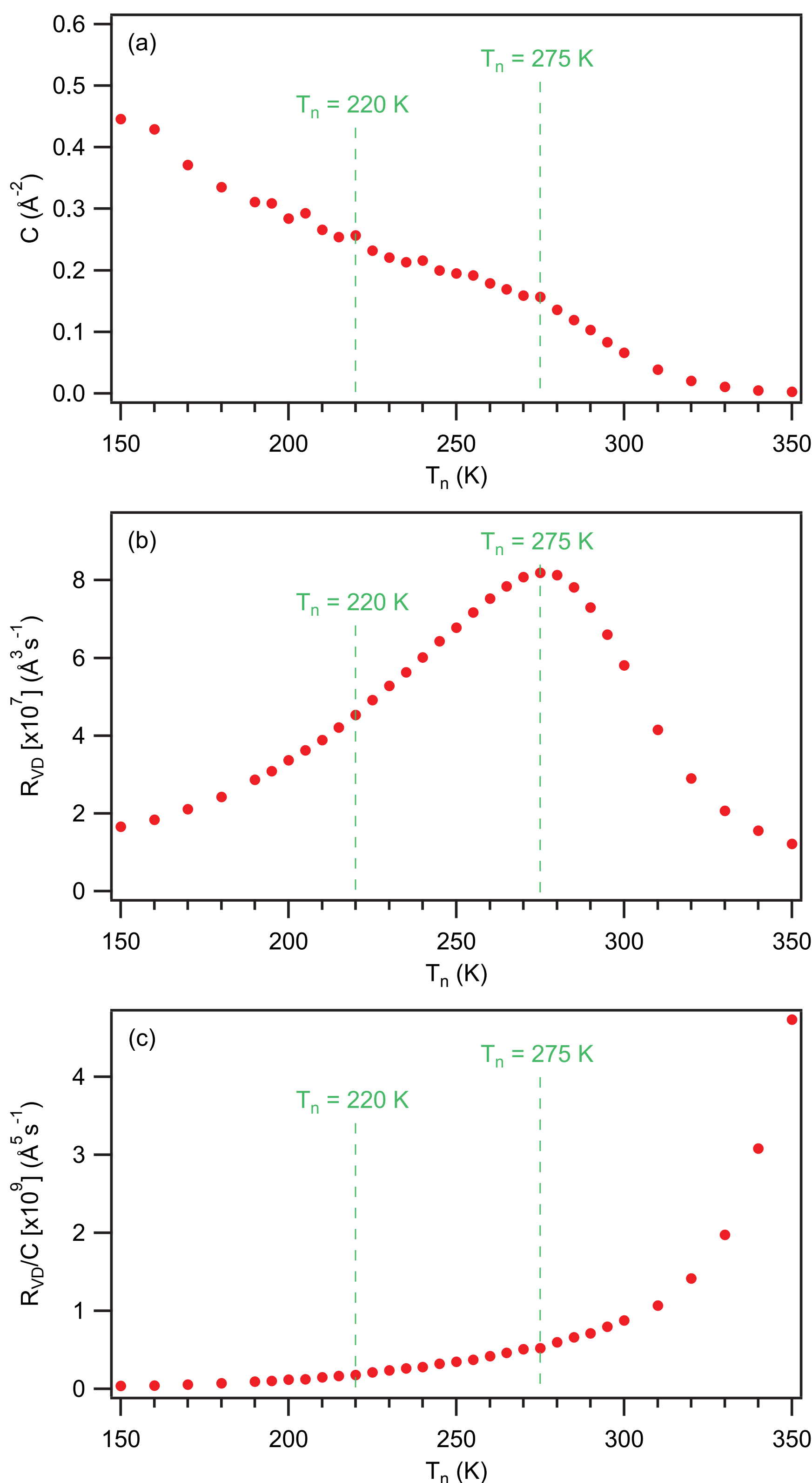

FIGURE 6.3: (a) The fluence (C) found by integrating $c(r)$ over the sink ($0 \leq r \leq r_o$) shown in Figure 6.2(a), plotted for a number of temperatures T_n. (b,c) Corresponding plots of R_{VD} and R_{VD}/C versus T_n. The Arrhenius analysis method yields good approximations to E_D and D_o at temperatures between $\sim$ 220 K and $\sim$ 275 K.

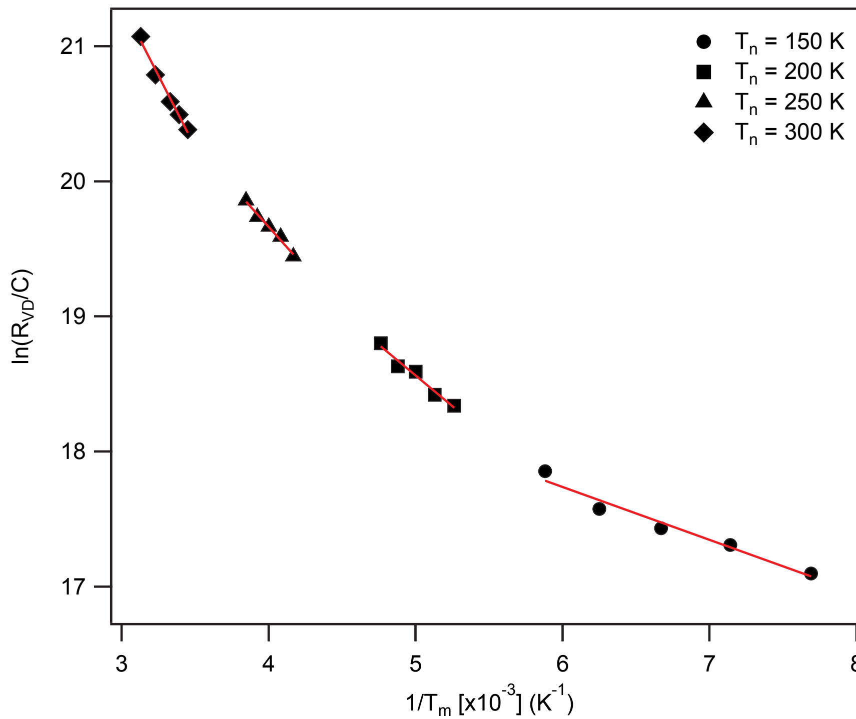

FIGURE 6.4: Arrhenius plots used to generate the data in Figure 6.5(a) at temperatures (T_n) of 150, 200, 250 and 300 K.

- EBID was simulated at each of the temperatures (T_n) plotted in Figure 6.5(a) in order to obtain a set of deposits and the corresponding vertical growth rates $R(r, T_n)$. Such data can be obtained experimentally by performing EBID as a function of substrate temperature.

- $N_a(r, T_n)$ was calculated using Equation 6.6, and using the values of V and σ used in the simulations. These values must be known if this procedure is applied to experimental data, as must $f(r, T_n)$. The latter can be obtained using Monte-Carlo simulations of electron trajectories in the deposit made at each temperature.

- $C(T_n)$, shown in Figure 6.3(a), was calculated by substituting $N_a(r, T_n)$ into Equation 6.3, and integrating $c(r, T_n)$ over $0 \leq r \leq r_o$.

- R_{VD}, shown in Figure 6.3(b), was calculated by performing each simulation with and without diffusion and by taking the difference between the resulting simulated volumes. It was analyzed on the basis of Equation 6.4. An experimental method for the measurement of R_{VD} is discussed below.

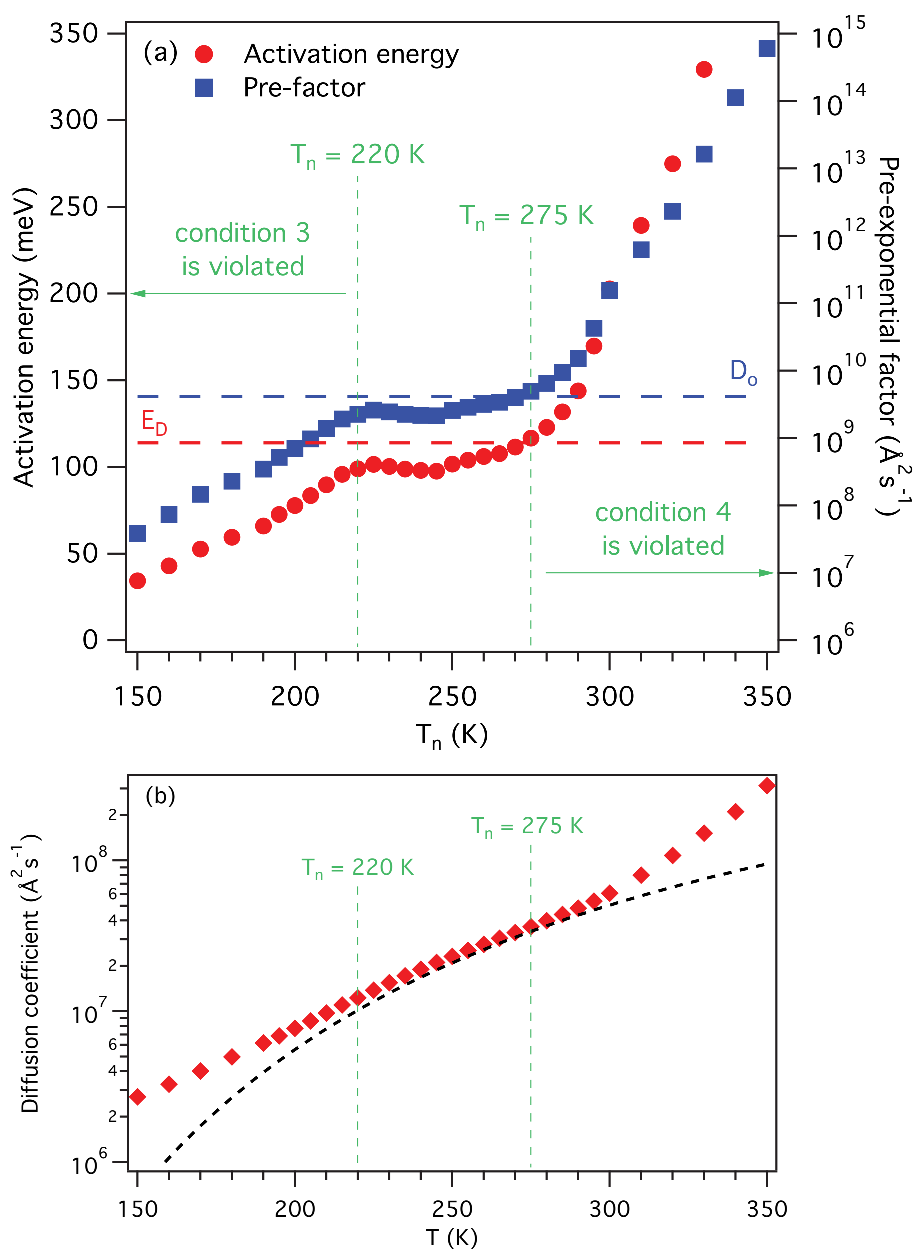

FIGURE 6.5: (a) Activation energy (red) and pre-exponential factor (blue) obtained by Arrhenius analysis of R_{VD}/C at a number of temperatures T_n. The quantities are approximately equal to E_D and D_o (shown as dashed lines) over the temperature window $220 \lesssim T_n \lesssim 275$ K. (b) Diffusion coefficient (D) versus temperature (dashed black curve), and diffusion coefficients (red diamonds) calculated using the activation energies and pre-exponential factors in (a).

- An Arrhenius plot of $ln(R_{VD}/C)$ was produced for each temperature T_n, using growth rate data from five temperatures T_m centered on T_n (i.e. $m = n + i$, where i is an integer in the range of -2 and 2). Figure 6.4 shows these plots for $T_n = 150$, 200, 250 and 300 K.

- The slope of each Arrhenius plot yields an activation energy (multiplied by $\frac{-1}{k_B T_n}$) and a pre-exponential factor (multiplied by V), as per Equation 6.5.

The data in Figure 6.5(a) shows that the activation energies and pre-exponential factors obtained using this procedure are good approximations to E_D and D_o over a particular temperature range, which in this case is $220 \lesssim T_n \lesssim 275$ K (the 'true' values $E_D = 114$ meV and $D_o = 4.16 \times 10^9$ Å^{2}s^{-1}, shown as dashed lines on the plot, are those that were used in the model [10] that was used to simulate EBID in the first step of the above procedure). It is noted that the method appears to produce correct diffusion coefficients (Figure 6.5(b)) at temperatures between ~ 275 K and ~ 300 K because, in this temperature range, errors in the activation energy and pre-exponential factor offset each other when Equation 2.5 is used to calculate D.

The reasons for why the Arrhenius analysis method fails at temperatures that are too high or too low are discussed in detail below.

6.5 Pre-requisites

Arrhenius analysis can be used to extract E_D and D_o if the following set of conditions is satisfied at each temperature T_n used to perform EBID:

1. *Steady state growth condition*: The vertical growth rate must be constant: $\frac{\partial R}{\partial t} \approx 0$.

2. *Significant adsorbate concentration gradient condition*: Adsorbate coverage must be low near the beam axis (as $r \to 0$) and high far away from the beam ($r \to \infty$).

3. *Diffusion-dominated replenishment condition*: Adsorbate replenishment near the deposit periphery (i.e. near r_o, shown in Figure 6.2(a)), must be dominated by diffusion, which is satisfied when $D\nabla^2 N_a \gg sF$ in the vicinity of r_o.

4. *Efficient adsorbate consumption condition*: In-diffusing adsorbates (i.e those diffusing from the source to the sink shown in Figure 6.2(a)) must be dissociated by electrons, which is satisfied when $\sigma f \gg 1/\tau$.

In the following sections each condition is discussed and with an explanation why the Arrhenius analysis method fails when any one of the above conditions is not satisfied.

6.5.1 Condition 1 - Steady State Growth

During the early stages of EBID, $R(r)$ changes with time as the surface irradiated by the electron beam evolves from that of a horizontal plane into a structure that eventually grows at a constant rate antiparallel to the electron beam (as discussed in [6, 10, 60, 74]). The growth rates used for Arrhenius analysis must be measured in the steady state so that changes in R_{VD}/C with T are caused purely by the temperature dependence of D, and are not affected by the temperature-dependence of the transition from the initial to the steady state. Experimentally, attainment of a steady state can be verified simply by fabricating a set of deposits as a function of growth time, and by measuring $R(r,t)$. Models such as the one used here output the time evolution of $R(r)$; hence, it is easy to ensure that Condition 1 is satisfied, and the results in Figure 6.5(a) were obtained in the steady state.

6.5.2 Condition 2 - Significant adsorbate concentration gradient

Adsorbate depletion under the electron beam is needed in order to generate a concentration gradient that gives rise to net flow of surface-adsorbed precursor molecules. In the absence of a concentration gradient, $c(r) \approx 0$, and diffusion does not contribute to EBID[10, 44]. In the opposite extreme, illustrated for the case of the Langmuir isotherm in Figure 6.2(b), $\Theta(r \to 0) \approx 0$ and $\Theta(r \to \infty) \approx 1$, R_{VD} is maximized and hence errors associated with the measurement of R_{VD} are minimized.

Significant depletion ($\Theta(r \to 0) \approx 0$) occurs in the so-called *mass transport limited growth regime* where the adsorbate dissociation rate is much greater than the adsorbate replenishment rate [44]: $\sigma f \gg sF/N_a + \tau^{-1}$. This condition is realized by using a high electron beam current density.

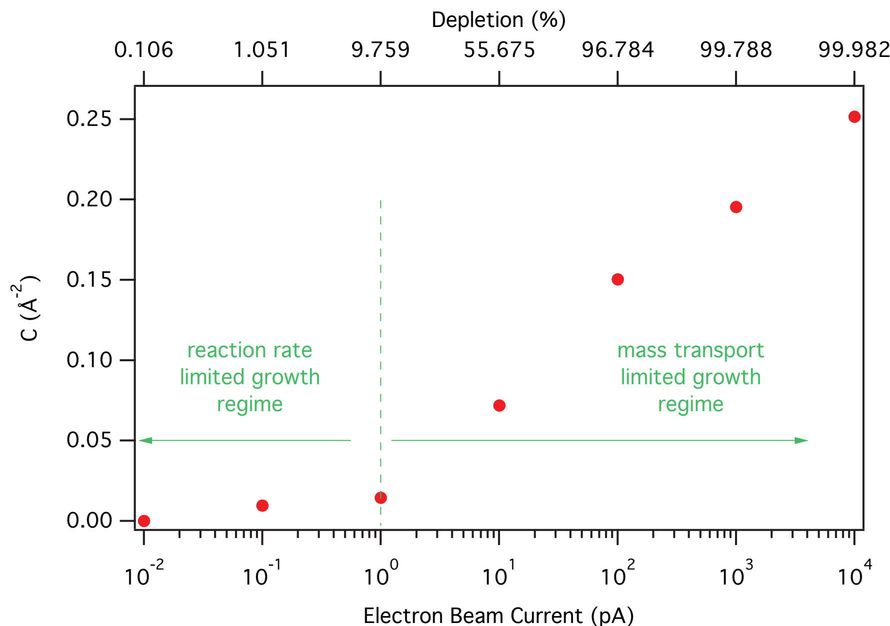

FIGURE 6.6: The fluence C plotted as a function of electron beam current.

Far away from the beam, adsorbate coverage is saturated (i.e. $\Theta(r \to \infty) \approx 1$) when the adsorption rate is much greater than the thermal desorption rate: $sF \gg N_a/\tau$. This condition is realized by using a high precursor gas pressure.

The data in Figure 6.5(a) were obtained in the limit of high depletion and are not affected by condition 2. To demonstrate what happens when the adsorbate concentration gradient is too small, Figure 6.6 shows a plot of C as a function of electron beam current (for a beam diameter of 100 nm). The extent of depletion, defined as the percentage $100\left(1 - \frac{\Theta(r\to 0)}{\Theta(r\to\infty)}\right)$, is shown on the top axis. The data show that $C \approx 0$ in the reaction rate limited growth regime, and that it increases with electron beam current in the mass transport limited growth regime. The Arrhenius analysis method obviously fails as $C \to 0$.

6.5.3 Condition 3 - Diffusion-dominated replenishment

Adsorbates diffusing from the source to the sink shown in Figure 6.2(a) give rise to the formation of the rings seen in Figure 6.1, and to R_{VD}. However, vacant surface sites in the sink are also populated by gas molecules adsorbing from the gas phase. Hence, the growth rate of the ring is dominated by the diffusing adsorbates only if $D\nabla^2 N_a \gg sF$ (in the vicinity of r_o). If this condition is not satisfied (i.e. if the diffusion rate is too low

because T is too low), then R_{VD} has a significant contribution from sF and the magnitude of this contribution increases with decreasing T. The net effect is seen in Figure 6.3(b) as a reduction in the slope of $R_{VD}(T)$ below ~ 220 K. Consequently, the Arrhenius analysis method fails at low temperatures (see Figure 6.5(a)) because the temperature-dependence of R_{VD}/C is not dominated by the temperature-dependence of D (which is defined by Equation 2.5).

To confirm that the failure of the Arrhenius analysis method at low tempertures is caused by violation of condition 3, Figure 6.7 shows a plot of the maximum flux of diffusing adsorbates ($\max[D\nabla^2 N_a(r)]$) versus T_n. The plot reveals that this flux is equal to the adsorption flux (sF) at ~ 200 K. That is, the condition $D\nabla^2 N_a \gg sF$ is violated as the 5 point temperature window used to generate each point T_n in Figure 6.5(a) approaches 200 K.

It is noted that at temperatures in excess of ~ 290 K, the plot in Figure 6.7 shows that $\max[D\nabla^2 N_a(r)]$ rapidly decreases with increasing T. This is caused by depopulation of the surface through thermal desorption which compromises the Arrhenius analysis method because the EBID rate approaches zero as $N_a \to 0$. This effect contributes to the failure of the method at high temperatures seen in Figure 6.5(a). However, the failure at high temperatures is also contributed to by condition 4, which is discussed below.

A plot equivalent to Figure 6.7 can be generated using experimental EBID data, and can therefore be used to find the range of temperatures over which condition 3 is satisfied.

6.5.4 Condition 4 - Efficient adsorbate consumption

The driving force of diffusion $c(r)$ defines the net mass transport of adsorbates from the source to the sink shown in Figure 6.2(a). However, the in-diffusing adsorbates must not desorb (term N_a/τ in Eqn. 3.1), but must instead be dissociated by electrons (term $\sigma f N_a$) in order to be consumed in the EBID reaction and contribute to R_{VD}. Hence, if the condition $\sigma f \gg 1/\tau$ is not satisfied (i.e. if the residence time of adsorbates at the surface is too short because the temperature is too high), then both R_{VD} and C are reduced by an amount that scales with T due to the exponential dependence of $1/\tau$ on

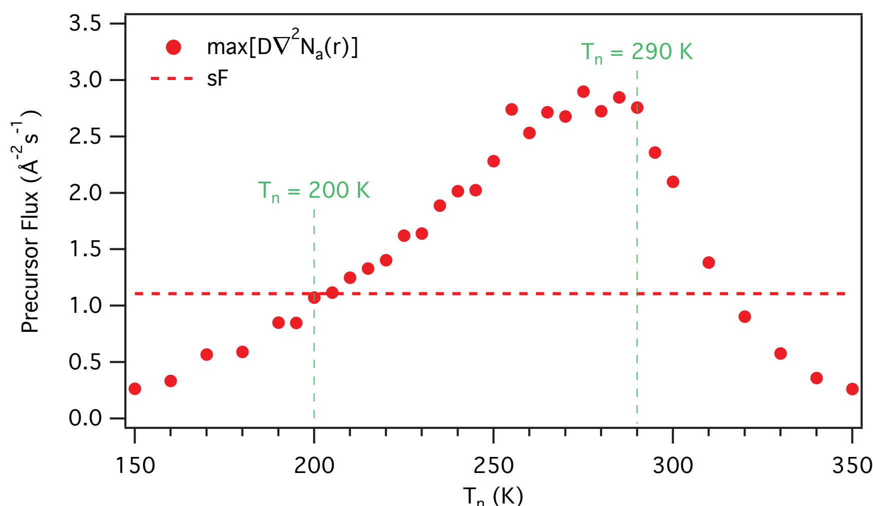

FIGURE 6.7: Maximum flux of diffusing adsorbates ($\max[D\nabla^2 N_a(r)]$) plotted at a number of temperatures T_n. The adsorption flux (sF) is shown as a dashed horizontal line.

T (Equation 2.3). This effect is seen in Figure 6.3(a-b), where it causes abrupt changes in the slopes of $C(T)$ and $R_{VD}(T)$ at temperatures greater than $\sim$ 275 K. Consequently, the Arrhenius analysis method fails at high temperatures (see Figure 6.5(a)) because the temperature-dependence of R_{VD}/C is not dominated by the temperature-dependence of D.

It is noted that the plot of R_{VD} on T_n in Figure 6.3(b) reveals that R_{VD} increases with T up to 275 K, beyond which it decays with increasing T. The general shape of this curve is the consequence of two competing exponential dependencies on temperature: the increase in D with T defined by Equation 2.5, and the decrease in τ with T defined by Equation 2.3. The rapid decay in $R_{VD}(T)$ at high temperatures is caused by violation of condition 4 (i.e. desorption of in-diffusing adsorbates) which becomes significant at 275 K, and is distinct from the thermal depopulation of the surface seen in Figure 6.7 which is insignificant below $\sim 290\ K$.

Experimentally, the maximum temperature at which condition 4 is satisfied can be found using plots of $C(T)$ and $R_{VD}(T)$ analogous to those shown in Figure 6.3.

6.6 Limitations of the Arrhenius analysis method

In the previous sections E_D and D_o can be found by Arrhenius analysis of nanostructure growth rates provided the analysis is performed over a specific temperature window. Within this window (shown in Figure 6.5(a)), the mean errors in E_D and D_o are 9% and 30%, respectively. The latter is greater because D_o is obtained from the y-intercepts of the Arrhenius plots shown in Figure 6.4, where the vertical axis is $ln(R_{VD}/C)$. The mean error in $ln(D_o)$ is smaller than 2%.

The valid temperature window is defined by the temperature-dependence of $ln(R_{VD}/C)$. Specifically, within the window, conditions 3 and 4 are satisfied and the temperature-dependence of $ln(R_{VD}/C)$ is dominated by that of D (given by Equation 2.5). Outside the window, the method fails because the temperature-dependence of $ln(R_{VD}/C)$ changes as condition 3 and/or 4 is violated. The method also fails if nanostructure growth rates are not measured in the steady state (condition 1), or the electron beam current density and hence the extent of adsorbate depletion is too low (condition 2). If all four conditions are satisfied, uncertainties in E_D and D_o are dominated by noise in $c(r)$ profiles (see Figure 6.2a) because of the ∇^2 operator in Equation 6.3. Hence, in practice, nanostructure geometries (i.e. growth rates) must be measured to a high degree of accuracy in order to minimize noise in $c(r)$.

The width of the valid temperature window is system specific (e.g. it is affected by the the corrugation factor[106] E_D/E_a), and can be maximized by tuning the precursor pressure and electron beam current density.

Finally, we note that deployment of the proposed method requires an experimental procedure for the attainment of $R_{VD}(T_n)$. The simplest procedure is to collect a set of $R(r, T_n)$ profiles such as that shown in Figure 6.1(b), subtract the lowest temperature profile $R(r, T_{min})$ from each $R(r, T_n)$ and integrate the resulting curves over r to get $R_{VD}(T_n)$. To demonstrate the validity of this approach we used our simulated $R(r, T_n)$ profiles to obtain $R_{VD}(T_n)$ using T_{min} values of 120 and 150 K. Figure 6.8 shows the resulting plots of activation energy and pre-exponential factor versus T_n. It shows that the analysis method works, but T_{min} affects the width of the valid temperature window (as expected), which is

characterized by a plateau in each curve. We note that the dependencies of the activation energy and pre-factor on T_n beyond the plateau depend on the method used to obtain R_{VD}, as is seen by the differences between the results in Figure 6.8 and 6.5(a). This is a consequence of the effects of T_{min} and violation of condition 4 on the diffusion-free component of the growth rate (produced by adsorption from the gas phase) at low and high T_n, respectively. The minimum temperature that can be used to perform conventional EBID (rather than cryogenic EBID [109, 110]) is limited by adsorbate condensation onto the substrate. Condensation (i.e. formation of multilayers rather than Langmuir adsorption defined by the term $(1-\Theta)$ in Equation 3.1) was ignored in our analysis (however, the precursor condensation temperature is easy to find and avoid experimentally [52, 109, 110]). More generally, the analysis presented here must be re-done for systems that do not exhibit Langmuir adsorption, and systems in which coverage-dependent phenomena such as adsorbate-adsorbate interactions are significant (i.e. such effects must be incorporated into Equation 3.1).

As a general guide to experimentalists the number of deposits required is quite large with the entire project estimated to take several weeks. This is due to the sheer number of deposits required to find the correct temperature range, electron beam current, and precursor gas pressure where diffusion is dominate. Once found the steady state growth rates also need to be determined for each of probed temperatures. Assuming best case conditions, using the temperature range shown here with one pillar per 10 K, with only 5 pillars to find steady state, that is 100 pillars, which depending on the growth rate of the precursor and the experimentalist the total time could be several days. In comparison the time required to simulate the electron flux profile of each pillars is insignificant of approximately 10 minutes per pillar due to the hybrid simulator's ability for custom surfaces to be input. The author through experience can imagine the several weeks mentioned earlier being the most likely time frame.

6.7 Conclusion

Using the hybrid Continuum-Monte Carlo model a method was developed which enables the calculation of diffusion energies and pre-exponential factors by Arrhenius analysis of

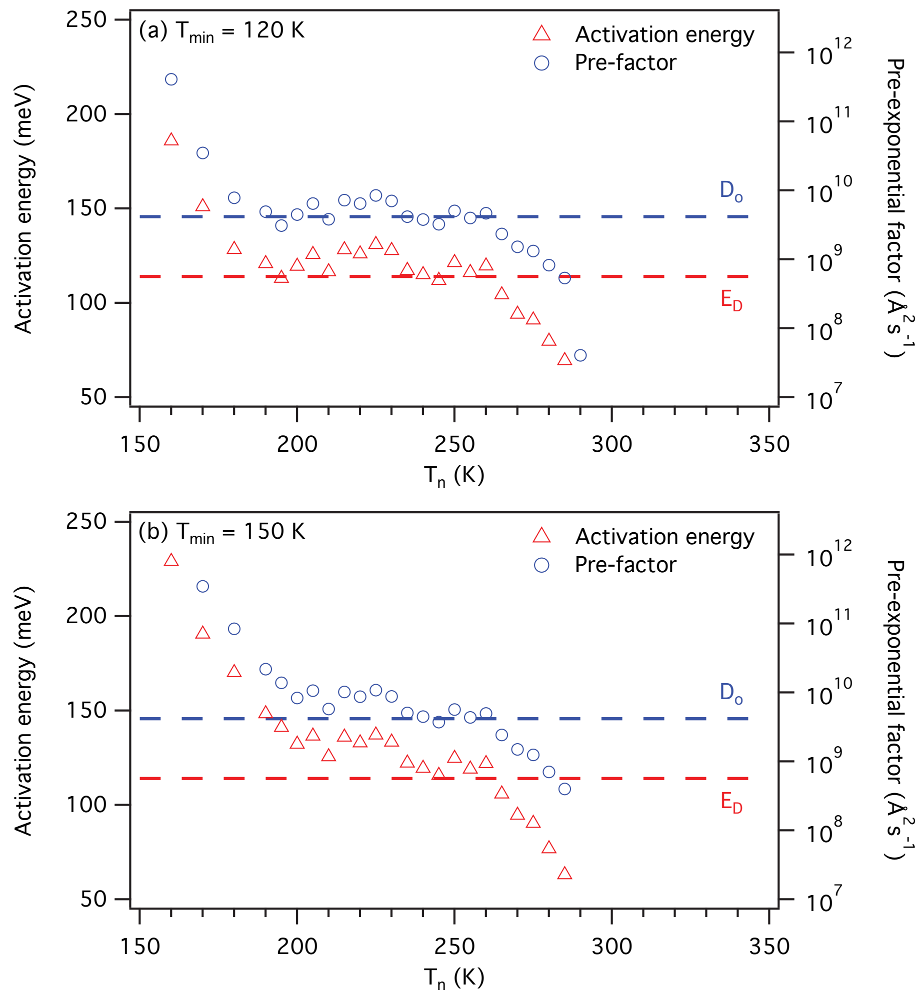

FIGURE 6.8: Activation energy (red) and pre-exponential factor (blue) obtained by Arrhenius analysis of R_{VD}/C at a number of temperatures T_n. The volume $R_{VD}(T_n)$ was estimated by subtracting (a) $R(r, T_{min} = 120 \text{ K})$ and (a) $R(r, T_{min} = 150 \text{ K})$ from each $R(r, T_n)$ profile, and integrating the resulting curves over r.

nanostructure deposition rates. The method is valid under specific growth conditions that were defined quantitatively. The results pave the way for experimental studies of adsorbate diffusion by EBID.

Chapter 7

Conclusion

A hybrid Continuum-Monte Carlo model was developed which simulates the growth of EBIED nano-structures over the length and time scales used in experiments. A Continuum EBIED model was used to simulate nanostructure growth, and a Monte Carlo electron scattering model to simulate electron trajectories where by using a hybrid approach the unique pitfalls of each modelling technique were overcome. Each model was extended beyond those available in literature to accommodate an evolving surface shape and the complex interaction of the electron beam with the surface. The resulting hybrid model was verified to show expected growth behaviour, in all relevant directions rather than a fixed direction, and the electron flux profile of primary, backscattered, and secondary electrons evolved with the growing structure.

The hybrid model was used to simulate the dependency EBID growth rates have on experimentally controlled growth parameters and how the relationship between them can be used to determine precursor specific coefficients. It was shown by performing an Arrhenius analysis of EBID growth rates under certain conditions activation energies and pre-exponential factors can be obtained and that these correspond to desorption and diffusion energies and attempt frequencies. Also, described were how deviations from these conditions effect the Arrhenius analysis and how these compare to previous observations seen in literature. The accuracy of the results presented opens the way for EBIED to be a complimentary characterisation technique to other conventional methods.

Appendix A

Diffusion Test Code

```
1  //
2  //  simulator.c
3  //
4  //  Created by Jared Cullen on 19/11/2012.
5  //  Last Updated by Jared Cullen on 3/12/2012.
6  //  Copyright (c) 2012 University of Technology, Sydney. All rights reserved.
7  //
8  //  Non-Uniform Grid Spacing Diffusion Test.
9
10 //Include Files.
11 #include <math.h>
12 #include <stdlib.h>
13 #include <stdio.h>
14 #include <time.h>
15
16 //Modules.
17 #include "./modules/solve_matrix.c"
18
19 //Main Program.
20 int main(int argc, char* argv[]) {
21     double specific_seed = 10.0;
22     struct timeval t1;
23     gettimeofday(&t1, NULL);
24     srand(specific_seed);
25     //srand(t1.tv_usec * t1.tv_sec);
26
27     //Variable Declarations.
28     int i,j;
29     int num_pts = 100;
30     int num_steps = 100;
31     double RHS[num_pts];
32     double answer[num_pts];
33     double LD[num_pts];
34     double CD[num_pts];
```

```
35      double UD[num_pts];
36      double conc[num_pts];
37      double r[num_pts];
38      double delta_t = 1.0E-9;
39      double delta_r = 1.0;
40      double delta_r_int = delta_r;
41      double diff_coeff = 1.0E8;
42      double asym_para;
43      double rand_num;
44      double area;
45      double sum = 0.0;
46      double pi = 3.141592653589793;
47      double temp_r;
48      double grad;
49      int uniform_switch = 0;
50
51      //Initial Calculations.
52      for (i=0;i<num_pts;i++) {
53          rand_num = rand();
54          conc[i] = 0.01;
55          if (uniform_switch == 1) {
56              r[i] = (double)(i+1)*delta_r;
57          } else {
58              r[i] = (double)(i+1)*delta_r+((rand_num/((double)RAND_MAX+1))-0.5)*
        delta_r*0.1;
59          }
60      }
61      conc[49] = 0.1;
62
63      printf("\n");
64      for (i=0;i<num_pts;i++) {
65          printf("%g\t%g\n",r[i],conc[i]);
66      }
67      printf("\n");
68
69      //Calculate Total Concentration.
70      j = 0;
71      area = pi*(pow(r[j]+(r[j+1]-r[j])/2.0,2.0)-pow(r[j]-delta_r/2.0,2.0));
72      sum += conc[j]/area;
73      j = num_pts-1;
74      sum = pi*(pow(r[j]+delta_r/2.0,2.0)-pow(r[j]-(r[j]-r[j-1])/2.0,2.0));
75      sum += conc[j]/area;
76      for (j=1;j<num_pts-1;j++) {
77          area = pi*(pow(r[j]+(r[j+1]-r[j])/2.0,2.0)-pow(r[j]-(r[j]-r[j-1])
        /2.0,2.0));
78          sum += conc[j]/area;
79      }
80      printf("%g\n",sum);
81      sum = 0.0;
82
83      //Solver
84      if (uniform_switch == 1) {
85          //Solver - Uniform
```

```
86          for (i=0;i<num_steps;i++) {
87              delta_r = r[1]-r[0];
88              RHS[0] = 0.0;
89              answer[0] = 1.0;
90              LD[0] = ((-1.0*diff_coeff)/(2.0*delta_r*delta_r))+(diff_coeff/(4.0*
    delta_r*delta_r));
91              CD[0] = 1.0;
92              UD[0] = -1.0;
93              delta_r = r[num_pts-1]-r[num_pts-2];
94              RHS[num_pts-1] = 0.0;
95              answer[num_pts-1] = 1.0;
96              LD[num_pts-1] = -1.0;
97              CD[num_pts-1] = 1.0;
98              UD[num_pts-1] = ((-1.0*diff_coeff)/(2.0*delta_r*delta_r))-(diff_coeff
    /(4.0*r[num_pts-1]*delta_r*delta_r));
99              for (j=1;j<(num_pts-1);j++) {
100                 delta_r = (r[j+1]-r[j-1])/2.0;
101                 RHS[j] = conc[j]*(1.0/delta_t-((2.0*diff_coeff)/(2.0*delta_r*
    delta_r)))+conc[j-1]*((diff_coeff/(2.0*delta_r*delta_r))-(diff_coeff/(4.0*r[
    num_pts-1]*delta_r*delta_r)))+conc[j+1]*((diff_coeff/(2.0*delta_r*delta_r))+(
    diff_coeff/(4.0*r[num_pts-1]*delta_r*delta_r)));
102                 answer[j] = 1.0;
103                 LD[j] = ((-1.0*diff_coeff)/(2.0*delta_r*delta_r))+(diff_coeff
    /(4.0*r[num_pts-1]*delta_r*delta_r));
104                 CD[j] = (1.0/delta_t)+((2.0*diff_coeff)/(2.0*delta_r*delta_r));
105                 UD[j] = ((-1.0*diff_coeff)/(2.0*delta_r*delta_r))-(diff_coeff
    /(4.0*r[num_pts-1]*delta_r*delta_r));
106             }
107             solve_matrix(num_pts,LD,CD,UD,RHS,answer);
108             for (j=0;j<num_pts;j++) {
109                 conc[j] = answer[j];
110             }
111             if (i%(num_steps/10) == 0) {
112                 printf("\n");
113                 for (j=0;j<num_pts;j++) {
114                     printf("%g\t%g\n",r[j],answer[j]);
115                 }
116                 printf("\n");
117             }
118
119             //Calculate Total Concentration.
120             j = 0;
121             area = pi*(pow(r[j]+(r[j+1]-r[j])/2.0,2.0)-pow(r[j]-delta_r/2.0,2.0))
    ;
122             sum += conc[j]/area;
123             j = num_pts-1;
124             sum = pi*(pow(r[j]+delta_r/2.0,2.0)-pow(r[j]-(r[j]-r[j-1])/2.0,2.0));
125             sum += conc[j]/area;
126             for (j=1;j<num_pts-1;j++) {
127                 area = pi*(pow(r[j]+(r[j+1]-r[j])/2.0,2.0)-pow(r[j]-(r[j]-r[j-1])
    /2.0,2.0));
128                 sum += conc[j]/area;
129             }
```

```
130             printf("%g\n",sum);
131             sum = 0.0;
132         }
133     } else {
134         //Solver - Non-Uniform
135         for (i=0;i<num_steps;i++) {
136             delta_r = r[1]-r[0];
137             asym_para = 0.0;
138             RHS[0] = 0.0;
139             answer[0] = 1.0;
140             LD[0] = (1.0/(1.0+asym_para))*((-1.0*diff_coeff)/(2.0*delta_r*delta_r
    ))+(diff_coeff/(4.0*delta_r*delta_r));
141             CD[0] = 1.0;
142             UD[0] = -1.0;
143             delta_r = r[num_pts-1]-r[num_pts-2];
144             asym_para = 0.0;
145             RHS[num_pts-1] = 0.0;
146             answer[num_pts-1] = 1.0;
147             LD[num_pts-1] = -1.0;
148             CD[num_pts-1] = 1.0;
149             UD[num_pts-1] = (1.0/(1.0-asym_para))*((-1.0*diff_coeff)/(2.0*delta_r
    *delta_r))-(diff_coeff/(4.0*r[num_pts-1]*delta_r*delta_r));
150             for (j=1;j<(num_pts-1);j++) {
151                 delta_r = (r[j+1]-r[j-1])/2.0;
152                 asym_para = (r[j]-r[j-1]-delta_r)/delta_r;
153                 RHS[j] = conc[j]*(1.0/delta_t-(2.0/(1.0-asym_para*asym_para))*((
    diff_coeff)/(2.0*delta_r*delta_r)))+conc[j-1]*(1.0/(1.0+asym_para))*((
    diff_coeff/(2.0*delta_r*delta_r))-(diff_coeff/(4.0*r[num_pts-1]*delta_r*
    delta_r)))+conc[j+1]*(1.0/(1.0-asym_para))*((diff_coeff/(2.0*delta_r*delta_r)
    )+(diff_coeff/(4.0*r[num_pts-1]*delta_r*delta_r)));
154                 answer[j] = 1.0;
155                 LD[j] = (1.0/(1.0+asym_para))*((-1.0*diff_coeff)/(2.0*delta_r*
    delta_r))+(diff_coeff/(4.0*r[num_pts-1]*delta_r*delta_r));
156                 CD[j] = (1.0/delta_t)+(2.0/(1.0-asym_para*asym_para))*((
    diff_coeff)/(2.0*delta_r*delta_r));
157                 UD[j] = (1.0/(1.0-asym_para))*((-1.0*diff_coeff)/(2.0*delta_r*
    delta_r))-(diff_coeff/(4.0*r[num_pts-1]*delta_r*delta_r));
158             }
159             solve_matrix(num_pts,LD,CD,UD,RHS,answer);
160             for (j=0;j<num_pts;j++) {
161                 conc[j] = answer[j];
162             }
163             if (i%(num_steps/10) == 0) {
164                 printf("\n");
165                 for (j=0;j<num_pts;j++) {
166                     printf("%g\t%g\n",r[j],answer[j]);
167                 }
168                 printf("\n");
169             }
170             //Calculate Total Concentration.
171             j = 0;
172             area = pi*(pow(r[j]+(r[j+1]-r[j])/2.0,2.0)-pow(r[j]-delta_r/2.0,2.0))
    ;
```

```
173             sum += conc[j]/area;
174             j = num_pts-1;
175             sum = pi*(pow(r[j]+delta_r/2.0,2.0)-pow(r[j]-(r[j]-r[j-1])/2.0,2.0));
176             sum += conc[j]/area;
177             for (j=1;j<num_pts-1;j++) {
178                 area = pi*(pow(r[j]+(r[j+1]-r[j])/2.0,2.0)-pow(r[j]-(r[j]-r[j-1])
    /2.0,2.0));
179                 sum += conc[j]/area;
180             }
181             printf("%g\n",sum);
182             sum = 0.0;
183             //Calculate new r values and then modify the conc values by the
    appropiate gradient.
184             for (j=1;j<(num_pts-1);j++) {
185                 rand_num = rand();
186                 temp_r = (double)(j+1)*delta_r_int+((rand_num/((double)RAND_MAX
    +1))-0.5)*delta_r_int*0.1;
187                 grad = (conc[j+1]-conc[j-1])/(r[j+1]-r[j-1]);
188                 conc[j] = conc[j]+(grad*(temp_r-r[j]));
189                 r[j] = temp_r;
190             }
191         }
192     }
193 }
```

Appendix B

Hybrid Continuum-Monte Carlo Simulator Code

The following code is for the Hybrid Continuum-Monte Carlo Simulator code. Modules from Numerical Recipes in C[111] are not reproduced here.

B.1 Constants

```
//
//  constants.h
//
//  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
//
//  This file contains all the constants required for the EBIED simulator.

#ifndef _constants_h
#define _constants_h
#endif

#define PI 3.141592653589793
#define BOLTZMANN_CONSTANT 1.38066E-23 //J/K
#define ELECTRON_CHARGE 1.60217657E-19 //coulombs
```

B.2 Variable Structures

```
//
//  constants.h
```

```
3  //
4  //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5  //
6  //  This file contains all the structures required for the EBIED simulator.
7
8  #ifndef _structures_h
9  #define _structures_h
10 #endif
11
12 struct precursor {
13     double reactive_product_flux;   // in A-2s-1.
14     double initial_gas_coverage;    //concentration
15     double gas_partial_pressure;    // in Pa.
16     double reactive_product_molecular_mass; // in amu*1.66053873E-27 to give kg.
17     double reactive_product_density;// in kg/A3.
18     double surface_area;            // in angstrom^2.
19     double desorption_time;         // in sec.
20     double desorption_energy;       // in eV.
21     double desorption_attempt_frequency; //in s.
22     double diffusion_coefficient;   // in angstrom^2/s.
23     double diffusion_energy;        // in eV.
24     double diffusion_attempt_frequency; // in angstrom^2/s.
25     double sticking_coefficient;    // (dimensionless)
26     double PE_electron_cross_section;  // in angstrom^2.
27     double BSE_electron_cross_section;  // in angstrom^2.
28     double SE_electron_cross_section;  // in angstrom^2.
29 };
30 typedef struct precursor Precursor;
31
32 struct toggle {
33     int electron_trajectory_simulator;  // MC electron trajectories, On(1) or Off
       (0).
34     int electron_trajectory_tracking;   // track electron trajectories, On(1) or
       Off(0).
35     int electron_beam_projection;       // electron beam surface projection, On
       (1) or Off(0).
36     int electron_beam_shape;            // top hat profile(1) or gaussian profile
       (0).
37     int surface_evolution;              // surface evolution, normal to the
       surface(1) or standard(0).
38     int previous_simulation;            // load the output data from a previous
       simulation.
39     int diffusion;                      // surface diffusion, on(1) or off(0). //
        FIXME: diffusion itself can be set to zero ???
40     int coverage;                       // initial surface coverage, monolayer(1)
        or no coverage (0).
41 };
42 typedef struct toggle Toggle;
43
44 struct material {
45     double atomic_number; //material atomic number.
46     double atomic_weight; //material atomic weight.
47     double density;       //material density, in g/cm3.
```

```
48      double epsilon;        //effective energy required to produce an SE, in keV.
49      double lambda;         //effective SE escape depth, in A.
50  };
51  typedef struct material Material;
52
53  struct electron_beam {
54      double cutoff_energy;      //cutoff energy in keV in bulk case.
55      double top_hat_abruptness; //abruptness of top hat electron beam profile.
56      double diameter;           //electron beam diameter in A.
57      double energy;             //input electron beam energy in keV.
58      double current;            //electron beam current, in electron/sec.
59      double tilt;               //electron beam tilt, in degrees.
60  };
61  typedef struct electron_beam Electron_beam;
```

B.3 Function Prototypes

```
1   //
2   //  prototypes.h
3   //
4   //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5   //
6   //  This file contains all the prototypes required for the EBIED simulator.
7
8   #ifndef _prototypes_h
9   #define _prototypes_h
10  #endif
11
12  #define MPI_ON
13
14  //Malloc Arrays
15  extern double *r_coordinate_array;
16  extern double *z_coordinate_array;
17  extern double *electron_flux_profile_array;
18  extern double *electron_starting_location_probability_array;
19  extern double *backscattered_electron_location_array;
20  extern double *secondary_electron_location_array;
21  extern double *etch_precursor_gas_concentration_previous_two_time_step_array;
22  extern double *etch_precursor_gas_concentration_previous_time_step_array;
23  extern double *etch_precursor_gas_concentration_current_time_step_array;
24  extern double *deposit_precursor_gas_concentration_previous_two_time_step_array;
25  extern double *deposit_precursor_gas_concentration_previous_time_step_array;
26  extern double *deposit_precursor_gas_concentration_current_time_step_array;
27  extern double *reactive_product_concentration_previous_time_step_array;
28  extern double *reactive_product_concentration_current_time_step_array;
29  extern double *growth_or_etch_rate_array;
30  extern double *electron_energy_deposited_array;
31  extern double *electron_maximum_z_depth_array;
32  extern double *r_coordinate_secondary_array_one;
33  extern double *r_coordinate_secondary_array_two;
```

```
34 extern double *surface_line_equation_array_one;
35 extern double *surface_line_equation_array_two;
36 extern double *surface_line_equation_array_three;
37 extern double *temporary_z_gradient;
38 extern double *previous_r_coordinate_array;
39 extern double *previous_z_coordinate_array;
40 extern double *
       previous_deposit_precursor_gas_concentration_previous_time_step_array;
41 extern double *previous_etch_precursor_gas_concentration_previous_time_step_array
       ;
42 extern double *previous_reactive_product_concentration_previous_time_step_array;
43 extern double *right_hand_side;
44 extern double *answer;
45 extern double *answer_temp;
46 extern double *lower_diagonal;
47 extern double *central_diagonal;
48 extern double *upper_diagonal;
49 extern double *r_array;
50 extern double *primary_electron_array;
51 extern double *backscattered_electron_array;
52 extern double *secondary_electron_array;
53 extern double *temp_primary_electron_array;
54 extern double *temp_primary_electron_flux_profile_array;
55 extern double *temp_electron_starting_location_probability_array;
56 extern double *local_secondary_electron_location_array;
57 extern double *sum_secondary_electron_location_array;
58 extern double *local_backscattered_electron_location_array;
59 extern double *sum_backscattered_electron_location_array;
60 extern double *local_electron_energy_deposited_array;
61 extern double *sum_electron_energy_deposited_array;
62 extern double *local_electron_maximum_z_depth_array;
63 extern double *sum_electron_maximum_z_depth_array;
64 extern double *electron_tracking_x_position_array;
65 extern double *electron_tracking_y_position_array;
66 extern double *electron_tracking_z_position_array;
67 extern double *r_coordinate_both_directions;
68 extern double *z_coordinate_both_directions;
69 extern double *temporary_r_coordinate_secondary_array_one;
70 extern double *temporary_r_coordinate_secondary_array_two;
71 extern double *temporary_surface_line_equation_array_one;
72 extern double *temporary_surface_line_equation_array_two;
73 extern double *temporary_surface_line_equation_array_three;
74 extern double *dC;
75 extern double *temporary_z_coordinate_array;
76 extern double *temporary_r_coordinate_array;
77 extern double *temporary_reactive_product_concentration_current_time_step_array_y
       ;
78 extern double *temporary_reactive_product_concentration_current_time_step_array_x
       ;
79 extern double *
       temporary_etch_precursor_gas_concentration_current_time_step_array_y;
80 extern double *
       temporary_etch_precursor_gas_concentration_current_time_step_array_x;
```

```
81 extern double *
       temporary_deposit_precursor_gas_concentration_current_time_step_array_y;
82 extern double *
       temporary_deposit_precursor_gas_concentration_current_time_step_array_x;
83 extern double *PE_electron_flux_profile_array;
84 extern double *BSE_electron_flux_profile_array;
85 extern double *SE_electron_flux_profile_array;
86 extern double **print_electron_flux_profile_array;
87 extern double **print_N_little_e;
88 extern double **print_N_little_d;
89 extern double **print_N_big_d;
90 extern double **print_growth_rate;
91 extern double **print_surface;
92 extern double **print_secondary_electrons;
93 extern double **print_backscattered_electrons;
94 extern double *temporary_ND_array;
95 extern double *time_per_time_step_array;
96
97 //Functions
98 void malloc_memory(int number_of_surface_bins,int
       length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,int
       maximum_electron_track_length,int number_of_simulation_time_steps);
99 void free_memory();
100 void *safe_malloc(int n_bytes);
101 void safe_free(double **ptr);
102 void logfile_printf(const char *fmt, ...);
103 void ludcmp(float **a, int n, int *indx, float *d, int *error_flag);
104 void lubksb(float **a, int n, int *indx, float b[]);
105
106 void ebied_solver(Precursor etch_precursor,Precursor deposit_precursor,Toggle
       toggle,double gas_temperature,double
       deposit_pinned_reaction_electron_cross_section,double
       deposit_precursor_reaction_electron_cross_section,double delta_t,double
       delta_r,int number_of_surface_bins,int no_etch_area);
107
108 void ebied_solver_no_diffusion(Precursor etch_precursor,Precursor
       deposit_precursor,Toggle toggle,double gas_temperature,double
       deposit_pinned_reaction_electron_cross_section,double
       deposit_precursor_reaction_electron_cross_section,double delta_t,double
       delta_r,int number_of_surface_bins);
109
110 void ebid_solver(Precursor deposit_precursor,Toggle toggle,double gas_temperature
       ,double delta_t,double delta_r,int number_of_surface_bins);
111
112 void electron_flux_profile(Toggle toggle,Electron_beam electron_beam,int
       number_of_surface_bins,double delta_t,double delta_r,double
       backscattered_electron_coefficient,double secondary_electron_coefficient,int
       i);
113
114 void primary_interpolation(double *r[],double *z[],int length,double delta_r,int
       number_of_points_original);
115
```

```
116 void secondary_interpolation(double r[],double *z[],double z_orig[],double
        r_interp[],int length,int number_of_points,int order,double delta_r);
117
118 void polynomial_fit(double (*coeff)[],double *gradient,double x_data[],double
        y_data[],int order,int number_of_points,int *error_flag);
119
120 void matrix_inversion(int N,float matrix_data[N][N],float (*y)[N][N],int *
        error_flag);
121
122 void matrix_multiplication(int A_r,int A_c,int B_r,int B_c,float matrix_A[A_r][
        A_c],float matrix_B[B_r][B_c],float (*matrix_C)[A_r][B_c]);
123
124 void monte_carlo_data_collection(int
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,int
        number_of_electron_trajectories,int number_of_surface_bins,int
        number_of_backscattered_electrons,double *backscattered_electron_coefficient,
        double *secondary_electron_coefficient);
125
126 void monte_carlo_electron_trajectory_simulator(Electron_beam electron_beam,
        Material lower_material,Material upper_material,int
        electron_trajectory_tracking,double delta_r,int
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,int
        number_of_surface_bins,double layered_material_interface_depth,int
        maximum_electron_track_length,int number_of_electron_trajectories,double
        z_depth_minimum,double z_depth_maximum,int *number_of_backscattered_electrons
        ,int seed);
127
128 void get_constants(double *sg_a,double *al_a,double *lam_a,double at_num,double
        at_wht,double density,double inc_energy);
129
130 void reset_coordinates(double *s_en,double *x,double *y,double *z,double *cx,
        double *cy,double *cz,double inc_energy,int surface_length);
131
132 void s_scatter(double energy,double al_a,double *sp,double *ga,double *cp);
133
134 void new_coord(double step,double x,double y,double z,double cx,double cy,double
        cz,double sp,double ga,double cp,double *ca,double *cb,double *cc,double *xn,
        double *yn,double *zn);
135
136 void reset_next_step(double ca,double cb,double cc,double xn,double yn,double zn,
        double step,double *cx,double *cy,double *cz,double *x,double *y,double *z,
        double *s_en,double density,double at_num,double at_wht,double SE_epsilon,
        double SE_lambda,int number_of_intersections,double exit_x1,double exit_y1,
        double exit_x2,double exit_y2,double e_length_d,double delta_r,int
        number_of_surface_bins,int intersection_index1);
137
138 void monte_carlo_surface_setup(int number_of_surface_bins,double *z_depth_minimum
        ,double *z_depth_maximum);
139
140 void set_default_parameters(int *number_of_electron_trajectories,
141                             int *number_of_surface_bins,
142                             int *number_of_simulation_time_steps,
143                             int *run_MC_every_zero_point_X_percent,
```

```
144                             int *save_simulation_data_every_X_percent,
145                             int *
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
146                             int *maximum_electron_track_length,
147                             double *delta_r,
148                             double *delta_t,
149                             Electron_beam *electron_beam,
150                             double *layered_material_interface_depth,
151                             Material *upper_material,
152                             Material *lower_material,
153                             double *gas_temperature,
154                             Precursor *etch_precursor,
155                             Precursor *deposit_precursor,
156                             double *
    deposit_pinned_reaction_electron_cross_section,
157                             double *
    deposit_precursor_reaction_electron_cross_section,
158                             Toggle *toggle,
159                             int *number_of_points,
160                             int *order,
161                             int *no_etch_order,
162                             int *seed,
163                             int *wave_type,
164                             int *square_wave_period,
165                             int *square_wave_min_current,
166                             int *square_wave_max_current,
167                             double *triangle_wave_rate,
168                             int *triangle_wave_min_current,
169                             int *triangle_wave_max_current,
170                             int *pulsing_period,
171                             int *pulse_on_time,
172                             int *pulse_off_time,
173                             int *time_delay,
174                             double *precursor_diffusion_tolerance,
175                             double *substrate_temperature);
176
177 void read_input_parameters(int *number_of_electron_trajectories,int *
    number_of_surface_bins,int *number_of_simulation_time_steps,int *
    run_MC_every_zero_point_X_percent,int *save_simulation_data_every_X_percent,
    int *length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    int *maximum_electron_track_length,double *delta_r,double *delta_t,
    Electron_beam *electron_beam,double *layered_material_interface_depth,
    Material *upper_material,Material *lower_material,double *gas_temperature,
    Precursor *etch_precursor,Precursor *deposit_precursor,double *
    deposit_pinned_reaction_electron_cross_section,double *
    deposit_precursor_reaction_electron_cross_section,Toggle *toggle,int *
    number_of_points,int *order,int *no_etch_area,int *seed,int *wave_type,int *
    square_wave_period,int *square_wave_min_current,int *square_wave_max_current,
    double *triangle_wave_rate,int *triangle_wave_min_current,int *
    triangle_wave_max_current,int *pulsing_period,int *pulse_on_time,int *
    pulse_off_time,int *time_delay,double *precursor_diffusion_tolerance,double *
    substrate_temperature);
178
```

```
179 void read_input_previous_simulation(int number_of_surface_bins);
180
181 void save_current_simulation_data(int
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,double
         backscattered_electron_coefficient,double secondary_electron_coefficient,int
         number_of_surface_bins,int number_of_electron_trajectories,int
        current_time_step,int val1,int val2,int number_of_simulation_time_steps,int
        save_simulation_data_every_X_percent);
182
183 void setup_save_files();
184
185 void solve_matrix(int n, double *a, double *b, double *c, double *v, double *x);
186
187 void surface_evolution(Precursor deposit_precursor,Precursor etch_precursor,
        Material lower_material,
188                        Material upper_material,int evolve_surface_normal,int
        number_of_points,int order,int number_of_surface_bins,double delta_t,double
        delta_r,double layered_material_interface_depth,int time_step,Toggle toggle);
189
190 void Free2DDoubleArray(double **theArray);
191
192 int are_all_areas_equal(double r[],double z[],int length,double delta_r);
193
194 int sgn(double x);
195
196 double compute_lambda(double energy,double al_a,double sg_a,double lam_a);
197
198 double mean_ionisation_potential(double at_num);
199
200 double stop_pwr(double energy,double at_num,double at_wht);
201
202 double** Make2DDoubleArray(int arraySizeX, int arraySizeY);
203
204 double linear_interp(double x[],double y[],int n,double xi);
205
206 int gsl_fit_linear(const double * x, const size_t xstride, const double * y,
        const size_t ystride, size_t n, double * c0, double * c1, double * cov00,
        double * cov01, double * cov11, double * sumsq);
```

B.4 Simulator Core

```
1 //
2 //  simulator.c
3 //
4 //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5 //
6 //  This is the central program for the EBIED Simulator.
7 //
8
9 #ifdef XCODE
```

```
10      //if compliling with xcode include these files
11      #include <math.h>
12      #include <stdlib.h>
13      #include <stdio.h>
14      #include <time.h>
15      #include <assert.h>
16      #include <sys/stat.h>
17      #include "./modules/constants.h"
18      #include "./modules/structures.h"
19      #include "./modules/prototypes.h"
20      #ifdef MPI_ON
21          //if also using openmpi include it too
22          #include <mpi.h>
23      #endif
24  #else
25      //if compliling with anything else e.g. gcc include all files
26      #include <math.h>
27      #include <stdlib.h>
28      #include <stdio.h>
29      #include <stdarg.h>
30      #include <time.h>
31      #include <assert.h>
32      #include <sys/stat.h>
33      #include <sys/time.h>
34      #include <string.h>
35      #include <gsl/gsl_fit.h>
36      #include "./modules/constants.h"
37      #include "./modules/structures.h"
38      #include "./modules/prototypes.h"
39      #ifdef MPI_ON
40          //if also using openmpi include it too
41          #include <mpi.h>
42      #endif
43      #include "./modules/solve_matrix.c"
44      #include "./modules/setup_save_files.c"
45      #include "./modules/save_current_simulation_data.c"
46      #include "./modules/ebie_ebid_solver.c"
47      #include "./modules/monte_carlo_surface_setup.c"
48      #include "./modules/monte_carlo_data_collection.c"
49      #include "./modules/monte_carlo_electron_trajectory_simulator.c"
50      #include "./modules/electron_flux_profile.c"
51      #include "./modules/surface_evolution.c"
52      #include "./modules/read_input_parameters.c"
53      #include "./modules/read_input_previous_simulation.c"
54      #include "./modules/interpolation.c"
55      #include "./modules/logfile_printf.c"
56  #endif
57
58  //Global Malloc Arrays
59  double *r_coordinate_array = NULL;
60  double *z_coordinate_array = NULL;
61  double *electron_flux_profile_array = NULL;
62  double *electron_starting_location_probability_array = NULL;
```

```
63 double *backscattered_electron_location_array = NULL;
64 double *secondary_electron_location_array = NULL;
65 double *etch_precursor_gas_concentration_previous_two_time_step_array = NULL;
66 double *etch_precursor_gas_concentration_previous_time_step_array = NULL;
67 double *etch_precursor_gas_concentration_current_time_step_array = NULL;
68 double *deposit_precursor_gas_concentration_previous_two_time_step_array = NULL;
69 double *deposit_precursor_gas_concentration_previous_time_step_array = NULL;
70 double *deposit_precursor_gas_concentration_current_time_step_array = NULL;
71 double *reactive_product_concentration_previous_time_step_array = NULL;
72 double *reactive_product_concentration_current_time_step_array = NULL;
73 double *growth_or_etch_rate_array = NULL;
74 double *electron_energy_deposited_array = NULL;
75 double *electron_maximum_z_depth_array = NULL;
76 double *r_coordinate_secondary_array_one = NULL;
77 double *r_coordinate_secondary_array_two = NULL;
78 double *surface_line_equation_array_one = NULL;
79 double *surface_line_equation_array_two = NULL;
80 double *surface_line_equation_array_three = NULL;
81 double *temporary_z_gradient = NULL;
82 double *previous_r_coordinate_array = NULL;
83 double *previous_z_coordinate_array = NULL;
84 double *previous_deposit_precursor_gas_concentration_previous_time_step_array =
       NULL;
85 double *previous_etch_precursor_gas_concentration_previous_time_step_array = NULL
       ;
86 double *previous_reactive_product_concentration_previous_time_step_array = NULL;
87 double *right_hand_side = NULL;
88 double *answer = NULL;
89 double *answer_temp = NULL;
90 double *lower_diagonal = NULL;
91 double *central_diagonal = NULL;
92 double *upper_diagonal = NULL;
93 double *r_array = NULL;
94 double *primary_electron_array = NULL;
95 double *backscattered_electron_array = NULL;
96 double *secondary_electron_array = NULL;
97 double *temp_primary_electron_array = NULL;
98 double *temp_primary_electron_flux_profile_array = NULL;
99 double *temp_electron_starting_location_probability_array = NULL;
100 double *local_secondary_electron_location_array = NULL;
101 double *sum_secondary_electron_location_array = NULL;
102 double *local_backscattered_electron_location_array = NULL;
103 double *sum_backscattered_electron_location_array = NULL;
104 double *local_electron_energy_deposited_array = NULL;
105 double *sum_electron_energy_deposited_array = NULL;
106 double *local_electron_maximum_z_depth_array = NULL;
107 double *sum_electron_maximum_z_depth_array = NULL;
108 double *electron_tracking_x_position_array = NULL;
109 double *electron_tracking_y_position_array = NULL;
110 double *electron_tracking_z_position_array = NULL;
111 double *r_coordinate_both_directions = NULL;
112 double *z_coordinate_both_directions = NULL;
113 double *temporary_r_coordinate_secondary_array_one = NULL;
```

```
114 double *temporary_r_coordinate_secondary_array_two = NULL;
115 double *temporary_surface_line_equation_array_one = NULL;
116 double *temporary_surface_line_equation_array_two = NULL;
117 double *temporary_surface_line_equation_array_three = NULL;
118 double *dC = NULL;
119 double *temporary_z_coordinate_array = NULL;
120 double *temporary_r_coordinate_array = NULL;
121 double *temporary_reactive_product_concentration_current_time_step_array_y = NULL
    ;
122 double *temporary_reactive_product_concentration_current_time_step_array_x = NULL
    ;
123 double *temporary_etch_precursor_gas_concentration_current_time_step_array_y =
    NULL;
124 double *temporary_etch_precursor_gas_concentration_current_time_step_array_x =
    NULL;
125 double *temporary_deposit_precursor_gas_concentration_current_time_step_array_y =
     NULL;
126 double *temporary_deposit_precursor_gas_concentration_current_time_step_array_x =
     NULL;
127 double *PE_electron_flux_profile_array = NULL;
128 double *BSE_electron_flux_profile_array = NULL;
129 double *SE_electron_flux_profile_array = NULL;
130 double **print_electron_flux_profile_array = NULL;
131 double **print_N_little_e = NULL;
132 double **print_N_little_d = NULL;
133 double **print_N_big_d = NULL;
134 double **print_growth_rate = NULL;
135 double **print_surface = NULL;
136 double **print_secondary_electrons = NULL;
137 double **print_backscattered_electrons = NULL;
138 double *temporary_ND_array = NULL;
139 double *time_per_time_step_array = NULL;
140
141 //standard linear interpolation method
142 double linear_interp(double x[],double y[],int n,double xi) {
143     double yi = 0.0;
144     int i;
145     for (i=0;i<n-1;i++) {
146         if ((xi > x[i] && xi < x[i+1]) || (xi == x[i]) || (xi == x[i+1])) {
147             yi = y[i]+(y[i+1]-y[i])*((xi-x[i])/(x[i+1]-x[i]));
148             return yi;
149         }
150     }
151     return yi;
152 }
153
154 //gets the sign (positive/negative) of a number
155 int sgn(double x) {
156    if (x < 0.0) {
157       return -1.0;
158    } else {
159       return 1.0;
160    }
```

```
161 }
162 
163 //malloc memory for a two dimentional array
164 double** Make2DDoubleArray(int arraySizeX, int arraySizeY) {
165     double** theArray;
166     int i;
167     theArray = (double**) malloc(arraySizeX*sizeof(double*));
168     for (i = 0; i < arraySizeX; i++) {
169         theArray[i] = (double*) malloc(arraySizeY*sizeof(double));
170     }
171     return theArray;
172 }
173 
174 //free malloc'ed memory for a two dimentional array
175 void Free2DDoubleArray(double **theArray) {
176     free(*theArray);
177     free(theArray);
178     theArray = NULL;
179 }
180 
181 //malloc memory and check if sucessful
182 void *safe_malloc(int n_bytes) {
183     void *ptr = malloc(n_bytes);
184     assert(ptr != NULL);
185     return ptr;
186 }
187 
188 //free malloc'ed memory
189 void safe_free(double **ptr) {
190     free(*ptr);
191     *ptr = NULL;
192 }
193 
194 //giant function to make all memory needed for simulator arrays
195 void malloc_memory(int number_of_surface_bins,int
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,int
    maximum_electron_track_length,int number_of_simulation_time_steps) {
196     //Declarations
197     int N = number_of_surface_bins*sizeof(double);
198     int N2 =
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array*sizeof
    (double);
199     int N3 = maximum_electron_track_length*sizeof(double);
200     int N4 = number_of_simulation_time_steps*sizeof(double);
201     r_coordinate_array = safe_malloc(N);
202     z_coordinate_array = safe_malloc(N);
203     electron_flux_profile_array = safe_malloc(N);
204     electron_starting_location_probability_array = safe_malloc(N);
205     backscattered_electron_location_array = safe_malloc(N);
206     secondary_electron_location_array = safe_malloc(N);
207     etch_precursor_gas_concentration_previous_two_time_step_array = safe_malloc(N
    );
208     etch_precursor_gas_concentration_previous_time_step_array = safe_malloc(N);
```

```
209     etch_precursor_gas_concentration_current_time_step_array = safe_malloc(N);
210     deposit_precursor_gas_concentration_previous_two_time_step_array =
        safe_malloc(N);
211     deposit_precursor_gas_concentration_previous_time_step_array = safe_malloc(N)
        ;
212     deposit_precursor_gas_concentration_current_time_step_array = safe_malloc(N);
213     reactive_product_concentration_previous_time_step_array = safe_malloc(N);
214     reactive_product_concentration_current_time_step_array = safe_malloc(N);
215     growth_or_etch_rate_array = safe_malloc(N);
216     electron_energy_deposited_array = safe_malloc(N2);
217     electron_maximum_z_depth_array = safe_malloc(N2);
218     r_coordinate_secondary_array_one = safe_malloc(2*N);
219     r_coordinate_secondary_array_two = safe_malloc(2*N);
220     surface_line_equation_array_one = safe_malloc(2*N);
221     surface_line_equation_array_two = safe_malloc(2*N);
222     surface_line_equation_array_three = safe_malloc(2*N);
223     temporary_z_gradient = safe_malloc(N);
224     previous_r_coordinate_array = safe_malloc(N);
225     previous_z_coordinate_array = safe_malloc(N);
226     previous_deposit_precursor_gas_concentration_previous_time_step_array =
        safe_malloc(N);
227     previous_etch_precursor_gas_concentration_previous_time_step_array =
        safe_malloc(N);
228     previous_reactive_product_concentration_previous_time_step_array =
        safe_malloc(N);
229     right_hand_side = safe_malloc(N);
230     answer = safe_malloc(N);
231     answer_temp = safe_malloc(N);
232     lower_diagonal = safe_malloc(N);
233     central_diagonal = safe_malloc(N);
234     upper_diagonal = safe_malloc(N);
235     r_array = safe_malloc(N);
236     primary_electron_array = safe_malloc(N);
237     backscattered_electron_array = safe_malloc(N);
238     secondary_electron_array = safe_malloc(N);
239     temp_primary_electron_array = safe_malloc(N);
240     temp_primary_electron_flux_profile_array = safe_malloc(N);
241     temp_electron_starting_location_probability_array = safe_malloc(N);
242     local_secondary_electron_location_array = safe_malloc(N);
243     sum_secondary_electron_location_array = safe_malloc(N);
244     local_backscattered_electron_location_array = safe_malloc(N);
245     sum_backscattered_electron_location_array = safe_malloc(N);
246     local_electron_energy_deposited_array = safe_malloc(N2);
247     sum_electron_energy_deposited_array = safe_malloc(N2);
248     local_electron_maximum_z_depth_array = safe_malloc(N2);
249     sum_electron_maximum_z_depth_array = safe_malloc(N2);
250     electron_tracking_x_position_array = safe_malloc(N3);
251     electron_tracking_y_position_array = safe_malloc(N3);
252     electron_tracking_z_position_array = safe_malloc(N3);
253     r_coordinate_both_directions = safe_malloc(2*N);
254     z_coordinate_both_directions = safe_malloc(2*N);
255     temporary_r_coordinate_secondary_array_one = safe_malloc(2*N);
256     temporary_r_coordinate_secondary_array_two = safe_malloc(2*N);
```

```
257     temporary_surface_line_equation_array_one = safe_malloc(2*N);
258     temporary_surface_line_equation_array_two = safe_malloc(2*N);
259     temporary_surface_line_equation_array_three = safe_malloc(2*N);
260     dC = safe_malloc(N);
261     temporary_z_coordinate_array = safe_malloc(N);
262     temporary_r_coordinate_array = safe_malloc(N);
263     temporary_reactive_product_concentration_current_time_step_array_y =
        safe_malloc(N);
264     temporary_reactive_product_concentration_current_time_step_array_x =
        safe_malloc(N);
265     temporary_etch_precursor_gas_concentration_current_time_step_array_y =
        safe_malloc(N);
266     temporary_etch_precursor_gas_concentration_current_time_step_array_x =
        safe_malloc(N);
267     temporary_deposit_precursor_gas_concentration_current_time_step_array_y =
        safe_malloc(N);
268     temporary_deposit_precursor_gas_concentration_current_time_step_array_x =
        safe_malloc(N);
269     PE_electron_flux_profile_array = safe_malloc(N);
270     BSE_electron_flux_profile_array = safe_malloc(N);
271     SE_electron_flux_profile_array = safe_malloc(N);
272     temporary_ND_array = safe_malloc(N);
273     time_per_time_step_array = safe_malloc(N4);
274 }
275 
276 //giant function to free all malloc'ed memory
277 void free_memory() {
278     safe_free(&r_coordinate_array);
279     safe_free(&z_coordinate_array);
280     safe_free(&electron_flux_profile_array);
281     safe_free(&electron_starting_location_probability_array);
282     safe_free(&backscattered_electron_location_array);
283     safe_free(&secondary_electron_location_array);
284     safe_free(&etch_precursor_gas_concentration_previous_two_time_step_array);
285     safe_free(&etch_precursor_gas_concentration_previous_time_step_array);
286     safe_free(&etch_precursor_gas_concentration_current_time_step_array);
287     safe_free(&deposit_precursor_gas_concentration_previous_two_time_step_array);
288     safe_free(&deposit_precursor_gas_concentration_previous_time_step_array);
289     safe_free(&deposit_precursor_gas_concentration_current_time_step_array);
290     safe_free(&reactive_product_concentration_previous_time_step_array);
291     safe_free(&reactive_product_concentration_current_time_step_array);
292     safe_free(&growth_or_etch_rate_array);
293     safe_free(&electron_energy_deposited_array);
294     safe_free(&electron_maximum_z_depth_array);
295     safe_free(&r_coordinate_secondary_array_one);
296     safe_free(&r_coordinate_secondary_array_two);
297     safe_free(&surface_line_equation_array_one);
298     safe_free(&surface_line_equation_array_two);
299     safe_free(&surface_line_equation_array_three);
300     safe_free(&temporary_z_gradient);
301     safe_free(&previous_r_coordinate_array);
302     safe_free(&previous_z_coordinate_array);
```

```
303     safe_free(&
        previous_deposit_precursor_gas_concentration_previous_time_step_array);
304     safe_free(&previous_etch_precursor_gas_concentration_previous_time_step_array
        );
305     safe_free(&previous_reactive_product_concentration_previous_time_step_array);
306     safe_free(&right_hand_side);
307     safe_free(&answer);
308     safe_free(&answer_temp);
309     safe_free(&lower_diagonal);
310     safe_free(&central_diagonal);
311     safe_free(&upper_diagonal);
312     safe_free(&r_array);
313     safe_free(&primary_electron_array);
314     safe_free(&backscattered_electron_array);
315     safe_free(&secondary_electron_array);
316     safe_free(&temp_primary_electron_array);
317     safe_free(&temp_primary_electron_flux_profile_array);
318     safe_free(&temp_electron_starting_location_probability_array);
319     safe_free(&local_secondary_electron_location_array);
320     safe_free(&sum_secondary_electron_location_array);
321     safe_free(&local_backscattered_electron_location_array);
322     safe_free(&sum_backscattered_electron_location_array);
323     safe_free(&local_electron_energy_deposited_array);
324     safe_free(&sum_electron_energy_deposited_array);
325     safe_free(&local_electron_maximum_z_depth_array);
326     safe_free(&sum_electron_maximum_z_depth_array);
327     safe_free(&electron_tracking_x_position_array);
328     safe_free(&electron_tracking_y_position_array);
329     safe_free(&electron_tracking_z_position_array);
330     safe_free(&r_coordinate_both_directions);
331     safe_free(&z_coordinate_both_directions);
332     safe_free(&temporary_r_coordinate_secondary_array_one);
333     safe_free(&temporary_r_coordinate_secondary_array_two);
334     safe_free(&temporary_surface_line_equation_array_one);
335     safe_free(&temporary_surface_line_equation_array_two);
336     safe_free(&temporary_surface_line_equation_array_three);
337     safe_free(&dC);
338     safe_free(&temporary_z_coordinate_array);
339     safe_free(&temporary_r_coordinate_array);
340     safe_free(&temporary_reactive_product_concentration_current_time_step_array_y
        );
341     safe_free(&temporary_reactive_product_concentration_current_time_step_array_x
        );
342     safe_free(&
        temporary_etch_precursor_gas_concentration_current_time_step_array_y);
343     safe_free(&
        temporary_etch_precursor_gas_concentration_current_time_step_array_x);
344     safe_free(&
        temporary_deposit_precursor_gas_concentration_current_time_step_array_y);
345     safe_free(&
        temporary_deposit_precursor_gas_concentration_current_time_step_array_x);
346     safe_free(&PE_electron_flux_profile_array);
347     safe_free(&BSE_electron_flux_profile_array);
```

```
348     safe_free(&SE_electron_flux_profile_array);
349     safe_free(&temporary_ND_array);
350     safe_free(&time_per_time_step_array);
351 }
352
353 //calculates the intial precursor coverage
354 void compute_EBIED_initial_coverage(double delta_t, Precursor deposit_precursor,
        Precursor etch_precursor, double *C_etch_p, double *C_depo_p) {
355     int i, n = 1e8;
356     double dt,t = 0.0;
357     double increase = 2.0;
358     #ifdef MPI_ON
359         int node_number;
360         MPI_Comm_rank(MPI_COMM_WORLD, &node_number);
361     #else
362         int node_number = 0;
363     #endif
364     dt = delta_t;
365
366
367     double s_depo = deposit_precursor.sticking_coefficient;
368     double f_depo = deposit_precursor.reactive_product_flux;
369     double A_depo = deposit_precursor.surface_area;
370     double t_depo = deposit_precursor.desorption_time;
371     double s_etch = etch_precursor.sticking_coefficient;
372     double f_etch = etch_precursor.reactive_product_flux;
373     double A_etch = etch_precursor.surface_area;
374     double t_etch = etch_precursor.desorption_time;
375     double C_depo_old = 0.0;
376     double C_etch_old = 0.0;
377
378     if (node_number == 0) {
379         logfile_printf("---------------------------------\n");
380         logfile_printf("Running Initial Precursor Gas Coverage Calculator...\n");
381         logfile_printf("---------------------------------\n");
382     }
383
384     if (f_depo > 0.0 && f_etch == 0.0) {
385         //if only depo gas
386         *C_depo_p = (s_depo*f_depo)/(s_depo*f_depo*A_depo+1/t_depo);
387         *C_etch_p = 0.0;
388     } else if (f_etch > 0.0 && f_depo == 0.0) {
389         //if only etch gas
390         *C_depo_p = 0.0;
391         *C_etch_p = (s_etch*f_etch)/(s_etch*f_etch*A_etch+1/t_etch);;
392     } else {
393         //if both gases
394         // get ballpark figures quickly
395         for(i=0;i<n;i++){
396             double C_vacancies = 1.0-(A_etch*C_etch_old+A_depo*C_depo_old);
397             *C_depo_p = dt*(s_depo*f_depo*C_vacancies - C_depo_old/t_depo) +
        C_depo_old;
```

```
398             *C_etch_p = dt*(s_etch*f_etch*C_vacancies - C_etch_old/t_etch) +
    C_etch_old;
399             if (*C_depo_p > 0.0 && *C_etch_p > 0.0) {  // if negative
    concentration, change dt rate of increase
400                 C_depo_old = *C_depo_p;
401                 C_etch_old = *C_etch_p;
402                 t += dt;
403                 dt *= increase;
404             } else {
405                 dt /= increase;
406                 increase /= 1.1;
407                 if (increase < 1.01) {
408                     increase = 1.01;
409                 }
410             }
411         }
412         // perfect the answer
413         for (i=0;i<n;i++) {
414             double C_vacancies = 1.0-(A_etch**C_etch_p+A_depo**C_depo_p);
415             *C_depo_p = delta_t*(s_depo*f_depo*C_vacancies - *C_depo_p/t_depo) +
    *C_depo_p;
416             *C_etch_p = delta_t*(s_etch*f_etch*C_vacancies - *C_etch_p/t_etch) +
    *C_etch_p;
417         }
418         if (isnan(*C_depo_p) || isnan(*C_etch_p)) {
419             *C_depo_p = (s_depo*f_depo)/(s_depo*f_depo*A_depo+1/t_depo);
420             *C_etch_p = (s_etch*f_etch)/(s_etch*f_etch*A_etch+1/t_etch);
421         }
422     }
423
424     if (node_number == 0) {
425         logfile_printf("----------------------------------\n");
426         logfile_printf("Time:%g seconds, Deposit Initial Coverage:%g,Etchant
    Initial Coverage:%g\n",(double)i*delta_t+t,*C_depo_p,*C_etch_p);
427         logfile_printf("----------------------------------\n");
428     }
429 }
430
431 //Main Program.
432 int main(int argc, char* argv[]) {
433     //Main Program Start.
434     time_t simulation_start_time,simulation_end_time;
435    simulation_start_time = time(0);
436
437     int node_number = 0;
438     int total_number_of_nodes = 1;
439
440     //Setup Save Files.
441     if (node_number == 0) {
442         setup_save_files();
443     }
444
445     //printf mpi status
```

```
446     #ifdef MPI_ON
447         MPI_Init(&argc, &argv);
448         MPI_Comm_rank(MPI_COMM_WORLD, &node_number);
449         MPI_Comm_size(MPI_COMM_WORLD, &total_number_of_nodes);
450         if (node_number == 0) {
451             logfile_printf("--------Simulation Started-------\n");
452             logfile_printf("OpenMPI is ON, running with a total of %d CPU core(s)
    .\n",total_number_of_nodes);
453         }
454     #else
455         logfile_printf("--------Simulation Started-------\n");
456         logfile_printf("OpenMPI is OFF, running with a total of %d CPU core(s).\n
    ",total_number_of_nodes);
457     #endif
458
459     //Structures Declarations.
460     Precursor etch_precursor;
461     Precursor deposit_precursor;
462     Toggle toggle;
463     Material upper_material;
464     Material lower_material;
465     Electron_beam electron_beam;
466
467     //Variable Declarations.
468     int i,j;
469     int run_MC_every_X_time_steps;
470     int save_simulation_data_every_X_time_steps;
471     int save_current_data_counter;
472     int number_of_backscattered_electrons;
473     int maxZindex;
474     double maxZ,FWHM,aspectRatio;
475     double backscattered_electron_coefficient = 0.0;
476     double secondary_electron_coefficient = 0.0;
477     double z_depth_minimum;
478     double z_depth_maximum;
479     double simulation_percentage;
480     double sum_flux = 0.0;
481     double C_etch = 0.0,C_depo = 0.0;
482     double current_time = 0.0;
483
484     //Input Parameters Variable Declarations.
485     int number_of_electron_trajectories = 0;
486     int number_of_surface_bins = 0;
487     int number_of_simulation_time_steps = 0;
488     int run_MC_every_zero_point_X_percent = 0;   // FIXME: minor -
    run_MC_every_zero_point_X_percent doesn't work when # timesteps is small
489     int save_simulation_data_every_X_percent = 0; // FIXME: minor -
    save_simulation_data_every_X_percent doesn't work when # timesteps is small
490     int length_of_electron_energy_deposited_and_electron_maximum_z_depth_array =
    0;
491     int maximum_electron_track_length = 0;
492     double delta_r;
493     double delta_t;
```

```
494     double layered_material_interface_depth;
495     double gas_temperature,substrate_temperature;
496     double deposit_pinned_reaction_electron_cross_section;
497     double deposit_precursor_reaction_electron_cross_section;
498     int number_of_points;
499     int order;
500     int no_etch_area;
501     int seed;
502     int val1 = 0;
503     int val2 = 1;
504     int wave_type;
505     int square_wave_period;
506     int square_wave_min_current;
507     int square_wave_max_current;
508     double triangle_wave_rate;
509     int triangle_wave_min_current;
510     int triangle_wave_max_current;
511     int pulsing_period;
512     int pulse_on_time;
513     int pulse_off_time;
514     double temp_ebeam_current;
515     int time_delay = 0;
516     double precursor_diffusion_tolerance;
517     double SE_MC_weighting_factor;
518     double sim_goodness = 1.0;
519     double max_dt = 0.1;
520
521     //Read In Input Parameter Values From Disk.
522     read_input_parameters(&number_of_electron_trajectories,&
        number_of_surface_bins,&number_of_simulation_time_steps,&
        run_MC_every_zero_point_X_percent,&save_simulation_data_every_X_percent,&
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,&
        maximum_electron_track_length,&delta_r,&delta_t,&electron_beam,&
        layered_material_interface_depth,&upper_material,&lower_material,&
        gas_temperature,&etch_precursor,&deposit_precursor,&
        deposit_pinned_reaction_electron_cross_section,&
        deposit_precursor_reaction_electron_cross_section,&toggle,&number_of_points,&
        order,&no_etch_area,&seed,&wave_type,&square_wave_period,&
        square_wave_min_current,&square_wave_max_current,&triangle_wave_rate,&
        triangle_wave_min_current,&triangle_wave_max_current,&pulsing_period,&
        pulse_on_time,&pulse_off_time,&time_delay,&precursor_diffusion_tolerance,&
        substrate_temperature);
523
524     if (electron_beam.tilt > 89.9) {
525         // input of an electron beam tilt where the beam would not intersect a
        flat surface
526         if (node_number == 0) {
527             logfile_printf("---------------------------------\n");
528             logfile_printf("Error: The electron beam tilt is greater than 89.9
        degrees, the simulator is now exiting! Decrease the electron beam tilt and
        restart the simulation.\n");
529             logfile_printf("---------------------------------\n");
530         }
```

```
531         #ifdef MPI_ON
532             MPI_Barrier(MPI_COMM_WORLD);
533             MPI_Finalize();
534         #else
535             exit(-1);
536         #endif
537     }
538 
539     //Print Input Parameters To Terminal
540     if (node_number == 0) {
541         logfile_printf("---Simulation Input Parameters---\n");
542         logfile_printf("number_of_electron_trajectories = %d\n",
    number_of_electron_trajectories);
543         logfile_printf("number_of_surface_bins = %d\n",number_of_surface_bins);
544         logfile_printf("number_of_simulation_time_steps = %d\n",
    number_of_simulation_time_steps);
545         logfile_printf("run_MC_every_zero_point_X_percent = %d\n",
    run_MC_every_zero_point_X_percent);
546         logfile_printf("save_simulation_data_every_X_percent = %d\n",
    save_simulation_data_every_X_percent);
547         logfile_printf("
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array = %d\n
    ",length_of_electron_energy_deposited_and_electron_maximum_z_depth_array);
548         logfile_printf("maximum_electron_track_length = %d\n",
    maximum_electron_track_length);
549         logfile_printf("delta_r = %g\n",delta_r);
550         logfile_printf("delta_t = %g\n",delta_t);
551         logfile_printf("electron_beam.cutoff_energy = %g\n",electron_beam.
    cutoff_energy);
552         logfile_printf("electron_beam.top_hat_abruptness = %g\n",electron_beam.
    top_hat_abruptness);
553         logfile_printf("electron_beam.diameter = %g\n",electron_beam.diameter);
554         logfile_printf("electron_beam.energy = %g\n",electron_beam.energy);
555         logfile_printf("electron_beam.current = %g\n",electron_beam.current);
556         logfile_printf("electron_beam.tilt = %g\n",electron_beam.tilt);
557         logfile_printf("layered_material_interface_depth = %g\n",
    layered_material_interface_depth);
558         logfile_printf("upper_material.atomic_number = %g\n",upper_material.
    atomic_number);
559         logfile_printf("upper_material.atomic_weight = %g\n",upper_material.
    atomic_weight);
560         logfile_printf("upper_material.density = %g\n",upper_material.density);
561         logfile_printf("upper_material.epsilon = %g\n",upper_material.epsilon);
562         logfile_printf("upper_material.lambda = %g\n",upper_material.lambda);
563         logfile_printf("lower_material.atomic_number = %g\n",lower_material.
    atomic_number);
564         logfile_printf("lower_material.atomic_weight = %g\n",lower_material.
    atomic_weight);
565         logfile_printf("lower_material.density = %g\n",lower_material.density);
566         logfile_printf("lower_material.epsilon = %g\n",lower_material.epsilon);
567         logfile_printf("lower_material.lambda = %g\n",lower_material.lambda);
568         logfile_printf("gas_temperature = %g\n",gas_temperature);
569         logfile_printf("substrate_temperature = %g\n",substrate_temperature);
```

```
570         logfile_printf("etch_precursor.gas_partial_pressure = %g\n",
        etch_precursor.gas_partial_pressure);
571         logfile_printf("etch_precursor.reactive_product_molecular_mass = %g\n",
        etch_precursor.reactive_product_molecular_mass);
572         logfile_printf("etch_precursor.surface_area = %g\n",etch_precursor.
        surface_area);
573         logfile_printf("etch_precursor.desorption_energy = %g\n",etch_precursor.
        desorption_energy);
574         logfile_printf("etch_precursor.desorption_attempt_frequency = %g\n",
        etch_precursor.desorption_attempt_frequency);
575         logfile_printf("etch_precursor.diffusion_energy = %g\n",etch_precursor.
        diffusion_energy);
576         logfile_printf("etch_precursor.diffusion_attempt_frequency = %g\n",
        etch_precursor.diffusion_attempt_frequency);
577         logfile_printf("etch_precursor.sticking_coefficient = %g\n",
        etch_precursor.sticking_coefficient);
578         logfile_printf("etch_precursor.PE_electron_cross_section = %g\n",
        etch_precursor.PE_electron_cross_section);
579         logfile_printf("etch_precursor.BSE_electron_cross_section = %g\n",
        etch_precursor.BSE_electron_cross_section);
580         logfile_printf("etch_precursor.SE_electron_cross_section = %g\n",
        etch_precursor.SE_electron_cross_section);
581         logfile_printf("deposit_precursor.gas_partial_pressure = %g\n",
        deposit_precursor.gas_partial_pressure);
582         logfile_printf("deposit_precursor.reactive_product_molecular_mass = %g\n"
        ,deposit_precursor.reactive_product_molecular_mass);
583         logfile_printf("deposit_precursor.surface_area = %g\n",deposit_precursor.
        surface_area);
584         logfile_printf("deposit_precursor.desorption_energy = %g\n",
        deposit_precursor.desorption_energy);
585         logfile_printf("deposit_precursor.desorption_attempt_frequency = %g\n",
        deposit_precursor.desorption_attempt_frequency);
586         logfile_printf("deposit_precursor.diffusion_energy = %g\n",
        deposit_precursor.diffusion_energy);
587         logfile_printf("deposit_precursor.diffusion_attempt_frequency = %g\n",
        deposit_precursor.diffusion_attempt_frequency);
588         logfile_printf("deposit_precursor.sticking_coefficient = %g\n",
        deposit_precursor.sticking_coefficient);
589         logfile_printf("deposit_precursor.PE_electron_cross_section = %g\n",
        deposit_precursor.PE_electron_cross_section);
590         logfile_printf("deposit_precursor.BSE_electron_cross_section = %g\n",
        deposit_precursor.BSE_electron_cross_section);
591         logfile_printf("deposit_precursor.SE_electron_cross_section = %g\n",
        deposit_precursor.SE_electron_cross_section);
592         logfile_printf("deposit_pinned_reaction_electron_cross_section = %g\n",
        deposit_pinned_reaction_electron_cross_section);
593         logfile_printf("deposit_precursor_reaction_electron_cross_section = %g\n"
        ,deposit_precursor_reaction_electron_cross_section);
594         logfile_printf("toggle.electron_trajectory_simulator = %d\n",toggle.
        electron_trajectory_simulator);
595         logfile_printf("toggle.electron_trajectory_tracking = %d\n",toggle.
        electron_trajectory_tracking);
```

```
596         logfile_printf("toggle.electron_beam_projection = %d\n",toggle.
        electron_beam_projection);
597         logfile_printf("toggle.electron_beam_shape = %d\n",toggle.
        electron_beam_shape);
598         logfile_printf("toggle.surface_evolution = %d\n",toggle.surface_evolution
        );
599         logfile_printf("toggle.previous_simulation = %d\n",toggle.
        previous_simulation);
600         logfile_printf("toggle.coverage = %d\n",toggle.coverage);
601         logfile_printf("wave_type = %d\n",wave_type);
602         logfile_printf("square_wave_period = %d\n",square_wave_period);
603         logfile_printf("square_wave_min_current = %d\n",square_wave_min_current);
604         logfile_printf("square_wave_max_current = %d\n",square_wave_max_current);
605         logfile_printf("triangle_wave_rate = %d\n",triangle_wave_rate);
606         logfile_printf("triangle_wave_min_current = %d\n",
        triangle_wave_min_current);
607         logfile_printf("triangle_wave_max_current = %d\n",
        triangle_wave_max_current);
608         logfile_printf("pulsing_period = %d\n",pulsing_period);
609         logfile_printf("pulse_on_time = %d\n",pulse_on_time);
610         logfile_printf("pulse_off_time = %d\n",pulse_off_time);
611         logfile_printf("number_of_points = %d\n",number_of_points);
612         logfile_printf("order = %d\n",order);
613         logfile_printf("no_etch_area = %d\n",no_etch_area);
614         logfile_printf("seed = %d\n",seed);
615         logfile_printf("time_delay = %d\n",time_delay);
616         logfile_printf("precursor_diffusion_tolerance = %g\n",
        precursor_diffusion_tolerance);
617         logfile_printf("---------------------------------\n");
618     }
619     maximum_electron_track_length = 1000;
620
621     //Malloc Variable Declarations.
622     malloc_memory(number_of_surface_bins,
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
        maximum_electron_track_length,number_of_simulation_time_steps);
623     print_electron_flux_profile_array = Make2DDoubleArray(200/
        save_simulation_data_every_X_percent,number_of_surface_bins);
624     print_N_little_e = Make2DDoubleArray(200/save_simulation_data_every_X_percent
        ,number_of_surface_bins);
625     print_N_little_d = Make2DDoubleArray(200/save_simulation_data_every_X_percent
        ,number_of_surface_bins);
626     print_N_big_d = Make2DDoubleArray(200/save_simulation_data_every_X_percent,
        number_of_surface_bins);
627     print_growth_rate = Make2DDoubleArray(200/
        save_simulation_data_every_X_percent,number_of_surface_bins);
628     print_surface = Make2DDoubleArray(200/save_simulation_data_every_X_percent,
        number_of_surface_bins);
629     print_secondary_electrons = Make2DDoubleArray(200/
        save_simulation_data_every_X_percent,number_of_surface_bins);
630     print_backscattered_electrons = Make2DDoubleArray(200/
        save_simulation_data_every_X_percent,number_of_surface_bins);
631
```

```
632     //Initial Calculations.
633     SE_MC_weighting_factor = 100000.0;
634     run_MC_every_X_time_steps = 1;
635     save_simulation_data_every_X_time_steps = number_of_simulation_time_steps
        /100*save_simulation_data_every_X_percent;//save the simulator data every X
        time steps to disk.
636     etch_precursor.reactive_product_flux = (etch_precursor.gas_partial_pressure
        *1.0E-20)/sqrt(2.0*PI*etch_precursor.reactive_product_molecular_mass*
        BOLTZMANN_CONSTANT*gas_temperature);//reactive product flux, in A-2s-1.
637     deposit_precursor.reactive_product_flux = (deposit_precursor.
        gas_partial_pressure*1.0E-20)/sqrt(2.0*PI*deposit_precursor.
        reactive_product_molecular_mass*BOLTZMANN_CONSTANT*gas_temperature);//
        reactive product flux, in A-2s-1.
638     etch_precursor.desorption_time = etch_precursor.desorption_attempt_frequency*
        exp((etch_precursor.desorption_energy*ELECTRON_CHARGE)/(BOLTZMANN_CONSTANT*
        substrate_temperature));
639     etch_precursor.diffusion_coefficient = etch_precursor.
        diffusion_attempt_frequency*exp(-(etch_precursor.diffusion_energy*
        ELECTRON_CHARGE)/(BOLTZMANN_CONSTANT*substrate_temperature));
640     deposit_precursor.desorption_time = deposit_precursor.
        desorption_attempt_frequency*exp((deposit_precursor.desorption_energy*
        ELECTRON_CHARGE)/(BOLTZMANN_CONSTANT*substrate_temperature));
641     deposit_precursor.diffusion_coefficient = deposit_precursor.
        diffusion_attempt_frequency*exp(-(deposit_precursor.diffusion_energy*
        ELECTRON_CHARGE)/(BOLTZMANN_CONSTANT*substrate_temperature));
642     save_current_data_counter = 1;
643     electron_beam.current *= 1.0E-12/ELECTRON_CHARGE;
644     temp_ebeam_current = electron_beam.current;
645
646     //print initial calculations
647     if (node_number == 0) {
648         logfile_printf("---------------------------------\n");
649         logfile_printf("Deposit Flux:%g A-2s-1,Etchant Flux:%g A-2s-1\n",
        deposit_precursor.reactive_product_flux,etch_precursor.reactive_product_flux)
        ;
650         logfile_printf("Deposit Desorption Time:%g s,Etchant Desorption Time:%g s
        \n",deposit_precursor.desorption_time,etch_precursor.desorption_time);
651         logfile_printf("Deposit Diffusion Coefficient:%g A2s-1,Etchant Diffusion
        Coefficient:%g A2s-1\n",deposit_precursor.diffusion_coefficient,
        etch_precursor.diffusion_coefficient);
652         logfile_printf("---------------------------------\n");
653     }
654
655     //check to see if diffusion coefficient is above 1.0
656     toggle.diffusion = 1;
657     if (deposit_precursor.diffusion_coefficient<1.0 && deposit_precursor.
        gas_partial_pressure > 0.0) {
658         if (node_number == 0) {
659             logfile_printf("---------------------------------\n");
660             logfile_printf("Warning: The deposition precursor molecule diffusion
        coefficient is less than 1.0!  Turning diffusion off!\n");
661             logfile_printf("---------------------------------\n");
662         }
```

```
663         toggle.diffusion = 0;
664     }
665     if (etch_precursor.diffusion_coefficient<1.0 && etch_precursor.
    gas_partial_pressure > 0.0) {
666         if (node_number == 0) {
667             logfile_printf("--------------------------------\n");
668             logfile_printf("Warning: The etch precursor molecule diffusion
    coefficient is less than 1.0!  Turning diffusion off!\n");
669             logfile_printf("--------------------------------\n");
670         }
671         toggle.diffusion = 0;
672     }
673 
674     //check dimentionless diffusion coefficient obeys the inequalty, 0<D'<0.5
675     if (toggle.diffusion) {
676         double dimentionless_diffusion_coefficient = (deposit_precursor.
    diffusion_coefficient*delta_t)/(delta_r*delta_r);
677         if (dimentionless_diffusion_coefficient < 0 ||
    dimentionless_diffusion_coefficient > 0.5) {
678             if (node_number == 0) {
679                 logfile_printf("--------------------------------\n");
680                 logfile_printf("Warning (Deposit Gas): The dimentionless
    diffusion coefficient, D'=%g, does not obey the inequalty, 0<D'<0.5, surface
    diffusion may no longer behave correctly. Increase the size of input
    parameter, 'delta_r', or decrease the size of input parameter, 'delta_t', to
    prevent this warning.\n",dimentionless_diffusion_coefficient);
681                 logfile_printf("--------------------------------\n");
682             }
683         }
684         dimentionless_diffusion_coefficient = (etch_precursor.
    diffusion_coefficient*delta_t)/(delta_r*delta_r);
685         if (dimentionless_diffusion_coefficient < 0 ||
    dimentionless_diffusion_coefficient > 0.5) {
686             if (node_number == 0) {
687                 logfile_printf("--------------------------------\n");
688                 logfile_printf("Warning (Etch Gas): The dimentionless diffusion
    coefficient, D'=%g, does not obey the inequalty, 0<D'<0.5, surface diffusion
    may no longer behave correctly. Increase the size of input parameter, '
    delta_r', or decrease the size of input parameter, 'delta_t', to prevent this
     warning.\n",dimentionless_diffusion_coefficient);
689                 logfile_printf("--------------------------------\n");
690             }
691         }
692     }
693 
694     //Setup Core Arrays.
695     if (toggle.previous_simulation == 0) {
696         // set up new simulation start
697         if (toggle.coverage) {
698             compute_EBIED_initial_coverage(delta_t, deposit_precursor,
    etch_precursor, &C_etch, &C_depo);
699         } else {
700             C_depo = 0.0;
```

```
701             C_etch = 0.0;
702         }
703         for (i=0;i<number_of_surface_bins;i++) {
704             deposit_precursor_gas_concentration_previous_two_time_step_array[i] =
     C_depo;//n_d;
705             deposit_precursor_gas_concentration_previous_time_step_array[i] =
    C_depo;//n_d;
706             etch_precursor_gas_concentration_previous_two_time_step_array[i] =
    C_etch;//n_e;
707             etch_precursor_gas_concentration_previous_time_step_array[i] = C_etch
    ;//n_e;
708             reactive_product_concentration_previous_time_step_array[i] = 0.0;
709             growth_or_etch_rate_array[i] = 0.0;
710             z_coordinate_array[i] = 0.0;
711             r_coordinate_array[i] = ((double)i+1.0)*delta_r;
712         }
713     } else { //fix EBIED previous simulation
714         read_input_previous_simulation(number_of_surface_bins);
715         for (i=0;i<number_of_surface_bins;i++) {
716             deposit_precursor_gas_concentration_previous_two_time_step_array[i] =
     previous_deposit_precursor_gas_concentration_previous_time_step_array[i];
717             deposit_precursor_gas_concentration_previous_time_step_array[i] =
    previous_deposit_precursor_gas_concentration_previous_time_step_array[i];
718             etch_precursor_gas_concentration_previous_two_time_step_array[i] =
    previous_etch_precursor_gas_concentration_previous_time_step_array[i];
719             etch_precursor_gas_concentration_previous_time_step_array[i] =
    previous_etch_precursor_gas_concentration_previous_time_step_array[i];
720             reactive_product_concentration_previous_time_step_array[i] =
    previous_reactive_product_concentration_previous_time_step_array[i];
721             growth_or_etch_rate_array[i] = 0.0;
722             z_coordinate_array[i] = previous_z_coordinate_array[i];
723             r_coordinate_array[i] = previous_r_coordinate_array[i];
724         }
725         C_depo =
    previous_deposit_precursor_gas_concentration_previous_time_step_array[i];
726         C_etch =
    previous_etch_precursor_gas_concentration_previous_time_step_array[i];
727     }
728 
729     //Electron Flux Profile.
730     electron_flux_profile(toggle,electron_beam,number_of_surface_bins,delta_t,
    delta_r,backscattered_electron_coefficient,secondary_electron_coefficient,i);
731 
732     /*if(toggle.electron_trajectory_simulator)
733     {
734         //Monte Carlo, Surface Setup.
735         monte_carlo_surface_setup(number_of_surface_bins,&z_depth_minimum,&
    z_depth_maximum);
736 
737         //Monte Carlo, Electron Trajectories.
```

```
738         monte_carlo_electron_trajectory_simulator(electron_beam,lower_material,
        upper_material,toggle.electron_trajectory_tracking,delta_r,
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
        number_of_surface_bins,layered_material_interface_depth,
        maximum_electron_track_length,number_of_electron_trajectories,z_depth_minimum
        ,z_depth_maximum,&number_of_backscattered_electrons,seed);
739
740         //Monte Carlo, Data Collection.
741         monte_carlo_data_collection(
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
        number_of_electron_trajectories,number_of_surface_bins,
        number_of_backscattered_electrons,&backscattered_electron_coefficient,&
        secondary_electron_coefficient);
742
743         //Electron Flux Profile.
744         electron_flux_profile(toggle,electron_beam,number_of_surface_bins,delta_t
        ,delta_r,backscattered_electron_coefficient,secondary_electron_coefficient,i)
        ;
745     }
746
747     //printf BSE and SE
748     if (node_number == 0) {
749         logfile_printf("---------------------------------\n");
750         logfile_printf("Secondary Electron Yield:%g,Backscattered Electron
        Coefficient:%g\n",secondary_electron_coefficient,
        backscattered_electron_coefficient);
751         logfile_printf("---------------------------------\n");
752     }
753
754     for (i=0;i<number_of_surface_bins;i++) {
755         sum_flux += electron_flux_profile_array[i];  // FIXME: does this need X
        annulus?
756     }
757
758     if (node_number == 0) {
759         logfile_printf("---------------------------------\n");
760         logfile_printf("Intergrated Flux:%g electrons/sec\n",sum_flux);
761         logfile_printf("---------------------------------\n");
762     }*/
763
764     //EBIED Simulator.
765     clock_t continuum_start_time,continuum_end_time;
766     clock_t surfevo_start_time,surfevo_end_time;
767     clock_t eflux_start_time,eflux_end_time;
768     clock_t montecarlo_start_time,montecarlo_end_time;
769     for (i=0;i<number_of_simulation_time_steps;i++) {
770    //Monte Carlo.
771         if(toggle.electron_trajectory_simulator && electron_beam.current > 0) {
772             montecarlo_start_time = clock();
773             //Monte Carlo, Surface Setup.
774             monte_carlo_surface_setup(number_of_surface_bins,&z_depth_minimum,&
        z_depth_maximum);
775
```

```
776            //Monte Carlo, Electron Trajectories.
777            monte_carlo_electron_trajectory_simulator(electron_beam,
        lower_material,upper_material,toggle.electron_trajectory_tracking,delta_r,
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
        number_of_surface_bins,layered_material_interface_depth,
        maximum_electron_track_length,number_of_electron_trajectories,z_depth_minimum
        ,z_depth_maximum,&number_of_backscattered_electrons,seed);
778
779            //Monte Carlo, Data Collection.
780            monte_carlo_data_collection(
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
        number_of_electron_trajectories,number_of_surface_bins,
        number_of_backscattered_electrons,&backscattered_electron_coefficient,&
        secondary_electron_coefficient);
781            if (node_number == 0) {
782                logfile_printf("---------------------------------\n");
783                logfile_printf("Secondary Electron Yield:%g,Backscattered
        Electron Coefficient:%g\n",secondary_electron_coefficient,
        backscattered_electron_coefficient);
784                logfile_printf("---------------------------------\n");
785            }
786            montecarlo_end_time = clock();
787            //Electron Flux Profile.
788            eflux_start_time = clock();
789            electron_flux_profile(toggle,electron_beam,number_of_surface_bins,
        delta_t,delta_r,backscattered_electron_coefficient,
        secondary_electron_coefficient,i);
790            eflux_end_time = clock();
791        }
792
793        //check for diffusion
794        if (toggle.diffusion) {
795            //EBIED Solver.
796            continuum_start_time = clock();
797            if (etch_precursor.gas_partial_pressure > 0.0) {
798                //if etch and/or depo run EBIED code
799                delta_t *= 1.1;
800                if (delta_t > max_dt) {
801                    delta_t = max_dt;
802                }
803                ebied_solver(etch_precursor,deposit_precursor,toggle,
        gas_temperature,deposit_pinned_reaction_electron_cross_section,
        deposit_precursor_reaction_electron_cross_section,delta_t,delta_r,
        number_of_surface_bins,no_etch_area);
804            } else {
805                //if depo only run EBID code
806                delta_t *= 1.1;
807                if (delta_t > max_dt) {
808                    delta_t = max_dt;
809                }
810                ebid_solver(deposit_precursor,toggle,gas_temperature,delta_t,
        delta_r,number_of_surface_bins);
811            }
```

```
812             current_time += delta_t;
813             time_per_time_step_array[i] = current_time;
814             continuum_end_time = clock();
815         } else {
816             //EBIED Solver, diffusion off.
817             ebied_solver_no_diffusion(etch_precursor,deposit_precursor,toggle,
    gas_temperature,deposit_pinned_reaction_electron_cross_section,
    deposit_precursor_reaction_electron_cross_section,delta_t,delta_r,
    number_of_surface_bins);
818         }
819
820         //Surface Evolution.
821         surfevo_start_time = clock();
822         surface_evolution(deposit_precursor,etch_precursor,lower_material,
    upper_material,toggle.surface_evolution,number_of_points,order,
    number_of_surface_bins,delta_t,delta_r,layered_material_interface_depth,i,
    toggle);
823         surfevo_end_time = clock();
824
825         //Save Current Simulation Data.
826         if (save_current_data_counter == save_simulation_data_every_X_time_steps
    && (node_number == 0)) {
827             save_current_simulation_data(
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    backscattered_electron_coefficient,secondary_electron_coefficient,
    number_of_surface_bins,number_of_electron_trajectories,i+2,val1,val2,
    number_of_simulation_time_steps,save_simulation_data_every_X_percent);
828             val1 += 2;
829             val2 += 2;
830             save_current_data_counter = 1;
831         } else {
832             save_current_data_counter++;
833         }
834
835         //Simulation Progress.
836         if ((i%(number_of_simulation_time_steps/100) == 0) && (i != 0) && (
    node_number == 0)) {
837             simulation_end_time = time(0);
838             simulation_percentage = (double)i/(double)
    number_of_simulation_time_steps*100.0;
839             double time_left_hours = floor((difftime(simulation_end_time,
    simulation_start_time)/simulation_percentage*(100.0-simulation_percentage))
    /60/60);
840             double time_left_mins = floor((difftime(simulation_end_time,
    simulation_start_time)/simulation_percentage*(100.0-simulation_percentage))
    /60-time_left_hours*60);
841             double time_left_secs = floor((difftime(simulation_end_time,
    simulation_start_time)/simulation_percentage*(100.0-simulation_percentage))-
    time_left_mins*60-time_left_hours*60*60);
842             logfile_printf("Progress: %g%%, Estimated Time Remaining: %g hour(s)
    %g minute(s) %g second(s). Current Time:%g, DeltaT:%g,TimeStep:%d\n",
    simulation_percentage,time_left_hours,time_left_mins,time_left_secs,
    current_time,delta_t,i);
```

```
843             if (z_coordinate_array[0] > 0.0) {
844                 maxZ = z_coordinate_array[1];
845                 maxZindex = 1;
846                 for (j=2;j<number_of_surface_bins;j++) {
847                     if (z_coordinate_array[j] > maxZ) {
848                         maxZ = z_coordinate_array[j];
849                         maxZindex = j;
850                     }
851                 }
852                 for (j=maxZindex;j<number_of_surface_bins;j++) {
853                     if (maxZ/2.0 > z_coordinate_array[j]) {
854                         FWHM = 2*r_coordinate_array[j-1];
855                         break;
856                     }
857                 }
858                 aspectRatio = maxZ/FWHM;
859             }
860             //calculate structure volume
861             for (j=0;j<number_of_surface_bins;j++) {
862                 if (z_coordinate_array[j] >= layered_material_interface_depth) {
863                     deposit_precursor.reactive_product_density = upper_material.
    density*1.0E-27;
864                 } else {
865                     deposit_precursor.reactive_product_density = lower_material.
    density*1.0E-27;
866                 }
867                 temporary_r_coordinate_array[j] = r_coordinate_array[j];
868                 temporary_z_coordinate_array[j] =
    reactive_product_concentration_current_time_step_array[j]*deposit_precursor.
    reactive_product_molecular_mass/deposit_precursor.reactive_product_density;
869             }
870             double ND_volume = temporary_z_coordinate_array[0]*PI*
    temporary_r_coordinate_array[0]*temporary_r_coordinate_array[0];
871             double current_volume = z_coordinate_array[0]*PI*r_coordinate_array
    [0]*r_coordinate_array[0];
872             double interval = (r_coordinate_array[number_of_surface_bins-1]-
    r_coordinate_array[0])/10000.0;
873             double xi = r_coordinate_array[0]+interval;
874             for (j=1;j<10000;j++) {
875                 current_volume += linear_interp(r_coordinate_array,
    z_coordinate_array,number_of_surface_bins,xi)*PI*(xi*xi-((xi-interval)*(xi-
    interval)));
876                 ND_volume += linear_interp(temporary_r_coordinate_array,
    temporary_z_coordinate_array,number_of_surface_bins,xi)*PI*(xi*xi-((xi-
    interval)*(xi-interval)));
877                 xi += interval;
878             }
879             //adjust max delta t based on simulation goodness
880             sim_goodness = 1.0-fabs(current_volume/ND_volume-1);
881             //max_dt *= sim_goodness;
882
883             logfile_printf("---------------------------------\n");
```

```
884             logfile_printf("Simulation Goodness [Good(1)-Bad(0)]:%g,Current
        Structure Volume (A3):%g,Deposited Structure Volume (A3):%g\n",sim_goodness,
        current_volume,ND_volume);
885             logfile_printf("---------------------------------\n");
886             //Check to see if molecules have diffused out of final bin.
887             if (deposit_precursor_gas_concentration_current_time_step_array[
        number_of_surface_bins-1] < C_depo*(1-(precursor_diffusion_tolerance/100))) {
888                 if (node_number == 0) {
889                     logfile_printf("---------------------------------\n");
890                     logfile_printf("Warning: Deposit precursor molecules have
        diffused out of final surface bin, surface diffusion may no longer behave
        correctly. Increase the size of input parameter, 'number_of_surface_bins', to
         prevent this warning.\n");
891                     logfile_printf("---------------------------------\n");
892                 }
893             }
894             if (etch_precursor_gas_concentration_current_time_step_array[
        number_of_surface_bins-1] < C_etch*(1-(precursor_diffusion_tolerance/100))) {
895                 if (node_number == 0) {
896                     logfile_printf("---------------------------------\n");
897                     logfile_printf("Warning: Etchant precursor molecules have
        diffused out of final surface bin, surface diffusion may no longer behave
        correctly. Increase the size of input parameter, 'number_of_surface_bins', to
         prevent this warning.\n");
898                     logfile_printf("---------------------------------\n");
899                 }
900             }
901         }
902 
903         #ifdef MPI_ON
904             MPI_Barrier(MPI_COMM_WORLD);
905         #endif
906     }
907 
908     //Main Program End.
909     if (node_number == 0) {
910         logfile_printf("Progress: 100%%\n");
911         //Save data for next simulation
912         FILE *fp;
913         fp=fopen("./inputs/input_previous_simulation.txt", "w");
914         for (i=0;i<number_of_surface_bins;i++) {
915             fprintf(fp,"%d\t%g\t%g\t%g\t%g\t%g\n",i+1,r_coordinate_array[i],
        z_coordinate_array[i],
        deposit_precursor_gas_concentration_current_time_step_array[i],
        etch_precursor_gas_concentration_current_time_step_array[i],
        reactive_product_concentration_current_time_step_array[i]);
916         }
917         fclose(fp);
918         simulation_end_time = time(0);
919         double time_left_hours = floor(difftime(simulation_end_time,
        simulation_start_time)/60/60);
920         double time_left_mins = floor(difftime(simulation_end_time,
        simulation_start_time)/60)-time_left_hours*60;
```

```
921         double time_left_secs = floor(difftime(simulation_end_time,
    simulation_start_time))-time_left_mins*60-time_left_hours*60*60;
922         logfile_printf("---------Simulation Ended--------\n");
923         logfile_printf("The total wall time was: %g hour(s) %g minute(s) %g
    second(s)\n",time_left_hours,time_left_mins,time_left_secs);
924         logfile_printf("--------------------------------\n");
925         if (z_coordinate_array[0] > 0.0) {
926             logfile_printf("MaxZ Height:%g\tFWHM:%g\tAspect Ratio:%g\tTotal Time
    :%g\n",maxZ,FWHM,aspectRatio,current_time);
927         }
928         //Calculate time for Continuum and Monte Carlo modules
929         logfile_printf("Continuum Run Time:%E sec\n",(double)(continuum_end_time-
    continuum_start_time)/CLOCKS_PER_SEC);
930         logfile_printf("Surface Evolution Run Time:%E sec\n",(double)(
    surfevo_end_time-surfevo_start_time)/CLOCKS_PER_SEC);
931         logfile_printf("Electron Flux Run Time:%E sec\n",(double)(eflux_end_time-
    eflux_start_time)/CLOCKS_PER_SEC);
932         logfile_printf("Monte Carlo Run Time:%E sec\n",(double)(
    montecarlo_end_time-montecarlo_start_time)/CLOCKS_PER_SEC);
933     }
934     #ifdef MPI_ON
935         MPI_Barrier(MPI_COMM_WORLD);
936         MPI_Finalize();
937     #endif
938     exit(0);
939 
940 }
```

B.5 EBID/EBIE & EBIED Solver

```
1  //
2  //  ebie_ebid_solver.c
3  //
4  //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5  //
6  //  This function contains the continuum model equations for EBID/EBIE/EBIED.
7  //
8  #ifdef XCODE
9  #include <stdlib.h>
10 #include <math.h>
11 #include <stdio.h>
12 #include "structures.h"
13 #include "prototypes.h"
14 #include <gsl/gsl_fit.h>
15 #endif
16 
17 //calculates the precursor and deposit molecule concentration over time
18 void ebied_solver(Precursor etch_precursor,
19                   Precursor deposit_precursor,
20                   Toggle toggle,
```

```
21                     double gas_temperature,
22                     double deposit_pinned_reaction_electron_cross_section,
23                     double deposit_precursor_reaction_electron_cross_section,
24                     double delta_t,
25                     double delta_r,
26                     int number_of_surface_bins,
27                     int no_etch_area)
28 {
29
30     int j;
31     double r;
32
33
34     //if surface evo, make an r array a string of point along the surface
35     //this maybe be incorrect and must FIX!!!
36     if (toggle.surface_evolution) {
37         r_array[0] = r_coordinate_array[0];
38         for (j=1;j<number_of_surface_bins;j++) {
39             double dr = r_coordinate_array[j]-r_coordinate_array[j-1];
40             double dz = z_coordinate_array[j]-z_coordinate_array[j-1];
41             r_array[j] = r_array[j-1]+sqrt(dr*dr + dz*dz);
42         }
43     } else {
44         for (j=0;j<number_of_surface_bins;j++) {
45             r_array[j] = r_coordinate_array[j];
46         }
47     }
48
49     double D_etch = etch_precursor.diffusion_coefficient;
50     double s_etch = etch_precursor.sticking_coefficient;
51     double f_etch = etch_precursor.reactive_product_flux;
52     double A_etch = etch_precursor.surface_area;
53     double t_etch = etch_precursor.desorption_time;
54     double sigma_PE_etch = etch_precursor.PE_electron_cross_section;
55     double sigma_BSE_etch = etch_precursor.BSE_electron_cross_section;
56     double sigma_SE_etch = etch_precursor.SE_electron_cross_section;
57
58     double D_depo = deposit_precursor.diffusion_coefficient;
59     double s_depo = deposit_precursor.sticking_coefficient;
60     double f_depo = deposit_precursor.reactive_product_flux;
61     double A_depo = deposit_precursor.surface_area;
62     double t_depo = deposit_precursor.desorption_time;
63     double sigma_PE_depo = deposit_precursor.PE_electron_cross_section;
64     double sigma_BSE_depo = deposit_precursor.BSE_electron_cross_section;
65     double sigma_SE_depo = deposit_precursor.SE_electron_cross_section;
66
67     //calculate Etch Molecule Concentration the first time
68     right_hand_side[0] = 0.0;
69     answer[0] = 1.0;
70     lower_diagonal[0] = ((-1.0*D_etch)/(2.0*delta_r*delta_r))+(D_etch/(4.0*
       delta_r));
71     central_diagonal[0] = 1.0;
72     upper_diagonal[0] = -1.0;
```

```
73      right_hand_side[number_of_surface_bins-1] = 0.0;
74      answer[number_of_surface_bins-1] = 1.0;
75      lower_diagonal[number_of_surface_bins-1] = -1.0;
76      central_diagonal[number_of_surface_bins-1] = 1.0;
77      r = r_array[number_of_surface_bins-1];
78      upper_diagonal[number_of_surface_bins-1] = ((-1.0*D_etch)/(2.0*delta_r*
        delta_r))-(D_etch/(4.0*r*delta_r));
79      for (j=1;j<(number_of_surface_bins-1);j++) {
80          if (j <= no_etch_area) {
81              sigma_PE_etch = 0.0;
82              sigma_BSE_etch = 0.0;
83              sigma_SE_etch = 0.0;
84          } else {
85              sigma_PE_etch = etch_precursor.PE_electron_cross_section;
86              sigma_BSE_etch = etch_precursor.BSE_electron_cross_section;
87              sigma_SE_etch = etch_precursor.SE_electron_cross_section;
88          }
89          r = r_array[j];
90          right_hand_side[j] = s_etch*f_etch+
        etch_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_etch*f_etch*A_etch/2.0-1.0/(2.0*t_etch)-(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])/2.0-((2.0*D_etch)/(2.0*delta_r*delta_r)))
        +(deposit_precursor_gas_concentration_previous_time_step_array[j])*(-1.0*
        s_etch*f_etch*A_depo)+
        etch_precursor_gas_concentration_previous_time_step_array[j-1]*((D_etch/(2.0*
        delta_r*delta_r))-(D_etch/(4.0*r*delta_r)))+
        etch_precursor_gas_concentration_previous_time_step_array[j+1]*((D_etch/(2.0*
        delta_r*delta_r))+(D_etch/(4.0*r*delta_r)));
91          lower_diagonal[j] = ((-1.0*D_etch)/(2.0*delta_r*delta_r))+(D_etch/(4.0*r*
        delta_r));
92          central_diagonal[j] = (1.0/delta_t+s_etch*f_etch*A_etch/2.0+1.0/(2.0*
        t_etch)+(sigma_PE_etch*PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])/2.0)+((2.0*D_etch)/(2.0*delta_r*delta_r));
93          upper_diagonal[j] = ((-1.0*D_etch)/(2.0*delta_r*delta_r))-(D_etch/(4.0*r*
        delta_r));
94          answer[j] = 1.0;
95      }
96
97      solve_matrix(number_of_surface_bins,
98                   lower_diagonal,
99                   central_diagonal,
100                  upper_diagonal,
101                  right_hand_side,
102                  answer);
103
104     //calculate the Deposit Molecule Concentration the first time
105     right_hand_side[0] = 0.0;
106     answer[0] = 1.0;
107     lower_diagonal[0] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))+(D_depo/(4.0*
        delta_r));
```

```
108     central_diagonal[0] = 1.0;
109     upper_diagonal[0] = -1.0;
110     right_hand_side[number_of_surface_bins-1] = 0.0;
111     answer[number_of_surface_bins-1] = 1.0;
112     lower_diagonal[number_of_surface_bins-1] = -1.0;
113     central_diagonal[number_of_surface_bins-1] = 1.0;
114     r = r_array[number_of_surface_bins-1];
115     upper_diagonal[number_of_surface_bins-1] = ((-1.0*D_depo)/(2.0*delta_r*
        delta_r))-(D_depo/(4.0*r*delta_r));
116     for (j=1;j<(number_of_surface_bins-1);j++) {
117         r = r_array[j];
118         right_hand_side[j] = s_depo*f_depo+
        deposit_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_depo*f_depo*A_depo/2.0-1.0/(2.0*t_depo)-(sigma_PE_depo*
        PE_electron_flux_profile_array[j]+sigma_BSE_depo*
        BSE_electron_flux_profile_array[j]+sigma_SE_depo*
        SE_electron_flux_profile_array[j])/2.0-(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])*(answer[j]+
        etch_precursor_gas_concentration_previous_time_step_array[j])*
        deposit_precursor_reaction_electron_cross_section/4.0-((2.0*D_depo)/(2.0*
        delta_r*delta_r)))+((answer[j]+
        etch_precursor_gas_concentration_previous_time_step_array[j])/2.0)*(-1.0*
        s_depo*f_depo*A_etch)+
        deposit_precursor_gas_concentration_previous_time_step_array[j-1]*((D_depo
        /(2.0*delta_r*delta_r))-(D_depo/(4.0*r*delta_r)))+
        deposit_precursor_gas_concentration_previous_time_step_array[j+1]*((D_depo
        /(2.0*delta_r*delta_r))+(D_depo/(4.0*r*delta_r)));
119         lower_diagonal[j] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))+(D_depo/(4.0*r*
        delta_r));
120         central_diagonal[j] = (1.0/delta_t+s_depo*f_depo*A_depo/2.0+1.0/(2.0*
        t_depo)+(sigma_PE_depo*PE_electron_flux_profile_array[j]+sigma_BSE_depo*
        BSE_electron_flux_profile_array[j]+sigma_SE_depo*
        SE_electron_flux_profile_array[j])/2.0+(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])*(answer[j]+
        etch_precursor_gas_concentration_previous_time_step_array[j])*
        deposit_precursor_reaction_electron_cross_section/4.0)+((2.0*D_depo)/(2.0*
        delta_r*delta_r));
121         upper_diagonal[j] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))-(D_depo/(4.0*r*
        delta_r));
122         answer[j] = 1.0;
123     }
124
125     solve_matrix(number_of_surface_bins,
126                  lower_diagonal,
127                  central_diagonal,
128                  upper_diagonal,
129                  right_hand_side,answer);
130
131     //calculate the Etch Molecule Concentration a second time
```

```
132     right_hand_side[0] = 0.0;
133     answer[0] = 1.0;
134     lower_diagonal[0] = ((-1.0*D_etch)/(2.0*delta_r*delta_r))+(D_etch/(4.0*
        delta_r));
135     central_diagonal[0] = 1.0;
136     upper_diagonal[0] = -1.0;
137     right_hand_side[number_of_surface_bins-1] = 0.0;
138     answer[number_of_surface_bins-1] = 1.0;
139     lower_diagonal[number_of_surface_bins-1] = -1.0;
140     central_diagonal[number_of_surface_bins-1] = 1.0;
141     r = r_array[number_of_surface_bins-1];
142     upper_diagonal[number_of_surface_bins-1] = ((-1.0*D_etch)/(2.0*delta_r*
        delta_r))-(D_etch/(4.0*r*delta_r));
143     for (j=1;j<(number_of_surface_bins-1);j++) {
144         if (j <= no_etch_area) {
145             sigma_PE_etch = 0.0;
146             sigma_BSE_etch = 0.0;
147             sigma_SE_etch = 0.0;
148         } else {
149             sigma_PE_etch = etch_precursor.PE_electron_cross_section;
150             sigma_BSE_etch = etch_precursor.BSE_electron_cross_section;
151             sigma_SE_etch = etch_precursor.SE_electron_cross_section;
152         }
153         r = r_array[j];
154         right_hand_side[j] = s_etch*f_etch+
        etch_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_etch*f_etch*A_etch/2.0-1.0/(2.0*t_etch)-(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])/2.0-((2.0*D_etch)/(2.0*delta_r*delta_r)))
        +((answer[j]+deposit_precursor_gas_concentration_previous_time_step_array[j])
        /2.0)*(-1.0*s_etch*f_etch*A_depo)+
        etch_precursor_gas_concentration_previous_time_step_array[j-1]*((D_etch/(2.0*
        delta_r*delta_r))-(D_etch/(4.0*r*delta_r)))+
        etch_precursor_gas_concentration_previous_time_step_array[j+1]*((D_etch/(2.0*
        delta_r*delta_r))+(D_etch/(4.0*r*delta_r)));
155         lower_diagonal[j] = ((-1.0*D_etch)/(2.0*delta_r*delta_r))+(D_etch/(4.0*r*
        delta_r));
156         central_diagonal[j] = (1.0/delta_t+s_etch*f_etch*A_etch/2.0+1.0/(2.0*
        t_etch)+(sigma_PE_etch*PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])/2.0)+((2.0*D_etch)/(2.0*delta_r*delta_r));
157         upper_diagonal[j] = ((-1.0*D_etch)/(2.0*delta_r*delta_r))-(D_etch/(4.0*r*
        delta_r));
158         answer[j] = 1.0;
159     }
160
161     solve_matrix(number_of_surface_bins,
162                  lower_diagonal,
163                  central_diagonal,
164                  upper_diagonal,
165                  right_hand_side,answer);
166
```

```
167     for (j=0;j<number_of_surface_bins;j++) {
168         etch_precursor_gas_concentration_current_time_step_array[j] = answer[j];
169     }
170 
171     //calculate the Deposit Molecule Concentration a second time
172     right_hand_side[0] = 0.0;
173     answer[0] = 1.0;
174     lower_diagonal[0] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))+(D_depo/(4.0*
        delta_r));
175     central_diagonal[0] = 1.0;
176     upper_diagonal[0] = -1.0;
177     right_hand_side[number_of_surface_bins-1] = 0.0;
178     answer[number_of_surface_bins-1] = 1.0;
179     lower_diagonal[number_of_surface_bins-1] = -1.0;
180     central_diagonal[number_of_surface_bins-1] = 1.0;
181     r = r_array[number_of_surface_bins-1];
182     upper_diagonal[number_of_surface_bins-1] = ((-1.0*D_depo)/(2.0*delta_r*
        delta_r))-(D_depo/(4.0*r*delta_r));
183     for (j=1;j<(number_of_surface_bins-1);j++) {
184         r = r_array[j];
185         right_hand_side[j] = s_depo*f_depo+
        deposit_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_depo*f_depo*A_depo/2.0-1.0/(2.0*t_depo)-(sigma_PE_depo*
        PE_electron_flux_profile_array[j]+sigma_BSE_depo*
        BSE_electron_flux_profile_array[j]+sigma_SE_depo*
        SE_electron_flux_profile_array[j])/2.0-(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])*(answer[j]+
        etch_precursor_gas_concentration_previous_time_step_array[j])*
        deposit_precursor_reaction_electron_cross_section/4.0-((2.0*D_depo)/(2.0*
        delta_r*delta_r)))+((answer[j]+
        etch_precursor_gas_concentration_previous_time_step_array[j])/2.0)*(-1.0*
        s_depo*f_depo*A_etch)+
        deposit_precursor_gas_concentration_previous_time_step_array[j-1]*((D_depo
        /(2.0*delta_r*delta_r))-(D_depo/(4.0*r*delta_r)))+
        deposit_precursor_gas_concentration_previous_time_step_array[j+1]*((D_depo
        /(2.0*delta_r*delta_r))+(D_depo/(4.0*r*delta_r)));
186         lower_diagonal[j] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))+(D_depo/(4.0*r*
        delta_r));
187         central_diagonal[j] = (1.0/delta_t+s_depo*f_depo*A_depo/2.0+1.0/(2.0*
        t_depo)+(sigma_PE_depo*PE_electron_flux_profile_array[j]+sigma_BSE_depo*
        BSE_electron_flux_profile_array[j]+sigma_SE_depo*
        SE_electron_flux_profile_array[j])/2.0+(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])*(answer[j]+
        etch_precursor_gas_concentration_previous_time_step_array[j])*
        deposit_precursor_reaction_electron_cross_section/4.0)+((2.0*D_depo)/(2.0*
        delta_r*delta_r));
188         upper_diagonal[j] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))-(D_depo/(4.0*r*
        delta_r));
189         answer[j] = 1.0;
```

```
190     }
191
192     solve_matrix(number_of_surface_bins,
193                  lower_diagonal,
194                  central_diagonal,
195                  upper_diagonal,
196                  right_hand_side,
197                  answer);
198
199     for (j=0;j<number_of_surface_bins;j++) {
200         deposit_precursor_gas_concentration_current_time_step_array[j] = answer[j
    ];
201     }
202
203     //calculate the Deposited Molecule Concentration
204     for (j=0;j<number_of_surface_bins;j++) {
205         if (j <= no_etch_area) {
206             sigma_PE_etch = 0.0;
207             sigma_BSE_etch = 0.0;
208             sigma_SE_etch = 0.0;
209         } else {
210             sigma_PE_etch = etch_precursor.PE_electron_cross_section;
211             sigma_BSE_etch = etch_precursor.BSE_electron_cross_section;
212             sigma_SE_etch = etch_precursor.SE_electron_cross_section;
213         }
214         if (deposit_precursor.gas_partial_pressure < 0.00001) {
215             reactive_product_concentration_current_time_step_array[j] = -(
    sigma_PE_etch*PE_electron_flux_profile_array[j]+sigma_BSE_etch*
    BSE_electron_flux_profile_array[j]+sigma_SE_etch*
    SE_electron_flux_profile_array[j])*
    etch_precursor_gas_concentration_current_time_step_array[j]*delta_t+
    reactive_product_concentration_previous_time_step_array[j];
216         } else {
217             reactive_product_concentration_current_time_step_array[j] = delta_t
    *(((sigma_PE_depo*PE_electron_flux_profile_array[j]+sigma_BSE_depo*
    BSE_electron_flux_profile_array[j]+sigma_SE_depo*
    SE_electron_flux_profile_array[j])*
    deposit_precursor_gas_concentration_current_time_step_array[j])-((
    sigma_PE_etch*PE_electron_flux_profile_array[j]+sigma_BSE_etch*
    BSE_electron_flux_profile_array[j]+sigma_SE_etch*
    SE_electron_flux_profile_array[j])*
    etch_precursor_gas_concentration_current_time_step_array[j])*(1.0-
    deposit_pinned_reaction_electron_cross_section*
    deposit_precursor_gas_concentration_current_time_step_array[j])*(
    deposit_precursor_reaction_electron_cross_section*
    reactive_product_concentration_previous_time_step_array[j])+(
    reactive_product_concentration_previous_time_step_array[j]/delta_t));
218         }
219     }
220 }
221
222
```

```
223 //calculates the precursor and deposit molecule concentration over time, without
        diffusion
224 void ebied_solver_no_diffusion(Precursor etch_precursor,Precursor
        deposit_precursor,Toggle toggle,double gas_temperature,double
        deposit_pinned_reaction_electron_cross_section,double
        deposit_precursor_reaction_electron_cross_section,double delta_t,double
        delta_r,int number_of_surface_bins) {
225
226     int j;
227
228     double s_etch = etch_precursor.sticking_coefficient;
229     double f_etch = etch_precursor.reactive_product_flux;
230     double A_etch = etch_precursor.surface_area;
231     double t_etch = etch_precursor.desorption_time;
232     double sigma_PE_etch = etch_precursor.PE_electron_cross_section;
233     double sigma_BSE_etch = etch_precursor.BSE_electron_cross_section;
234     double sigma_SE_etch = etch_precursor.SE_electron_cross_section;
235
236     double s_depo = deposit_precursor.sticking_coefficient;
237     double f_depo = deposit_precursor.reactive_product_flux;
238     double A_depo = deposit_precursor.surface_area;
239     double t_depo = deposit_precursor.desorption_time;
240     double sigma_PE_depo = deposit_precursor.PE_electron_cross_section;
241     double sigma_BSE_depo = deposit_precursor.BSE_electron_cross_section;
242     double sigma_SE_depo = deposit_precursor.SE_electron_cross_section;
243
244     for (j=0;j<number_of_surface_bins;j++) {
245         //Etch Molecule Concentration
246         etch_precursor_gas_concentration_current_time_step_array[j] = delta_t*(
        s_etch*f_etch+deposit_precursor_gas_concentration_previous_time_step_array[j
        ]*(-1.0*s_etch*f_etch*A_depo)+
        etch_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_etch*f_etch*A_etch-1.0/t_etch-(sigma_PE_etch*PE_electron_flux_profile_array
        [j]+sigma_BSE_etch*BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])));
247         //Deposit Molecule Concentration
248         deposit_precursor_gas_concentration_current_time_step_array[j] = delta_t
        *(s_depo*f_depo+etch_precursor_gas_concentration_previous_time_step_array[j
        ]*(-1.0*s_depo*f_depo*A_etch)+
        deposit_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_depo*f_depo*A_depo-1.0/t_depo-(sigma_PE_depo*PE_electron_flux_profile_array
        [j]+sigma_BSE_depo*BSE_electron_flux_profile_array[j]+sigma_SE_depo*
        SE_electron_flux_profile_array[j])-(sigma_PE_etch*
        PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])*
        etch_precursor_gas_concentration_previous_time_step_array[j]*
        deposit_precursor_reaction_electron_cross_section));
249         //Deposited Molecule Concentration
```

```
250         reactive_product_concentration_current_time_step_array[j] = delta_t*(((
        sigma_PE_depo*PE_electron_flux_profile_array[j]+sigma_BSE_depo*
        BSE_electron_flux_profile_array[j]+sigma_SE_depo*
        SE_electron_flux_profile_array[j])*
        deposit_precursor_gas_concentration_current_time_step_array[j])-((
        sigma_PE_etch*PE_electron_flux_profile_array[j]+sigma_BSE_etch*
        BSE_electron_flux_profile_array[j]+sigma_SE_etch*
        SE_electron_flux_profile_array[j])*
        etch_precursor_gas_concentration_current_time_step_array[j])*(1.0-
        deposit_pinned_reaction_electron_cross_section*
        deposit_precursor_gas_concentration_current_time_step_array[j])*(
        deposit_precursor_reaction_electron_cross_section*
        reactive_product_concentration_previous_time_step_array[j])+(
        reactive_product_concentration_previous_time_step_array[j]/delta_t));
251     }
252 }
253 
254 //calculates the precursor and deposit molecule concentration over time, EBID
        only
255 void ebid_solver(Precursor deposit_precursor,Toggle toggle,double gas_temperature
        ,double delta_t,double delta_r,int number_of_surface_bins) {
256     #ifdef MPI_ON
257         int node_number;
258         MPI_Comm_rank(MPI_COMM_WORLD, &node_number);
259     #else
260         int node_number = 0;
261     #endif
262 
263     int j;
264     double r;
265 
266     //if surface evo, make an r array a string of point along the surface
267     //this maybe be incorrect and must FIX!!!
268     if (toggle.surface_evolution) {
269         r_array[0] = r_coordinate_array[0];
270         for (j=1;j<number_of_surface_bins;j++) {
271             double dr = r_coordinate_array[j]-r_coordinate_array[j-1];
272             double dz = z_coordinate_array[j]-z_coordinate_array[j-1];
273             r_array[j] = r_array[j-1]+sqrt(dr*dr + dz*dz);
274         }
275     } else {
276         for (j=0;j<number_of_surface_bins;j++) {
277             r_array[j] = r_coordinate_array[j];
278         }
279     }
280 
281     double D_depo = deposit_precursor.diffusion_coefficient;
282     double s_depo = deposit_precursor.sticking_coefficient;
283     double f_depo = deposit_precursor.reactive_product_flux;
284     double A_depo = deposit_precursor.surface_area;
285     double t_depo = deposit_precursor.desorption_time;
286     double sigma_PE_depo = deposit_precursor.PE_electron_cross_section;
287     double sigma_BSE_depo = deposit_precursor.BSE_electron_cross_section;
```

```
288     double sigma_SE_depo = deposit_precursor.SE_electron_cross_section;
289
290    //construct electron flux array
291    for (j=0;j<number_of_surface_bins;j++) {
292       electron_flux_profile_array[j] = sigma_PE_depo*
     PE_electron_flux_profile_array[j]+sigma_BSE_depo*
     BSE_electron_flux_profile_array[j]+sigma_SE_depo*
     SE_electron_flux_profile_array[j];
293    }
294    //fit tails of eflux profile to smooth it out
295    //find r position at 10% of the eflux maximum
296    int r_pos_10percent;
297    for (r_pos_10percent=0;r_pos_10percent<number_of_surface_bins;r_pos_10percent
     ++) {
298       if (electron_flux_profile_array[r_pos_10percent] <
     electron_flux_profile_array[0]/200) {
299          break;
300       }
301    }
302    //find r position at eflux ~= 0.001 e.g. very small
303    int r_pos_0001;
304    for (r_pos_0001=r_pos_10percent;r_pos_0001<number_of_surface_bins;r_pos_0001
     ++) {
305       if (electron_flux_profile_array[r_pos_0001] < 0.01) {
306          break;
307       }
308    }
309    //construct arrays to fit
310    double x_data[r_pos_0001-r_pos_10percent];
311    double y_data[r_pos_0001-r_pos_10percent];
312    for (j=0;j<r_pos_0001-r_pos_10percent;j++) {
313       x_data[j] = r_coordinate_array[j+r_pos_10percent];
314       y_data[j] = log10(electron_flux_profile_array[j+r_pos_10percent]);
315    }
316    //fit data
317    double c0;
318    double c1;
319    double cov00;
320    double cov01;
321    double cov11;
322    double sumsq;
323    gsl_fit_linear(x_data,1,y_data,1,r_pos_0001-r_pos_10percent,&c0,&c1,&cov00,&
     cov01,&cov11,&sumsq);
324
325    //print result to logfile
326    if (node_number == 0) {
327       logfile_printf("# best fit: Y = %g + %g X\n", c0, c1);
328       logfile_printf("# covariance matrix:\n");
329       logfile_printf("# [ %g, %g\n#   %g, %g]\n",cov00, cov01, cov01, cov11);
330       logfile_printf("# sumsq = %g\n", sumsq);
331       logfile_printf("# r0.5 = %g, r0001 = %g\n", r_coordinate_array[
     r_pos_10percent],r_coordinate_array[r_pos_0001]);
332    }
```

```
333
334     //set eflux profile beyond 10% value to fit result
335     for (j=r_pos_10percent;j<r_pos_0001;j++) {
336        electron_flux_profile_array[j] = pow(10.0,c0+c1*r_coordinate_array[j]);
337     }
338     //smooth edges to make the disconnect line up better
339     for (j=-2;j<=2;j++) {
340        electron_flux_profile_array[j+r_pos_10percent] = (
        electron_flux_profile_array[j+r_pos_10percent-2]+electron_flux_profile_array[
        j+r_pos_10percent-1]+electron_flux_profile_array[j+r_pos_10percent]+
        electron_flux_profile_array[j+r_pos_10percent+1]+electron_flux_profile_array[
        j+r_pos_10percent+2])/5.0;
341     }
342     for (j=-2;j<=2;j++) {
343        electron_flux_profile_array[j+r_pos_0001] = (electron_flux_profile_array[j+
        r_pos_0001-2]+electron_flux_profile_array[j+r_pos_0001-1]+
        electron_flux_profile_array[j+r_pos_0001]+electron_flux_profile_array[j+
        r_pos_0001+1]+electron_flux_profile_array[j+r_pos_0001+2])/5.0;
344     }
345
346
347      //calculated EBID, Nd
348      right_hand_side[0] = 0.0;
349      answer[0] = 1.0;
350      lower_diagonal[0] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))+(D_depo/(4.0*
        delta_r));
351      central_diagonal[0] = 1.0;
352      upper_diagonal[0] = -1.0;
353      right_hand_side[number_of_surface_bins-1] = 0.0;
354      answer[number_of_surface_bins-1] = 1.0;
355      lower_diagonal[number_of_surface_bins-1] = -1.0;
356      central_diagonal[number_of_surface_bins-1] = 1.0;
357      r = r_array[number_of_surface_bins-1];
358      upper_diagonal[number_of_surface_bins-1] = ((-1.0*D_depo)/(2.0*delta_r*
        delta_r))-(D_depo/(4.0*r*delta_r));
359      for (j=1;j<(number_of_surface_bins-1);j++) {
360          r = r_array[j];
361          right_hand_side[j] = s_depo*f_depo+
        deposit_precursor_gas_concentration_previous_time_step_array[j]*(1.0/delta_t-
        s_depo*f_depo*A_depo/2.0-1.0/(2.0*t_depo)-electron_flux_profile_array[j
        ]/2.0-((2.0*D_depo)/(2.0*delta_r*delta_r)))+
        deposit_precursor_gas_concentration_previous_time_step_array[j-1]*((D_depo
        /(2.0*delta_r*delta_r))-(D_depo/(4.0*r*delta_r)))+
        deposit_precursor_gas_concentration_previous_time_step_array[j+1]*((D_depo
        /(2.0*delta_r*delta_r))+(D_depo/(4.0*r*delta_r)));
362          lower_diagonal[j] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))+(D_depo/(4.0*r*
        delta_r));
363          central_diagonal[j] = (1.0/delta_t+s_depo*f_depo*A_depo/2.0+1.0/(2.0*
        t_depo)+electron_flux_profile_array[j]/2.0+((2.0*D_depo)/(2.0*delta_r*delta_r
        )));
364          upper_diagonal[j] = ((-1.0*D_depo)/(2.0*delta_r*delta_r))-(D_depo/(4.0*r*
        delta_r));
365          answer[j] = 1.0;
```

```
366      }
367      solve_matrix(number_of_surface_bins,
368                   lower_diagonal,
369                   central_diagonal,
370                   upper_diagonal,
371                   right_hand_side,
372                   answer);
373
374      //calculate ND
375      for (j=0;j<number_of_surface_bins;j++) {
376          deposit_precursor_gas_concentration_current_time_step_array[j] = answer[j
         ];
377          reactive_product_concentration_current_time_step_array[j] =
         electron_flux_profile_array[j]*
         deposit_precursor_gas_concentration_current_time_step_array[j]*delta_t+
         reactive_product_concentration_previous_time_step_array[j];
378      }
379  }
```

B.6 Electron Flux Profile

```
1   //
2   //  electron_flux_profile.c
3   //
4   //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5   //
6   //  This is the module which calculates the electron flux profile with a shape of
         a top hat or gaussian and with projection turned on or off, for the EBIED
        Simulator.
7   //
8   #ifdef XCODE
9   #include <stdlib.h>
10  #include <stdio.h>
11  #include <math.h>
12  #include "structures.h"
13  #include "constants.h"
14  #include "prototypes.h"
15  #endif
16
17  void electron_flux_profile(Toggle toggle,
18                             Electron_beam electron_beam,
19                             int number_of_surface_bins,
20                             double delta_t,
21                             double delta_r,
22                             double backscattered_electron_coefficient,
23                             double secondary_electron_coefficient,int i) {
24
25      int j;
26      double sum_primary;
27      double sum_backscattered;
```

```
28      double sum_secondary;
29      double n_electrons_per_sec;
30
31      if (toggle.electron_beam_shape) {
32          // top hat shape
33          // FIXME: correct area (Jared)
34          sum_primary = 0.0;
35      } else {
36          // Gaussian
37          sum_primary = PI*pow((r_coordinate_array[1]-r_coordinate_array[0])
        /2.0,2.0);
38      }
39      sum_backscattered = 1.0;  // prevent "divide by zero" in normalization
40      sum_secondary = 1.0;      // prevent "divide by zero" in normalization
41
42      // define the beam (the primary electron array)
43      for (j=0;j<number_of_surface_bins;j++) {
44          if (toggle.electron_beam_shape) {
45              //top hat beam shape
46              temp_primary_electron_array[j] = 1.0/(exp(electron_beam.
        top_hat_abruptness*(2.0*r_coordinate_array[j]/electron_beam.diameter-1.0))
        +1.0);
47          } else {
48              //gaussian beam shape
49              temp_primary_electron_array[j] = exp(-1.0*pow(r_coordinate_array[j
        ],2.0)/pow(electron_beam.diameter/2.0,2.0));
50          }
51
52          double r1;
53          double r2;
54          if(j==0) {
55              r1 = r_coordinate_array[j] - delta_r/2.0;
56              r2 = (r_coordinate_array[j+1]+r_coordinate_array[j])/2.0;
57          }
58          else if(j == number_of_surface_bins-1) {
59              r1 = (r_coordinate_array[j]+r_coordinate_array[j-1])/2.0;
60              r2 = r_coordinate_array[j] + delta_r/2.0;
61          } else {
62              r1 = (r_coordinate_array[j]+r_coordinate_array[j-1])/2.0;
63              r2 = (r_coordinate_array[j+1]+r_coordinate_array[j])/2.0;
64          }
65          double annular_area = PI*(r2*r2 - r1*r1);
66          if (annular_area <= 0.0) {
67              if (toggle.electron_beam_projection) {
68                  annular_area = 0.0;
69              } else {
70                  annular_area = abs(PI*(r2*r2 - r1*r1));
71              }
72          }
73          primary_electron_array[j] = annular_area*temp_primary_electron_array[j];
74      }
75
76      // copy locations
```

```
77      for (j=0;j<number_of_surface_bins;j++) {
78          backscattered_electron_array[j] = backscattered_electron_location_array[j
        ];
79          secondary_electron_array[j] = secondary_electron_location_array[j];
80      }
81
82      // compute sums
83      for (j=0;j<number_of_surface_bins;j++) {
84          sum_primary += primary_electron_array[j];
85          sum_backscattered += backscattered_electron_array[j];
86          sum_secondary += secondary_electron_array[j];
87      }
88
89      // normalize arrays
90      for (j=0;j<number_of_surface_bins;j++) {
91          primary_electron_array[j] = (primary_electron_array[j]/sum_primary)*
        electron_beam.current;
92          backscattered_electron_array[j] = (backscattered_electron_array[j]/
        sum_backscattered)*electron_beam.current*backscattered_electron_coefficient;
93          secondary_electron_array[j] = (secondary_electron_array[j]/sum_secondary)
        *electron_beam.current*secondary_electron_coefficient;
94      }
95
96      // prepare for the MC algorithm
97      for (j=0;j<number_of_surface_bins;j++) {
98          temp_primary_electron_flux_profile_array[j] = primary_electron_array[j];
99      }
100
101     // transform back to "normal space" from "annular space"
102     for (j=0;j<number_of_surface_bins;j++) {
103         double r1;
104         double r2;
105         double sin90 = sin(90*PI/180.0);
106         double distc,dista;
107         double sinA = 1.0;
108         if(j==0) {
109             r1 = r_coordinate_array[j] - delta_r/2.0;
110             r2 = (r_coordinate_array[j+1]+r_coordinate_array[j])/2.0;
111         }
112         else if(j == number_of_surface_bins-1) {
113             r1 = (r_coordinate_array[j]+r_coordinate_array[j-1])/2.0;
114             r2 = r_coordinate_array[j] + delta_r/2.0;
115         } else {
116             r1 = (r_coordinate_array[j]+r_coordinate_array[j-1])/2.0;
117             r2 = (r_coordinate_array[j+1]+r_coordinate_array[j])/2.0;
118         }
119         if (toggle.electron_beam_projection) {
120             //Surface Projection On
121             if(j==0) {
122                 dista = r_coordinate_array[j+1]-r_coordinate_array[j];
123                 distc = sqrt(pow(z_coordinate_array[j+1]-z_coordinate_array[j
        ],2.0)+pow(r_coordinate_array[j+1]-r_coordinate_array[j],2.0));
124                 sinA = (sin90/distc*dista);
```

```
125             } else if(j == number_of_surface_bins-1) {
126                 sinA = 1.0;
127             } else {
128                 dista = r_coordinate_array[j+1]-r_coordinate_array[j-1];
129                 distc = sqrt(pow(z_coordinate_array[j+1]-z_coordinate_array[j
    -1],2.0)+pow(r_coordinate_array[j+1]-r_coordinate_array[j-1],2.0));
130                 sinA = (sin90/distc*dista);
131             }
132         }
133         double annular_area;
134         if (sinA < 0.0) {
135             annular_area = PI*(r1*r1 - r2*r2);
136         } else {
137             annular_area = PI*(r2*r2 - r1*r1);
138         }
139         if (annular_area <= 0.0) {
140             annular_area = 1.0;
141         }
142         n_electrons_per_sec = primary_electron_array[j]*sinA +
    backscattered_electron_array[j] + secondary_electron_array[j];
143 
144         PE_electron_flux_profile_array[j] = primary_electron_array[j]*sinA/
    annular_area;
145         BSE_electron_flux_profile_array[j] = backscattered_electron_array[j]/
    annular_area;
146         SE_electron_flux_profile_array[j] = secondary_electron_array[j]/
    annular_area;
147         electron_flux_profile_array[j] = PE_electron_flux_profile_array[j]+
    BSE_electron_flux_profile_array[j]+SE_electron_flux_profile_array[j];
148     }
149     //flaten first bin when MC is ON
150     if (toggle.electron_trajectory_simulator) {
151         electron_flux_profile_array[0] = electron_flux_profile_array[1];
152     }
153 
154     // compute cumulative sum of the flux profile (probability distribution of
    how likely an electron is to enter the substrate from the primary beam
155     temp_electron_starting_location_probability_array[0] =
    temp_primary_electron_flux_profile_array[0];
156     for (j=1;j<number_of_surface_bins;j++) {
157         temp_electron_starting_location_probability_array[j] =
    temp_primary_electron_flux_profile_array[j]+
    temp_electron_starting_location_probability_array[j-1];
158     }
159     // normalize from 0 to 1
160     for (j=0;j<number_of_surface_bins;j++) {
161         electron_starting_location_probability_array[j] =
    temp_electron_starting_location_probability_array[j]/
    temp_electron_starting_location_probability_array[number_of_surface_bins-1];
162     }
163 }
```

B.7 Linear Interpolation

```
//
//  interpolation.c
//
//  Created by Jared Cullen on 26/03/2012.
//  Last Updated by Jared Cullen on 19/11/2013.
//  Copyright (c) 2013 University of Technology, Sydney. All rights reserved.
//
//  This function performs linear interpolation on the input arrays for the EBIED
     Simulator.
//
#ifdef __MACH__
#include <math.h>
#include <stdio.h>
#include <assert.h>
#include "constants.h"
#include "structures.h"
#include "prototypes.h"
#endif

void primary_interpolation(double *r[],
                           double *z[],
                           int length,
                           double delta_r,
                           int number_of_points) {
    int i;
    double r1,r2,r3,z1,z2,z3,angle,grad;
    double distance;
    double cos_d, sin_d;

    i = 0;
    (*r)[i] = delta_r;

   i = 1;
   r1 = (*r)[i-1];
        r2 = (*r)[i];
        z1 = (*z)[i-1];
        z2 = (*z)[i];
        distance = sqrt(pow(z2-z1,2.0)+pow(r2-r1,2.0));
        grad = (z2-z1)/(r2-r1);
   angle = atan(grad);
        cos_d = cos(angle)*(delta_r-distance);
        sin_d = sin(angle)*(delta_r-distance);
        (*r)[i] = cos_d+r2;
        (*z)[i] = sin_d+z2;

    for (i=2;i<length-1;i++) {
        r1 = (*r)[i-1];
        r2 = (*r)[i];
   r3 = (*r)[i+1];
        z1 = (*z)[i-1];
        z2 = (*z)[i];
```

```
51    z3 = (*z)[i+1];
52         distance = sqrt(pow(z2-z1,2.0)+pow(r2-r1,2.0));
53         //grad = (z2-z1)/(r2-r1);
54    grad = (3*(z1*r1+z2*r2+z3*r3)-(r3+r2+r1)*(z3+z2+z1))/(3*(r1*r1+r2*r2+r3*r3)-(
       r1+r2+r3)*(r1+r2+r3));
55         angle = atan(grad);
56         cos_d = cos(angle)*(delta_r-distance);
57         sin_d = sin(angle)*(delta_r-distance);
58         (*r)[i] = cos_d+r2;
59         (*z)[i] = sin_d+z2;
60     }
61
62    i = length-1;
63    r1 = (*r)[i-1];
64         r2 = (*r)[i];
65         z1 = (*z)[i-1];
66         z2 = (*z)[i];
67         distance = sqrt(pow(z2-z1,2.0)+pow(r2-r1,2.0));
68         grad = (z2-z1)/(r2-r1);
69    angle = atan(grad);
70         cos_d = cos(angle)*(delta_r-distance);
71         sin_d = sin(angle)*(delta_r-distance);
72         (*r)[i] = cos_d+r2;
73         (*z)[i] = sin_d+z2;
74 }
75
76 void secondary_interpolation(double r[],double *z[],double z_orig[],double
       r_interp[],int length,int number_of_points,int order,double delta_r) {
77     int p1 = (number_of_points-1)/2;
78     int p2 = (number_of_points-1)/2;
79     int i;
80     double z_grad;
81     double z_temp,r_temp;
82
83     for (i=1;i<p1;i++) {
84         z_temp = z_orig[-(i-p1)];
85         r_temp = -r[-(i-p1)];
86         z_grad = (z_orig[i+p2]-z_temp)/(r[i+p2]-r_temp);
87         (*z)[i] = z_grad*(r_interp[i]-r[i])+(*z)[i];
88     }
89     for (i=p1;i<length;i++) {
90         if (i+p2 > length) {
91             p2--;
92         }
93         z_grad = (z_orig[i+p2]-z_orig[i-p1])/(r[i+p2]-r[i-p1]);
94         (*z)[i] = z_grad*(r_interp[i]-r[i])+(*z)[i];
95     }
96 }
```

B.8 Monte Carlo Data Collection

```c
//
//  monte_carlo_data_collection.c
//
//  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
//
//  this module collects all the data from each node together and redistributes this data back.
#ifdef XCODE
#include <stdlib.h>
#include <stdio.h>
#include "structures.h"
#include "prototypes.h"
#ifdef MPI_ON
    #include <mpi.h>
#endif
#endif

void monte_carlo_data_collection(int length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,int number_of_electron_trajectories,int number_of_surface_bins,int number_of_backscattered_electrons,double *backscattered_electron_coefficient, double *secondary_electron_coefficient) {

    int i,j,k;
    double secondary_electron_location_sum;//,k;

    #ifdef MPI_ON
        //collect data from all nodes
        //collect,number_of_backscattered_electrons
        int local_number_of_backscattered_electrons = number_of_backscattered_electrons;
        int sum_number_of_backscattered_electrons = 0;
        MPI_Reduce(&local_number_of_backscattered_electrons,&sum_number_of_backscattered_electrons,1,MPI_INT,
                   MPI_SUM,0,MPI_COMM_WORLD);
        MPI_Bcast(&sum_number_of_backscattered_electrons,1,MPI_INT,0,MPI_COMM_WORLD);
        number_of_backscattered_electrons = sum_number_of_backscattered_electrons;

        //collect,secondary_electron_location_array
        for (j=0;j<number_of_surface_bins;j++) {
            local_secondary_electron_location_array[j] = secondary_electron_location_array[j];
            sum_secondary_electron_location_array[j] = 0.0;
        }
        MPI_Reduce(local_secondary_electron_location_array,sum_secondary_electron_location_array,number_of_surface_bins,MPI_DOUBLE,
                   MPI_SUM,0,MPI_COMM_WORLD);
        MPI_Bcast(sum_secondary_electron_location_array,number_of_surface_bins,MPI_DOUBLE,0,MPI_COMM_WORLD);
        for (j=0;j<number_of_surface_bins;j++) {
```

```
41            secondary_electron_location_array[j] =
    sum_secondary_electron_location_array[j];
42        }
43
44        //collect,backscattered_electron_location_array
45        for (j=0;j<number_of_surface_bins;j++) {
46            local_backscattered_electron_location_array[j] =
    backscattered_electron_location_array[j];
47            sum_backscattered_electron_location_array[j] = 0.0;
48        }
49        MPI_Reduce(local_backscattered_electron_location_array,
    sum_backscattered_electron_location_array,number_of_surface_bins,MPI_DOUBLE,
    MPI_SUM,0,MPI_COMM_WORLD);
50        MPI_Bcast(sum_backscattered_electron_location_array,
    number_of_surface_bins,MPI_DOUBLE,0,MPI_COMM_WORLD);
51        for (j=0;j<number_of_surface_bins;j++) {
52            backscattered_electron_location_array[j] =
    sum_backscattered_electron_location_array[j];
53        }
54
55        //collect,electron_energy_deposited_array
56        for (j=0;j<
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array;j++) {
57            local_electron_energy_deposited_array[j] =
    electron_energy_deposited_array[j];
58            sum_electron_energy_deposited_array[j] = 0.0;
59        }
60        MPI_Reduce(local_electron_energy_deposited_array,
    sum_electron_energy_deposited_array,
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
61        MPI_Bcast(sum_electron_energy_deposited_array,
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    MPI_DOUBLE,0,MPI_COMM_WORLD);
62        for (j=0;j<
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array;j++) {
63            electron_energy_deposited_array[j] =
    sum_electron_energy_deposited_array[j];
64        }
65
66        //collect,electron_energy_deposited_array
67        for (j=0;j<
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array;j++) {
68            local_electron_maximum_z_depth_array[j] =
    electron_maximum_z_depth_array[j];
69            sum_electron_maximum_z_depth_array[j] = 0.0;
70        }
71        MPI_Reduce(local_electron_maximum_z_depth_array,
    sum_electron_maximum_z_depth_array,
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
```

```
72          MPI_Bcast(sum_electron_maximum_z_depth_array,
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
        MPI_DOUBLE,0,MPI_COMM_WORLD);
73          for (j=0;j<
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array;j++) {
74              electron_maximum_z_depth_array[j] =
        sum_electron_maximum_z_depth_array[j];
75          }
76      #endif
77
78      //calculated SE yield and BSE coeff
79      secondary_electron_location_sum = 0.0;
80      for (j=0;j<number_of_surface_bins;j++) {
81          secondary_electron_location_sum += secondary_electron_location_array[j];
82      }
83      *secondary_electron_coefficient = secondary_electron_location_sum/(double)
        number_of_electron_trajectories;
84      *backscattered_electron_coefficient = (double)
        number_of_backscattered_electrons/(double)number_of_electron_trajectories;
85
86      //smooth SE and BSE profiles, to an 11 point window
87      k = 0;
88      for (i=0;i<number_of_surface_bins;i++) {
89          sum_secondary_electron_location_array[i] = 0;
90          sum_backscattered_electron_location_array[i] = 0;
91          //k = 0;
92          for (j=-k;j<=k;j++) {
93              if (i+j >= 0 && i+j < number_of_surface_bins) {
94                  sum_secondary_electron_location_array[i] +=
        secondary_electron_location_array[i+j];
95                  sum_backscattered_electron_location_array[i] +=
        backscattered_electron_location_array[i+j];
96                  //k++;
97              }
98          }
99          if (i == 0) {
100             sum_secondary_electron_location_array[i] +=
        secondary_electron_location_array[i];
101             sum_backscattered_electron_location_array[i] +=
        backscattered_electron_location_array[i];
102         }
103         sum_secondary_electron_location_array[i] =
        sum_secondary_electron_location_array[i]/(k*2+1);
104         sum_backscattered_electron_location_array[i] =
        sum_backscattered_electron_location_array[i]/(k*2+1);
105         k++;
106         if (k > 5) {
107             k = 5;
108         }
109     }
110     for (i=0;i<number_of_surface_bins;i++) {
111         secondary_electron_location_array[i] =
        sum_secondary_electron_location_array[i];
```

```
112         backscattered_electron_location_array[i] =
    sum_backscattered_electron_location_array[i];
113     }
114 }
```

B.9 Monte Carlo Electron Trajectories

```
1  //
2  //  monte_carlo_electron_trajectory_simulator.c
3  //
4  //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5  //
6  //  This is the module which performs monte carlo electron trajectory simulation
    turned on or off, and electron tracking on or off for the EBIED Simulator.
7  //
8  #ifdef XCODE
9  #include "structures.h"
10 #include "constants.h"
11 #include "prototypes.h"
12 #include <sys/time.h>
13 #include <stdlib.h>
14 #include <stdio.h>
15 #include <math.h>
16 #include <assert.h>
17 
18 #ifdef MPI_ON
19     #include <mpi.h>
20 #endif
21 #endif
22 
23 //check if trajectory crossed into a segment
24 void segments_intersect(double x11, double y11, double x12, double y12,
25                         double x21, double y21, double x22, double y22,
26                         int *intersection,double *x,double *y) {
27     // check if two segments intersect, return 1 if true, 0 if false
28     //outputs the intersection x,y coordinates
29 
30     //start as false
31     *intersection = 0; // false
32 
33     int count = 0;
34 
35     double A1 = y12-y11;
36     double B1 = x11-x12;
37     double C1 = A1*x11+B1*y11;
38 
39     double A2 = y22-y21;
40     double B2 = x21-x22;
41     double C2 = A2*x21+B2*y21;
42 
```

```
43      double det = A1*B2 - A2*B1;
44      if(det != 0) {
45          // lines intersect at x and y
46          *x = (B2*C1 - B1*C2)/det;
47          *y = (A1*C2 - A2*C1)/det;
48
49          // order the points small to large
50          double xx1 = fmin(x21, x22);
51          double xx2 = fmax(x21, x22);
52          double yy1 = fmin(y21, y22);
53          double yy2 = fmax(y21, y22);
54
55          // check if intersection is within segment
56          if (xx1 < *x) {
57              count++;
58          } else if (xx1-*x < 0.00001 && -0.00001 < xx1-*x) {
59              count++;
60          }
61          if (*x < xx2) {
62              count++;
63          } else if (xx2-*x < 0.00001 && -0.00001 < xx2-*x) {
64              count++;
65          }
66          if (yy1 < *y) {
67              count++;
68          } else if (yy1-*y < 0.00001 && -0.00001 < yy1-*y) {
69              count++;
70          }
71          if (*y < yy2) {
72              count++;
73          } else if (yy2-*y < 0.00001 && -0.00001 < yy2-*y) {
74              count++;
75          }
76          if (count == 4) {
77              *intersection = 1; // true
78          }
79      }
80  }
81
82  //checks if electron is leaving the surface
83  void is_leaving_volume(double z1,
84                         double z2,
85                         double r1,
86                         double r2,
87                         int number_of_surface_bins,
88                         double* z,
89                         double* r,
90                         int *outputBinIndex,// 0 not leaving, 1 and above is bin+1
91                         double *output_r,
92                         double *output_z)   {
93      int intersection = 0;
94      double intersection_r,intersection_z;
95
```

```
96      // walk the whole surface to check for intersections
97      int i;
98      int start = 0;
99      int end = number_of_surface_bins-1;
100     for(i=start; i<end; i++) {
101         // check if electron trajectory crosses surface segment
102         segments_intersect(r1,z1,r2,z2,r[i],z[i],r[i+1],z[i+1],&intersection,&
        intersection_r,&intersection_z);
103         if(intersection) {
104             if (z1 < intersection_z && intersection_z < z2) {
105                 *outputBinIndex = i+1;
106                 *output_r = intersection_r;
107                 *output_z = intersection_z;
108                 break;//output where intersection occured, index,r,z
109             }
110         }
111     }
112 }
113
114 //checks if electron is re-entering surface
115 void is_reentering_volume(double z1,
116                           double z2,
117                           double r1,
118                           double r2,
119                           int number_of_surface_bins,
120                           double* z,
121                           double* r,
122                           int *outputBinIndex,// 0 not leaving, 1 and above is bin
        +1
123                           double *output_r,
124                           double *output_z) {
125     int intersection = 0;
126     double intersection_r,intersection_z;
127
128     // walk the whole surface to check for intersections
129     int i;
130     if (r2 < r1) {
131         for(i=*outputBinIndex; i>=0; i--) { //start from exit point +1
132             // check if electron trajectory crosses surface segment
133             segments_intersect(r1,z1,r2,z2,r[i],z[i],r[i+1],z[i+1],&intersection
        ,&intersection_r,&intersection_z);
134             if(intersection) {
135                 *outputBinIndex = i+1;
136                 *output_r = intersection_r;
137                 *output_z = intersection_z;
138                 break;//output where reentry occured, index,r,z
139             }
140         }
141     } else {
142         for(i=*outputBinIndex; i<number_of_surface_bins-1; i++) { //start from
        exit point +1
143             // check if electron trajectory crosses surface segment
```

```
144             segments_intersect(r1,z1,r2,z2,r[i],z[i],r[i+1],z[i+1],&intersection
    ,&intersection_r,&intersection_z);
145             if(intersection) {
146                 *outputBinIndex = i+1;
147                 *output_r = intersection_r;
148                 *output_z = intersection_z;
149                 break;//output where reentry occured, index,r,z
150             }
151         }
152     }
153
154 }
155
156 //main MC module
157 void monte_carlo_electron_trajectory_simulator(Electron_beam electron_beam,
158                                                Material lower_material,
159                                                Material upper_material,
160                                                int electron_trajectory_tracking,
161                                                double delta_r,
162                                                int
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
163                                                int number_of_surface_bins,
164                                                double
    layered_material_interface_depth,
165                                                int maximum_electron_track_length,
166                                                int
    number_of_electron_trajectories,
167                                                double z_depth_minimum,
168                                                double z_depth_maximum,
169                                                int *
    number_of_backscattered_electrons,
170                                                int seed) {
171
172     // Refer to the book by David Joy (1995)
173     //      "Monte carlo modelling for electron microscopy and microanalysis"
174     //      Chapter 3 - Single Scattering Model
175     //
176     #ifdef MPI_ON
177         int node_number,total_number_of_nodes;
178         MPI_Comm_rank(MPI_COMM_WORLD, &node_number);
179         MPI_Comm_size(MPI_COMM_WORLD, &total_number_of_nodes);
180     #else
181         int node_number = 0;
182         int total_number_of_nodes = 1;
183     #endif
184
185     //set random number seed
186     struct timeval t1;
187     gettimeofday(&t1, NULL);
188     if (seed < 0) {
189         #ifdef MPI_ON
190             srand((unsigned)(t1.tv_usec * t1.tv_sec)*node_number);
191         #else
```

```
192             srand((unsigned)(t1.tv_usec * t1.tv_sec));
193         #endif
194     } else {
195         #ifdef MPI_ON
196             srand(seed*node_number);
197         #else
198             srand(seed);
199         #endif
200     }
201 
202     int i;
203     int current_electron_number = 1;
204     int electron_tracking_counter;
205     int number_of_electron_trajectories_per_node = (int)floor((double)
        number_of_electron_trajectories/(double)total_number_of_nodes);
206     int number_of_intersections,intersection_index1,intersection_index2,
        lost_electrons = 0;
207     double at_num,at_wht,density,sg_a,al_a,lam_a,s_en,x,y,z,xn,yn,zn,cx,cy,cz,sp,
        ga,cp,ca,cb,cc,lambda,lambda1,lambda2;
208     double electron_step_length,electron_step_length1,electron_step_length2,
        random_number,maximum_electron_z_depth,exit_x1,exit_x2,exit_y1,exit_y2,
        vaccum_distance;
209     double SE_epsilon,SE_lambda;
210     Material material;
211 
212     // set values initially
213     number_of_intersections = 0;
214     exit_x1 = exit_x2 = exit_y1 = exit_y2 = number_of_surface_bins - 1;
215     intersection_index1 = intersection_index2 = number_of_surface_bins - 1;
216 
217     // reset input arrays
218     maximum_electron_z_depth = 0;
219     *number_of_backscattered_electrons = 0;
220     for (i=0;i<number_of_surface_bins;i++) {
221         secondary_electron_location_array[i] = 0.0;
222         backscattered_electron_location_array[i] = 0.0;
223     }
224     for (i=0;i<
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array;i++) {
225         electron_energy_deposited_array[i] = 0.0;
226         electron_maximum_z_depth_array[i] = 0.0;
227     }
228 
229     //
230     if ((node_number == 0) && electron_trajectory_tracking == 1) {
231         FILE *fp;
232         fp=fopen("./outputs/output_track.txt", "w");
233         fclose(fp);
234     }
235     // loop over every electron requested
236     while (current_electron_number <= number_of_electron_trajectories_per_node) {
237 
238         // re-clear electron tracking array
```

```
239         if (electron_trajectory_tracking == 1) {
240             for (i=0;i<maximum_electron_track_length;i++) {
241                 electron_tracking_x_position_array[i] = 0.0;
242                 electron_tracking_y_position_array[i] = 0.0;
243                 electron_tracking_z_position_array[i] = 0.0;
244             }
245             electron_tracking_counter = 0;
246         }
247 
248         // setup the material at surface current depth into material
249         //  (if we have etched deeply, we might be in the lower material)
250         if (z_coordinate_array[0] <= layered_material_interface_depth) {
251             material = lower_material;
252         } else {
253             material = upper_material;
254         }
255         at_num = material.atomic_number;
256         at_wht = material.atomic_weight;
257         density = material.density;
258         SE_epsilon = material.epsilon;
259         SE_lambda = material.lambda;
260         get_constants(&sg_a,&al_a,&lam_a,at_num,at_wht,density,electron_beam.
    energy);
261 
262         // find the starting point (taking into account surface height, and beam
    probability)
263         //where the electron comes from into the surface
264         reset_coordinates(&s_en,&x,&y,&z,&cx,&cy,&cz,electron_beam.energy,
    number_of_surface_bins);
265 
266         if (electron_trajectory_tracking == 1) {
267             electron_tracking_x_position_array[electron_tracking_counter] = x;
268             electron_tracking_y_position_array[electron_tracking_counter] = y;
269             electron_tracking_z_position_array[electron_tracking_counter] = z;
270             electron_tracking_counter++;
271         }
272 
273         //the first scattering event (handles 0 - 89.9 tilted electron beam)
274         random_number = rand();
275         lambda = compute_lambda(s_en,al_a,sg_a,lam_a);
276         electron_step_length = -lambda*log(random_number/((double)RAND_MAX+1));
277         cp = cos((PI/180.0)*electron_beam.tilt);//angle of electron beam tilt
278         sp = sqrt(1.0-cp*cp);
279         ga = 2.0*PI*cos((PI/180.0)*0.0);//azimuthal scattering angle = 0
280         new_coord(electron_step_length,x,y,z,cx,cy,cz,sp,ga,cp,&ca,&cb,&cc,&xn,&
    yn,&zn);
281         reset_next_step(ca,cb,cc,xn,yn,zn,electron_step_length,&cx,&cy,&cz,&x,&y
    ,&z,&s_en,density,at_num,at_wht,SE_epsilon,SE_lambda,number_of_intersections,
    exit_x1,exit_y1,exit_x2,exit_y2,
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    delta_r,number_of_surface_bins,intersection_index1);
282 
283         if (electron_trajectory_tracking == 1) {
```

```
284             electron_tracking_x_position_array[electron_tracking_counter] = x;
285             electron_tracking_y_position_array[electron_tracking_counter] = y;
286             electron_tracking_z_position_array[electron_tracking_counter] = z;
287             electron_tracking_counter++;
288         }
289 
290         // now electron is in the substrate
291         // iterate until it loses energy or exits surface
292         number_of_intersections = 0; //electron has not intersected the surface
293         while ((s_en > electron_beam.cutoff_energy) && (number_of_intersections
    == 0)) {
294             if (zn >= layered_material_interface_depth) {
295                 material = lower_material;
296             } else {
297                 material = upper_material;
298             }
299             at_num = material.atomic_number;
300             at_wht = material.atomic_weight;
301             density = material.density;
302             SE_epsilon = material.epsilon;
303             SE_lambda = material.lambda;
304             get_constants(&sg_a,&al_a,&lam_a,at_num,at_wht,density,electron_beam.
    energy);
305 
306             // compute scattering event
307             lambda = compute_lambda(s_en,al_a,sg_a,lam_a);
308             random_number = rand();
309             electron_step_length = -lambda*log(random_number/((double)RAND_MAX+1)
    );
310             s_scatter(s_en,al_a,&sp,&ga,&cp);
311             new_coord(electron_step_length,x,y,z,cx,cy,cz,sp,ga,cp,&ca,&cb,&cc,&
    xn,&yn,&zn);
312 
313             // if electron leaves simulation then record lost_electrons += 1 and
    exit
314             double r1 = sqrt(pow(x,2)+pow(y,2));
315             double rn1 = sqrt(pow(xn,2)+pow(yn,2));
316             if (rn1 > (number_of_surface_bins*delta_r)) {
317                 lost_electrons++;
318                 break;
319             }
320             // if electron is near the surface, check for surface exit
321             if (-zn >= z_depth_minimum) {
322                 int electron_left_volume = 0;
323                 double electron_left_r,electron_left_z;
324                 is_leaving_volume(-z,-zn,r1,rn1,number_of_surface_bins*2,
    z_coordinate_both_directions,r_coordinate_both_directions,
325                                  &electron_left_volume,&electron_left_r,&
    electron_left_z);//did an electron leave and where
326                 if (electron_left_volume) {
327                     int electron_reentered_volume = electron_left_volume;
328                     double electron_reentered_r,electron_reentered_z;
```

```
329                     is_reentering_volume(-z,-zn,r1,rn1,number_of_surface_bins*2,
    z_coordinate_both_directions,r_coordinate_both_directions,
330                                          &electron_reentered_volume,&
    electron_reentered_r,&electron_reentered_z);//did an electron reenter and
    where
331                     if (electron_left_volume > number_of_surface_bins) {
332                         electron_left_volume = electron_left_volume-
    number_of_surface_bins;
333                     } else {
334                         electron_left_volume = number_of_surface_bins-
    electron_left_volume;
335                     }
336                     if (electron_reentered_volume > number_of_surface_bins) {
337                         electron_reentered_volume = electron_reentered_volume-
    number_of_surface_bins;
338                     } else {
339                         electron_reentered_volume = number_of_surface_bins-
    electron_reentered_volume;
340                     }
341                     //*outputBinIndex = i+1;
342                     if (electron_reentered_volume == electron_left_volume) {
343                          //one intersection records the backscattered electron
344                         *number_of_backscattered_electrons += 1;
345                         maximum_electron_z_depth = 0.0;
346                         //electron_left_volume = ceil(electron_left_volume/
    delta_r);
347                         backscattered_electron_location_array[
    electron_left_volume-1] += 1.0;
348                         number_of_intersections = 1;//electron has BS
349                     } else {
350                         //two intersections adjusts the trajectory for reentry
351                         //electron_left_volume = ceil(electron_left_volume/
    delta_r);
352                         //electron_reentered_volume = ceil(
    electron_reentered_volume/delta_r);
353                         backscattered_electron_location_array[
    electron_left_volume-1] += 1.0;
354                         backscattered_electron_location_array[
    electron_reentered_volume-1] += 1.0;
355                         vaccum_distance = sqrt(pow(electron_reentered_r-
    electron_left_r,2.0)+pow(electron_reentered_z-electron_left_z,2.0));
356                         xn = x+(electron_step_length+vaccum_distance)*ca;
357                         yn = y+(electron_step_length+vaccum_distance)*cb;
358                         zn = z+(electron_step_length+vaccum_distance)*cc;
359                         //printf("xn:%g,yn:%g,zn:%g,x:%g,y:%g,z:%g\n",xn,yn,zn,x,
    y,z);
360                         number_of_intersections = 2;//electron has FS
361                         //temporary code to allow compatibility with old code
    until testing complete
362                         exit_x2 = electron_reentered_r;
363                         exit_y2 = electron_reentered_z;
364                         intersection_index2 = electron_reentered_volume-1;
365                         //
```

```
366                     }
367                     //temporary code to allow compatibility with old code until
    testing complete
368                     exit_x1 = electron_left_r;
369                     exit_y1 = electron_left_z;
370                     intersection_index1 = electron_left_volume-1;
371                     //
372                 } else {
373                     number_of_intersections = 0;//electron in material
374                 }
375             }
376 
377             // if we have crossed from one material to the next
378             if ((-z < layered_material_interface_depth) && (-zn >
    layered_material_interface_depth)) {
379                 lambda2 = lambda;
380                 at_num = upper_material.atomic_number;
381                 at_wht = upper_material.atomic_weight;
382                 density = upper_material.density;
383                 SE_epsilon = upper_material.epsilon;
384                 SE_lambda = upper_material.lambda;
385                 get_constants(&sg_a,&al_a,&lam_a,at_num,at_wht,density,
    electron_beam.energy);
386                 lambda = compute_lambda(s_en,al_a,sg_a,lam_a);
387                 lambda1 = lambda;
388                 electron_step_length2 = sqrt(pow((
    layered_material_interface_depth-(-z))/cc,2));
389                 electron_step_length1 = sqrt(pow(((-zn)-
    layered_material_interface_depth)/cc,2));
390                 if (lower_material.atomic_number >= upper_material.atomic_number)
     {
391                     electron_step_length1 = electron_step_length1*(lambda2/
    lambda1);
392                 } else {
393                     electron_step_length1 = electron_step_length1*(lambda1/
    lambda2);
394                 }
395                 xn = x+(electron_step_length1+electron_step_length2)*ca;
396                 yn = y+(electron_step_length1+electron_step_length2)*cb;
397                 zn = z+(electron_step_length1+electron_step_length2)*cc;
398                 //printf("xn:%g,yn:%g,zn:%g,x:%g,y:%g,z:%g\n",xn,yn,zn,x,y,z);
399             } else if ((-z > layered_material_interface_depth) && (-zn <
    layered_material_interface_depth)) {
400                 lambda1 = lambda;
401                 at_num = lower_material.atomic_number;
402                 at_wht = lower_material.atomic_weight;
403                 density = lower_material.density;
404                 SE_epsilon = lower_material.epsilon;
405                 SE_lambda = lower_material.lambda;
406                 get_constants(&sg_a,&al_a,&lam_a,at_num,at_wht,density,
    electron_beam.energy);
407                 lambda = compute_lambda(s_en,al_a,sg_a,lam_a);
408                 lambda2 = lambda;
```

```
409                 electron_step_length1 = sqrt(pow((
    layered_material_interface_depth-(-z))/cc,2));
410                 electron_step_length2 = sqrt(pow(((-zn)-
    layered_material_interface_depth)/cc,2));
411                 if (upper_material.atomic_number >= lower_material.atomic_number)
     {
412                     electron_step_length2 = electron_step_length2*(lambda1/
    lambda2);
413                 } else {
414                     electron_step_length2 = electron_step_length2*(lambda2/
    lambda1);
415                 }
416                 xn = x+(electron_step_length1+electron_step_length2)*ca;
417                 yn = y+(electron_step_length1+electron_step_length2)*cb;
418                 zn = z+(electron_step_length1+electron_step_length2)*cc;
419                 //printf("xn:%g,yn:%g,zn:%g,x:%g,y:%g,z:%g\n",xn,yn,zn,x,y,z);
420             }
421 
422             // track max depth
423             if (maximum_electron_z_depth < zn) {
424                 maximum_electron_z_depth = zn;
425             }
426 
427             if (electron_trajectory_tracking == 1) {
428                 electron_tracking_x_position_array[electron_tracking_counter] =
    xn;
429                 electron_tracking_y_position_array[electron_tracking_counter] =
    yn;
430                 electron_tracking_z_position_array[electron_tracking_counter] =
    zn;
431                 electron_tracking_counter++;
432             }
433             //
434             reset_next_step(ca,cb,cc,xn,yn,zn,electron_step_length,&cx,&cy,&cz,&x
    ,&y,&z,&s_en,density,at_num,at_wht,SE_epsilon,SE_lambda,
    number_of_intersections,exit_x1,exit_y1,exit_x2,exit_y2,
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
    delta_r,number_of_surface_bins,intersection_index1);
435             //allow simulation to continue for FSE
436             if (number_of_intersections == 2) {
437                 number_of_intersections = 0;
438             }
439         }// while loop complete
440         //  scattering of electron in material complete
441         //   (electron has come to a rest or
442         //    electron has left the material)
443 
444         current_electron_number = current_electron_number + 1;
445         if (maximum_electron_z_depth >= 0.0) {
446             maximum_electron_z_depth += (
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array-1.0)
    /2.0;
447         } else {
```

```
448             maximum_electron_z_depth = (
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array-1.0)
    /2.0+maximum_electron_z_depth;
449         }
450         if (((int)round(maximum_electron_z_depth) < 0) || ((int)round(
    maximum_electron_z_depth) >
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array)) {
451             //logfile_printf("4 (int)round(maximum_electron_z_depth):%d\n",(int)
    round(maximum_electron_z_depth));
452             logfile_printf("---------------------------------\n");
453             logfile_printf("Warning (Code 4): An electron has travelled outside
    the defined depth array.  Increase the size of input parameter, '
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array', to
    prevent this warning.\n");
454             logfile_printf("---------------------------------\n");
455             #ifdef MPI_ON
456             MPI_Barrier(MPI_COMM_WORLD);
457             MPI_Finalize();
458             #endif
459             exit(0);
460         } else {
461             electron_maximum_z_depth_array[(int)round(maximum_electron_z_depth)]
    += 1.0;
462         }
463         maximum_electron_z_depth = -(
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array-1.0)
    /2.0;
464         if ((node_number == 0) && electron_trajectory_tracking == 1) {
465             FILE *fp;
466             fp=fopen("./outputs/output_track.txt", "a");
467             for (i=0;i<maximum_electron_track_length;i++) {
468                 fprintf(fp, "%g\t%g\t%g\n",
469                         electron_tracking_x_position_array[i],
470                         electron_tracking_y_position_array[i],
471                         electron_tracking_z_position_array[i]);
472                 if ((electron_tracking_x_position_array[i] == 0) && (
    electron_tracking_x_position_array[i+1] == 0)) {
473                     break;
474                 }
475             }
476             fclose(fp);
477         }
478     }
479     if (lost_electrons > 0 && node_number == 0) {//tell the user some electrons
    got out of the simulation area
480         logfile_printf("---------------------------------\n");
481         logfile_printf("Warning! %d electron(s) escaped the simulation area.
    Increase the size of input parameter, 'number_of_surface_bins', to prevent
    this warning.\n",lost_electrons);
482         logfile_printf("---------------------------------\n");
483     }
484 }
485
```

```
486
487
488 double mean_ionisation_potential(double at_num) { //keV
489     // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
490     //mean ionization potential represents the effective average energy loss per
        interaction between the incident electron and the solid
491     return((9.76*at_num+(58.5/pow(at_num,0.19)))*0.001);
492 }
493
494 void get_constants(double *sg_a,double *al_a,double *lam_a,double at_num,double
        at_wht,double density,double inc_energy) {
495     // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
496    //computes some constants needed by the program
497    double er;
498    *al_a = pow(at_num,0.67)*3.4E-3;
499    //relativistically correct the beam energy for use up to 500 keV
500    er = (inc_energy+511.0)/(inc_energy+1024.0);
501    er = er*er;
502    *lam_a = at_wht/(density*6.0221415E23); //lambda in cm
503    *lam_a = *lam_a*1.0E8; //put into angstroms
504    *sg_a = at_num*at_num*4*PI*5.21E-21*er;
505 }
506
507 void reset_coordinates(double *s_en,double *x,double *y,double *z,double *cx,
        double *cy,double *cz,double inc_energy,int surface_length) {
508     // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
509     int i;
510     //work out r position
511     double rand_num = rand();
512     for (i=0;i<surface_length;i++) {
513       if (electron_starting_location_probability_array[i] > (rand_num/((double)
        RAND_MAX+1))) {
514           break;
515       }
516    }
517     double r_pos = r_coordinate_array[i];
518     //work out angle
519     rand_num = rand();
520     double r_angle = (rand_num/((double)RAND_MAX+1))*2.0*PI;
521     //convert to cartesian
522     *x = r_pos*cos(r_angle);
523     *y = r_pos*sin(r_angle);
524     *z = -z_coordinate_array[i];
525     //printf("x:%g,y:%g,z:%g\n",*x,*y,*z);
526     //final starting bits
527     *cx = 0.0;
528    *cy = 0.0;
529    *cz = 1.0;
530    *s_en = inc_energy;
531 }
532
533 double compute_lambda(double energy,double al_a,double sg_a,double lam_a) {
534     // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
```

```
535     //computes elastic MFP for single scattering model
536     double al,ak,sg;
537     al = al_a/energy;
538     ak = al*(1.0+al);
539     //giving sg cross-section in cm2 as
540     sg = sg_a/(energy*energy*ak);
541     //and lambda in angstroms is
542     return(lam_a/sg);
543 }
544 
545 void s_scatter(double energy,double al_a,double *sp,double *ga,double *cp) {
546      // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
547     double rand_num = rand();
548     double al,Rl;
549     al = al_a/energy;
550     Rl = rand_num/((double)RAND_MAX+1);
551     *cp = 1.0-((2.0*al*Rl)/(1.0+al-Rl));
552     *sp = sqrt(1.0-(*cp)*(*cp));
553     //and get the azimuthal scattering angle
554     rand_num = rand();
555     *ga = 2.0*PI*(rand_num/((double)RAND_MAX+1));
556 }
557 
558 void new_coord(double step,double x,double y,double z,double cx,double cy,double
        cz,double sp,double ga,double cp,double *ca,double *cb,double *cc,double *xn,
        double *yn,double *zn) {
559      // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
560     //gets xn,yn,zn from x,y,z and scattering angles
561     //find the transformation angles
562     double an_m,an_n,v1,v2,v3,v4;
563     if (cz == 0.0) {
564         cz = 0.000001;
565     }
566     an_m = (-cx/cz);
567     an_n = 1.0/sqrt(1+(an_m*an_m));
568     //save computation time by getting all the transcendentals first
569     v1 = an_n*sp;
570     v2 = an_n*an_m*sp;
571     v3 = cos(ga);
572     v4 = sin(ga);
573     //find the new direction cosines
574     *ca = (cx*cp)+(v1*v3)+(cy*v2*v4);
575     *cb = (cy*cp)+(v4*(cz*v1-cx*v2));
576     *cc = (cz*cp)+(v2*v3)-(cy*v1*v4);
577     //and get the new coordinates
578     *xn = x+step*(*ca);
579     *yn = y+step*(*cb);
580     *zn = z+step*(*cc);
581      //printf("xn:%g,yn:%g,zn:%g,x:%g,y:%g,z:%g\n",*xn,*yn,*zn,x,y,z);
582 }
583 
584 void reset_next_step(double ca,
585                      double cb,
```

```
586                         double cc,
587                         double xn,
588                         double yn,
589                         double zn,
590                         double step,
591                         double *cx,
592                         double *cy,
593                         double *cz,
594                         double *x,
595                         double *y,
596                         double *z,
597                         double *s_en,
598                         double density,
599                         double at_num,
600                         double at_wht,
601                         double SE_epsilon,
602                         double SE_lambda,
603                         int number_of_intersections,
604                         double exit_x1,
605                         double exit_y1,
606                         double exit_x2,
607                         double exit_y2,
608                         double e_length_d,
609                         double delta_r,
610                         int number_of_surface_bins,
611                         int intersection_index1) {
612    // Refer to Chapter 3 "Monte carlo modelling for EM" by David Joy (1995)
613    //resets variables for next trajectory step
614   double del_E;
615   int i,dist_z_index,dist_zn_index,num_bins;
616   double step1,step2,r1,rn1;
617    double num_sec,dist_z,dist_zn,dist;
618    int z_idx, zn_idx;
619    //
620   r1 = sqrt(pow(*x,2)+pow(*y,2));
621   rn1 = sqrt(pow(xn,2)+pow(yn,2));
622   //find the energy lost on this step
623   del_E = step*stop_pwr(*s_en,at_num,at_wht)*density*1E-8;
624    //so the current energy is
625   *s_en = *s_en-del_E;
626   if (number_of_intersections == 2) {
627      step1 = sqrt(pow(exit_x1-r1,2)+pow(exit_y1-(*z),2));
628      step2 = sqrt(pow(rn1-exit_x2,2)+pow((-zn)-exit_y2,2));
629        z_idx = (int)round( (e_length_d-1.0)/2.0+(*z) );
630        zn_idx = (int)round( (e_length_d-1.0)/2.0+zn );
631        if ((z_idx >= 0) && (z_idx < e_length_d) && (zn_idx >= 0) && (zn_idx <
    e_length_d)) {
632            electron_energy_deposited_array[z_idx] += del_E*(step1/(step1+step2))
    ;
633            electron_energy_deposited_array[zn_idx] += del_E*(step2/(step1+step2)
    );
634        } else {
635            //logfile_printf("8 z_idx:%d,z:%g\n",z_idx,(*z));
```

```
636             //logfile_printf("8 zn_idx:%d,zn:%g\n",zn_idx,zn);
637             logfile_printf("---------------------------------\n");
638             logfile_printf("Warning (Code 8): An electron has travelled outside
     the defined depth array.  Increase the size of input parameter, '
     length_of_electron_energy_deposited_and_electron_maximum_z_depth_array', to
     prevent this warning.\n");
639             logfile_printf("---------------------------------\n");
640             #ifdef MPI_ON
641             MPI_Barrier(MPI_COMM_WORLD);
642             MPI_Finalize();
643             #endif
644             exit(0);
645         }
646     } else {
647         zn_idx = (int)round((e_length_d-1.0)/2.0+zn);
648         if ((zn_idx < 0) || (zn_idx > e_length_d)) {
649             //logfile_printf("7 zn_idx:%d,zn:%g\n",zn_idx,zn);
650             logfile_printf("---------------------------------\n");
651             logfile_printf("Warning (Code 7): An electron has travelled outside
     the defined depth array.  Increase the size of input parameter, '
     length_of_electron_energy_deposited_and_electron_maximum_z_depth_array', to
     prevent this warning.\n");
652             logfile_printf("---------------------------------\n");
653             #ifdef MPI_ON
654             MPI_Barrier(MPI_COMM_WORLD);
655             MPI_Finalize();
656             #endif
657             exit(0);
658             // FIXME: hack to prevent energy deposited outside array length
659             //        is there a better way to deal with this?
660             //zn = *z;
661         } else {
662             electron_energy_deposited_array[zn_idx] += del_E;
663         }
664     }
665     // electron leaving material
666     if (number_of_intersections == 1) {
667         step = sqrt(pow(exit_x1-r1,2)+pow(exit_y1-(-*z),2));
668         del_E = step*stop_pwr(*s_en,at_num,at_wht)*density*1E-8;
669         num_sec = del_E*(1.0/SE_epsilon);
670         r1 = sqrt(pow(*x,2)+pow(*y,2));
671         rn1 = exit_x1;
672         dist_z = dist_zn = 10000000.0;
673         dist_z_index = dist_zn_index = number_of_surface_bins - 1;
674 
675         // find dist_z where electron is leaving
676         for (i=0;i<number_of_surface_bins;i++) {
677             dist = sqrt(pow(r_coordinate_array[i]-r1,2)+pow(z_coordinate_array[i
     ]-(-(*z)),2));
678             if (dist < dist_z) {
679                 dist_z = dist;
680                 dist_z_index = i;
681             } else {
```

```
682                     break;
683                 }
684             }
685             // find dist_zn where electron is leaving
686             for (i=0;i<number_of_surface_bins;i++) {
687                 dist = sqrt(pow(r_coordinate_array[i]-rn1,2)+pow(z_coordinate_array[i
        ]-(-zn),2));
688                 if (dist < dist_zn) {
689                     dist_zn = dist;
690                     dist_zn_index = i;
691                 } else {
692                     break;
693                 }
694             }
695             // distribute the SE based on where they could escape from
696             num_bins = abs(dist_zn_index-dist_z_index);
697             if (num_bins == 0) {
698                 secondary_electron_location_array[dist_z_index] += 0.5*num_sec*exp(-
        dist_z/SE_lambda);
699             } else if (num_bins == 1) {
700                 secondary_electron_location_array[dist_z_index] += 0.5*num_sec/2.0*
        exp(-dist_z/SE_lambda);
701                 secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/2.0*
        exp(-dist_zn/SE_lambda);
702             } else {
703                 secondary_electron_location_array[dist_z_index] += 0.5*num_sec/(
        double)(num_bins+1)*exp(-dist_z/SE_lambda);
704                 secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/(
        double)(num_bins+1)*exp(-dist_zn/SE_lambda);
705                 if (dist_zn_index > dist_z_index) {
706                     for (i=1;i<num_bins;i++) {
707                         dist = sqrt(pow(r_coordinate_array[dist_z_index+i]-sqrt(pow(*
        x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
        double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index+i]-(-*z+(step/(double
        )(num_bins)*(double)i)*cc),2));
708                         secondary_electron_location_array[dist_z_index+i] += 0.5*
        num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
709                     }
710                 } else {
711                     for (i=1;i<num_bins;i++) {
712                         dist = sqrt(pow(r_coordinate_array[dist_z_index-i]-sqrt(pow(*
        x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
        double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index-i]-(-*z+(step/(double
        )(num_bins)*(double)i)*cc),2));
713                         secondary_electron_location_array[dist_z_index-i] += 0.5*
        num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
714                     }
715                 }
716             }
717         } else if (number_of_intersections == 2) {
718             //
719             step = sqrt(pow(exit_x1-r1,2)+pow(exit_y1-(-*z),2));
720             del_E = step*stop_pwr(*s_en,at_num,at_wht)*density*1E-8;
```

```
721         num_sec = del_E*(1.0/SE_epsilon);
722         r1 = sqrt(pow(*x,2)+pow(*y,2));
723         rn1 = exit_x1;
724         dist_z = dist_zn = 10000000.0;
725         dist_z_index = dist_zn_index = number_of_surface_bins - 1;
726 
727         for (i=0;i<number_of_surface_bins;i++) {
728             dist = sqrt(pow(r_coordinate_array[i]-r1,2)+pow(z_coordinate_array[i
    ]-(-(*z)),2));
729             if (dist < dist_z) {
730                 dist_z = dist;
731                 dist_z_index = i;
732             } else {
733                 break;
734             }
735         }
736         for (i=0;i<number_of_surface_bins;i++) {
737             dist = sqrt(pow(r_coordinate_array[i]-rn1,2)+pow(z_coordinate_array[i
    ]-(-zn),2));
738             if (dist < dist_zn) {
739                 dist_zn = dist;
740                 dist_zn_index = i;
741             } else {
742                 break;
743             }
744         }
745         // handle electron exit (re-entry is below)
746         num_bins = abs(dist_zn_index-dist_z_index);
747         if (num_bins == 0) {
748             secondary_electron_location_array[dist_z_index] += 0.5*num_sec*exp(-
    dist_z/SE_lambda);
749         } else if (num_bins == 1) {
750             secondary_electron_location_array[dist_z_index] += 0.5*num_sec/2.0*
    exp(-dist_z/SE_lambda);
751             secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/2.0*
    exp(-dist_zn/SE_lambda);
752         } else {
753             secondary_electron_location_array[dist_z_index] += 0.5*num_sec/(
    double)(num_bins+1)*exp(-dist_z/SE_lambda);
754             secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/(
    double)(num_bins+1)*exp(-dist_zn/SE_lambda);
755             if (dist_zn_index > dist_z_index) {
756                 for (i=1;i<num_bins;i++) {
757                     dist = sqrt(pow(r_coordinate_array[dist_z_index+i]-sqrt(pow(*
    x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
    double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index+i]-(-*z+(step/(double
    )(num_bins)*(double)i)*cc),2));
758                     secondary_electron_location_array[dist_z_index+i] += 0.5*
    num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
759                 }
760             } else {
761                 for (i=1;i<num_bins;i++) {
```

```
762                     dist = sqrt(pow(r_coordinate_array[dist_z_index-i]-sqrt(pow(*
    x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
    double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index-i]-(-*z+(step/(double
    )(num_bins)*(double)i)*cc),2));
763                     secondary_electron_location_array[dist_z_index-i] += 0.5*
    num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
764                 }
765             }
766         }
767         //
768         step = sqrt(pow(rn1-exit_x2,2)+pow((-zn)-exit_y2,2));
769         del_E = step*stop_pwr(*s_en,at_num,at_wht)*density*1E-8;
770         num_sec = del_E*(1.0/SE_epsilon);
771         r1 = exit_x2;
772         rn1 = sqrt(pow(xn,2)+pow(yn,2));
773         dist_z = dist_zn = 10000000.0;
774         dist_z_index = dist_zn_index = number_of_surface_bins - 1;
775 
776         for (i=0;i<number_of_surface_bins;i++) {
777             dist = sqrt(pow(r_coordinate_array[i]-r1,2)+pow(z_coordinate_array[i
    ]-(-(*z)),2));
778             if (dist < dist_z) {
779                 dist_z = dist;
780                 dist_z_index = i;
781             } else {
782                 break;
783             }
784         }
785         for (i=0;i<number_of_surface_bins;i++) {
786             dist = sqrt(pow(r_coordinate_array[i]-rn1,2)+pow(z_coordinate_array[i
    ]-(-zn),2));
787             if (dist < dist_zn) {
788                 dist_zn = dist;
789                 dist_zn_index = i;
790             } else {
791                 break;
792             }
793         }
794 
795         // handle re-entry
796         num_bins = abs(dist_zn_index-dist_z_index);
797         if (num_bins == 0) {
798             secondary_electron_location_array[dist_z_index] += 0.5*num_sec*exp(-
    dist_z/SE_lambda);
799         } else if (num_bins == 1) {
800             secondary_electron_location_array[dist_z_index] += 0.5*num_sec/2.0*
    exp(-dist_z/SE_lambda);
801             secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/2.0*
    exp(-dist_zn/SE_lambda);
802         } else {
803             secondary_electron_location_array[dist_z_index] += 0.5*num_sec/(
    double)(num_bins+1)*exp(-dist_z/SE_lambda);
```

```
804             secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/(
    double)(num_bins+1)*exp(-dist_zn/SE_lambda);
805             if (dist_zn_index > dist_z_index) {
806                 for (i=1;i<num_bins;i++) {
807                     dist = sqrt(pow(r_coordinate_array[dist_z_index+i]-sqrt(pow(*
    x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
    double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index+i]-(-*z+(step/(double
    )(num_bins)*(double)i)*cc),2));
808                     secondary_electron_location_array[dist_z_index+i] += 0.5*
    num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
809                 }
810             } else {
811                 for (i=1;i<num_bins;i++) {
812                     dist = sqrt(pow(r_coordinate_array[dist_z_index-i]-sqrt(pow(*
    x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
    double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index-i]-(-*z+(step/(double
    )(num_bins)*(double)i)*cc),2));
813                     secondary_electron_location_array[dist_z_index-i] += 0.5*
    num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
814                 }
815             }
816         }
817     } else {
818         //
819         num_sec = del_E*(1.0/SE_epsilon);
820         dist_z = dist_zn = 10000000.0;
821         dist_z_index = dist_zn_index = number_of_surface_bins - 1;
822
823         for (i=0;i<number_of_surface_bins;i++) {
824             dist = sqrt(pow(r_coordinate_array[i]-r1,2)+pow(z_coordinate_array[i
    ]-(-(*z)),2));
825             if (dist < dist_z) {
826                 dist_z = dist;
827                 dist_z_index = i;
828             } else {
829                 break;
830             }
831         }
832         for (i=0;i<number_of_surface_bins;i++) {
833             dist = sqrt(pow(r_coordinate_array[i]-rn1,2)+pow(z_coordinate_array[i
    ]-(-zn),2));
834             if (dist < dist_zn) {
835                 dist_zn = dist;
836                 dist_zn_index = i;
837             } else {
838                 break;
839             }
840         }
841         //
842         num_bins = abs(dist_zn_index-dist_z_index);
843         if (num_bins == 0) {
844             secondary_electron_location_array[dist_z_index] += 0.5*num_sec*exp(-
    dist_z/SE_lambda);
```

```
845         } else if (num_bins == 1) {
846             secondary_electron_location_array[dist_z_index] += 0.5*num_sec/2.0*
    exp(-dist_z/SE_lambda);
847             secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/2.0*
    exp(-dist_zn/SE_lambda);
848         } else {
849             secondary_electron_location_array[dist_z_index] += 0.5*num_sec/(
    double)(num_bins+1)*exp(-dist_z/SE_lambda);
850             secondary_electron_location_array[dist_zn_index] += 0.5*num_sec/(
    double)(num_bins+1)*exp(-dist_zn/SE_lambda);
851             if (dist_zn_index > dist_z_index) {
852                 for (i=1;i<num_bins;i++) {
853                     dist = sqrt(pow(r_coordinate_array[dist_z_index+i]-sqrt(pow(*
    x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
    double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index+i]-(-*z+(step/(double
    )(num_bins)*(double)i)*cc),2));
854                     secondary_electron_location_array[dist_z_index+i] += 0.5*
    num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
855                 }
856             } else {
857                 for (i=1;i<num_bins;i++) {
858                     dist = sqrt(pow(r_coordinate_array[dist_z_index-i]-sqrt(pow(*
    x+(step/(double)(num_bins)*(double)i)*ca,2)+pow(*y+(step/(double)(num_bins)*(
    double)i)*cb,2)),2)+pow(z_coordinate_array[dist_z_index-i]-(-*z+(step/(double
    )(num_bins)*(double)i)*cc),2));
859                     secondary_electron_location_array[dist_z_index-i] += 0.5*
    num_sec/(double)(num_bins+1)*exp(-dist/SE_lambda);
860                 }
861             }
862         }
863     }
864     //
865 
866    *cx = ca;
867    *cy = cb;
868    *cz = cc;
869    *x = xn;
870    *y = yn;
871    *z = zn;
872 }
873 
874 double stop_pwr(double energy,double at_num,double at_wht) {
875    //this computes the stopping power in keV/g/cm2 using the modified Bethe
    expression of Eg. (3.21)
876    double temp;
877    if (energy < 0.025) {
878       energy = 0.025;
879    }
880    temp = log(1.166*(energy+0.85*mean_ionisation_potential(at_num))/
    mean_ionisation_potential(at_num));
881    return(temp*78500*at_num/(at_wht*energy));
882 
883 }
```

B.10 Monte Carlo Surface Setup

```
//
//  monte_carlo_surface_setup.c
//
//  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
//
//  This is the module sets up the input surface needed by monte carlo module.
//
#ifdef XCODE
#include <stdlib.h>
#include "structures.h"
#include "prototypes.h"
#endif

void monte_carlo_surface_setup(int number_of_surface_bins,
                               double *z_depth_minimum,
                               double *z_depth_maximum) {
    int array_length,j,k;
    array_length = number_of_surface_bins*2;

    // find surface z min and max
    *z_depth_minimum = 100000.0;
    *z_depth_maximum = 0.0;
    for (k=0;k<number_of_surface_bins;k++) {
        if (z_coordinate_array[k] < *z_depth_minimum) {
            *z_depth_minimum = z_coordinate_array[k];
        }
        if (z_coordinate_array[k] > *z_depth_maximum) {
            *z_depth_maximum = z_coordinate_array[k];
        }
    }

    // create a doubled mirrored surface in r
    j = number_of_surface_bins-1;
    for (k=0;k<number_of_surface_bins;k++) {
        r_coordinate_both_directions[k] = -r_coordinate_array[j];
        z_coordinate_both_directions[k] = z_coordinate_array[j];
        r_coordinate_both_directions[k+number_of_surface_bins] =
    r_coordinate_array[k];
        z_coordinate_both_directions[k+number_of_surface_bins] =
    z_coordinate_array[k];
        j--;
    }

    // Note: the r,z values are the midpoints of the bins, to find which bin to
    assign an electron,
    //   we must calculate the half-bins and do a line-line test
    //
    // define bins shifted by +half and -half bin
    r_coordinate_secondary_array_one[0] = r_coordinate_both_directions[0];
    r_coordinate_secondary_array_two[0] = (r_coordinate_both_directions[1]+
    r_coordinate_both_directions[0])/2.0;
```

```
48      for (k=1;k<array_length-1;k++) {
49          r_coordinate_secondary_array_one[k] = (r_coordinate_both_directions[k]+
        r_coordinate_both_directions[k-1])/2.0;
50          r_coordinate_secondary_array_two[k] = (r_coordinate_both_directions[k]+
        r_coordinate_both_directions[k+1])/2.0;
51      }
52      r_coordinate_secondary_array_one[array_length-1] = (
        r_coordinate_both_directions[array_length-1]+r_coordinate_both_directions[
        array_length-2])/2.0;
53      r_coordinate_secondary_array_two[array_length-1] =
        r_coordinate_both_directions[array_length-1];
54
55      // compute surface line equations:  Ax + By = C
56      //      _array_one = A
57      //      _array_two = B
58      //      _array_three =
59      k = 0;
60      surface_line_equation_array_one[k] = ((z_coordinate_both_directions[k+1]+
        z_coordinate_both_directions[k])/2.0)-z_coordinate_both_directions[k];
61      surface_line_equation_array_two[k] = r_coordinate_secondary_array_one[k]-
        r_coordinate_secondary_array_two[k];
62      surface_line_equation_array_three[k] =
        temporary_surface_line_equation_array_one[k]*r_coordinate_secondary_array_one
        [k]
63                                                    +
        temporary_surface_line_equation_array_two[k]*z_coordinate_both_directions[k];
64      for (k=1;k<array_length-1;k++) {
65          surface_line_equation_array_one[k] = ((z_coordinate_both_directions[k]+
        z_coordinate_both_directions[k+1])/2.0)
66                                                        -((z_coordinate_both_directions[k
        ]+z_coordinate_both_directions[k-1])/2.0);
67          surface_line_equation_array_two[k] = r_coordinate_secondary_array_one[k]-
        r_coordinate_secondary_array_two[k];
68          surface_line_equation_array_three[k] =
        temporary_surface_line_equation_array_one[k]*r_coordinate_secondary_array_one
        [k]
69                                                        +
        temporary_surface_line_equation_array_two[k]*((z_coordinate_both_directions[k
        ]+z_coordinate_both_directions[k-1])/2.0);
70      }
71      k = array_length-1;
72      surface_line_equation_array_one[k] = z_coordinate_both_directions[k]-((
        z_coordinate_both_directions[k]+z_coordinate_both_directions[k-1])/2.0);
73      surface_line_equation_array_two[k] = r_coordinate_secondary_array_one[k]-
        r_coordinate_secondary_array_two[k];
74      surface_line_equation_array_three[k] =
        temporary_surface_line_equation_array_one[k]*r_coordinate_secondary_array_one
        [k]+temporary_surface_line_equation_array_two[k]*z_coordinate_both_directions
        [k];
75  }
```

B.11 Read Input Parameters

```
1  //
2  //  read_input_parameters.c
3  //
4  //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5  //
6  //  This is the module reads in the input parameter values from an input file,
       for the EBIED Simulator.
7  //
8  #ifdef XCODE
9  #include <stdio.h>
10 #include <string.h>
11 #include "structures.h"
12 #include "prototypes.h"
13 #endif
14 
15 void set_default_parameters(int *number_of_electron_trajectories,
16                             int *number_of_surface_bins,
17                             int *number_of_simulation_time_steps,
18                             int *run_MC_every_zero_point_X_percent,
19                             int *save_simulation_data_every_X_percent,
20                             int *
       length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
21                             int *maximum_electron_track_length,
22                             double *delta_r,
23                             double *delta_t,
24                             Electron_beam *electron_beam,
25                             double *layered_material_interface_depth,
26                             Material *upper_material,
27                             Material *lower_material,
28                             double *gas_temperature,
29                             Precursor *etch_precursor,
30                             Precursor *deposit_precursor,
31                             double *
       deposit_pinned_reaction_electron_cross_section,
32                             double *
       deposit_precursor_reaction_electron_cross_section,
33                             Toggle *toggle,
34                             int *number_of_points,
35                             int *order,
36                             int *no_etch_area,
37                             int *seed,
38                             int *wave_type,
39                             int *square_wave_period,
40                             int *square_wave_min_current,
41                             int *square_wave_max_current,
42                             double *triangle_wave_rate,
43                             int *triangle_wave_min_current,
44                             int *triangle_wave_max_current,
45                             int *pulsing_period,
46                             int *pulse_on_time,
47                             int *pulse_off_time,
```

```
48                                int *time_delay,
49                                double *precursor_diffusion_tolerance,
50                                double *substrate_temperature) {
51      //Default Input Parameter Values
52      *number_of_electron_trajectories = 1000;
53      *number_of_surface_bins = 1000;
54      *number_of_simulation_time_steps = 1000;
55      *run_MC_every_zero_point_X_percent = 1;
56      *save_simulation_data_every_X_percent = 10;
57      *length_of_electron_energy_deposited_and_electron_maximum_z_depth_array =
        100001;
58      *maximum_electron_track_length = 2000;
59      *delta_r = 1.0;
60      *delta_t = 1.0E-6;
61      (*electron_beam).cutoff_energy = 0.05;
62      (*electron_beam).top_hat_abruptness = 25.0;
63      (*electron_beam).diameter = 40.0;
64      (*electron_beam).energy = 1.0;
65      (*electron_beam).current = 6.241509e9;
66      (*electron_beam).tilt = 0.0;
67      *layered_material_interface_depth = 0.0;
68      (*upper_material).atomic_number = 47.0; //upper material atomic number.
69      (*upper_material).atomic_weight = 107.8682; //upper material atomic weight.
70      (*upper_material).density = 10.49; //upper material density, in g/cm3.
71      (*upper_material).epsilon = 0.05; //effective energy required to produce an
        SE, in keV.
72      (*upper_material).lambda = 10.0; //effective SE escape depth, in A.
73      (*lower_material).atomic_number = 47.0; //lower material atomic number.
74      (*lower_material).atomic_weight = 107.8682; //lower material atomic weight.
75      (*lower_material).density = 10.49; //lower material density, in g/cm3.
76      (*lower_material).epsilon = 0.05; //effective energy required to produce an
        SE, in keV.
77      (*lower_material).lambda = 10.0; //effective SE escape depth, in A.
78      *gas_temperature = 290.0;
79      (*etch_precursor).gas_partial_pressure = 1.0;
80      (*etch_precursor).reactive_product_molecular_mass = 2.12981E-25;
81      (*etch_precursor).surface_area = 68.0;
82      (*etch_precursor).desorption_energy = 1.0;
83      (*etch_precursor).desorption_attempt_frequency = 1.0;
84      (*etch_precursor).diffusion_energy = 1.0;
85      (*etch_precursor).diffusion_attempt_frequency = 1.0E8;
86      (*etch_precursor).sticking_coefficient = 1.0;
87      (*etch_precursor).PE_electron_cross_section = 1.0;
88      (*etch_precursor).BSE_electron_cross_section = 1.0;
89      (*etch_precursor).SE_electron_cross_section = 1.0;
90      (*deposit_precursor).gas_partial_pressure = 1.0;
91      (*deposit_precursor).reactive_product_molecular_mass = 2.12981E-25;
92      (*deposit_precursor).surface_area = 68.0;
93      (*deposit_precursor).desorption_energy = 1.0;
94      (*deposit_precursor).desorption_attempt_frequency = 1.0;
95      (*deposit_precursor).diffusion_energy = 1.0;
96      (*deposit_precursor).diffusion_attempt_frequency = 1.0E8;
97      (*deposit_precursor).sticking_coefficient = 1.0;
```

```
98      (*deposit_precursor).PE_electron_cross_section = 1.0;
99      (*deposit_precursor).BSE_electron_cross_section = 1.0;
100     (*deposit_precursor).SE_electron_cross_section = 1.0;
101     *deposit_pinned_reaction_electron_cross_section = 1.0;
102     *deposit_precursor_reaction_electron_cross_section = 1.0;
103     (*toggle).electron_trajectory_simulator = 1;
104     (*toggle).electron_trajectory_tracking = 0;
105     (*toggle).electron_beam_projection = 1;
106     (*toggle).electron_beam_shape = 0;
107     (*toggle).surface_evolution = 1;
108     (*toggle).previous_simulation = 0;
109     (*toggle).coverage = 1;
110     wave_type = 0;
111     *square_wave_period = 100;
112     *square_wave_min_current = 100;
113     *square_wave_max_current = 1000;
114     *triangle_wave_rate = 1;
115     *triangle_wave_min_current = 100;
116     *triangle_wave_max_current = 1000;
117     *number_of_points = 21;
118     *order = 1;
119     *no_etch_area = 0;
120     *seed = -1;
121     *time_delay = 0;
122     *precursor_diffusion_tolerance = 0.1;
123     *substrate_temperature = 290.0;
124 }
125
126 void read_input_parameters(int *number_of_electron_trajectories,
127                            int *number_of_surface_bins,
128                            int *number_of_simulation_time_steps,
129                            int *run_MC_every_zero_point_X_percent,
130                            int *save_simulation_data_every_X_percent,
131                            int *
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,
132                            int *maximum_electron_track_length,
133                            double *delta_r,
134                            double *delta_t,
135                            Electron_beam *electron_beam,
136                            double *layered_material_interface_depth,
137                            Material *upper_material,
138                            Material *lower_material,
139                            double *gas_temperature,
140                            Precursor *etch_precursor,
141                            Precursor *deposit_precursor,
142                            double *deposit_pinned_reaction_electron_cross_section
        ,
143                            double *
        deposit_precursor_reaction_electron_cross_section,
144                            Toggle *toggle,
145                            int *number_of_points,
146                            int *order,
147                            int *no_etch_area,
```

```
148                             int *seed,
149                             int *wave_type,
150                             int *square_wave_period,
151                             int *square_wave_min_current,
152                             int *square_wave_max_current,
153                             double *triangle_wave_rate,
154                             int *triangle_wave_min_current,
155                             int *triangle_wave_max_current,
156                             int *pulsing_period,
157                             int *pulse_on_time,
158                             int *pulse_off_time,
159                             int *time_delay,
160                             double *precursor_diffusion_tolerance,
161                             double *substrate_temperature) {
162     //Start, Read In Input Parameters
163     char variable_name[100];
164     double variable_data_double;
165     int variable_data_int;
166     int variable_data_long_int;
167     FILE *fp;
168 
169     //Read In Input Paremeters, Double Data Type
170     fp = fopen("./inputs/input_parameters.txt","r");
171     while (!feof(fp)) {
172         fscanf(fp,"%s = %lf\n",variable_name,&variable_data_double);
173         if (strcmp("delta_r",variable_name) == 0) {
174             *delta_r = variable_data_double;
175             continue;
176         }
177         if (strcmp("delta_t",variable_name) == 0) {
178             *delta_t = variable_data_double;
179             continue;
180         }
181         if (strcmp("electron_beam.cutoff_energy",variable_name) == 0) {
182             (*electron_beam).cutoff_energy = variable_data_double;
183             continue;
184         }
185         if (strcmp("electron_beam.top_hat_abruptness",variable_name) == 0) {
186             (*electron_beam).top_hat_abruptness = variable_data_double;
187             continue;
188         }
189         if (strcmp("electron_beam.diameter",variable_name) == 0) {
190             (*electron_beam).diameter = variable_data_double;
191             continue;
192         }
193         if (strcmp("electron_beam.energy",variable_name) == 0) {
194             (*electron_beam).energy = variable_data_double;
195             continue;
196         }
197         if (strcmp("electron_beam.current",variable_name) == 0) {
198             (*electron_beam).current = variable_data_double;
199             continue;
200         }
```

```
201         if (strcmp("electron_beam.tilt",variable_name) == 0) {
202             (*electron_beam).tilt = variable_data_double;
203             continue;
204         }
205         if (strcmp("layered_material_interface_depth",variable_name) == 0) {
206             *layered_material_interface_depth = variable_data_double;
207             continue;
208         }
209         if (strcmp("upper_material.atomic_number",variable_name) == 0) {
210             (*upper_material).atomic_number = variable_data_double;
211             continue;
212         }
213         if (strcmp("upper_material.atomic_weight",variable_name) == 0) {
214             (*upper_material).atomic_weight = variable_data_double;
215             continue;
216         }
217         if (strcmp("upper_material.density",variable_name) == 0) {
218             (*upper_material).density = variable_data_double;
219             continue;
220         }
221         if (strcmp("upper_material.epsilon",variable_name) == 0) {
222             (*upper_material).epsilon = variable_data_double;
223             continue;
224         }
225         if (strcmp("upper_material.lambda",variable_name) == 0) {
226             (*upper_material).lambda = variable_data_double;
227             continue;
228         }
229         if (strcmp("lower_material.atomic_number",variable_name) == 0) {
230             (*lower_material).atomic_number = variable_data_double;
231             continue;
232         }
233         if (strcmp("lower_material.atomic_weight",variable_name) == 0) {
234             (*lower_material).atomic_weight = variable_data_double;
235             continue;
236         }
237         if (strcmp("lower_material.density",variable_name) == 0) {
238             (*lower_material).density = variable_data_double;
239             continue;
240         }
241         if (strcmp("lower_material.epsilon",variable_name) == 0) {
242             (*lower_material).epsilon = variable_data_double;
243             continue;
244         }
245         if (strcmp("lower_material.lambda",variable_name) == 0) {
246             (*lower_material).lambda = variable_data_double;
247             continue;
248         }
249         if (strcmp("gas_temperature",variable_name) == 0) {
250             *gas_temperature = variable_data_double;
251             continue;
252         }
253         if (strcmp("substrate_temperature",variable_name) == 0) {
```

```
254             *substrate_temperature = variable_data_double;
255             continue;
256         }
257         if (strcmp("etch_precursor.gas_partial_pressure",variable_name) == 0) {
258             (*etch_precursor).gas_partial_pressure = variable_data_double;
259             continue;
260         }
261         if (strcmp("etch_precursor.reactive_product_molecular_mass",variable_name
    ) == 0) {
262             (*etch_precursor).reactive_product_molecular_mass =
    variable_data_double;
263             continue;
264         }
265         if (strcmp("etch_precursor.surface_area",variable_name) == 0) {
266             (*etch_precursor).surface_area = variable_data_double;
267             continue;
268         }
269         if (strcmp("etch_precursor.desorption_energy",variable_name) == 0) {
270             (*etch_precursor).desorption_energy = variable_data_double;
271             continue;
272         }
273         if (strcmp("etch_precursor.desorption_attempt_frequency",variable_name)
    == 0) {
274             (*etch_precursor).desorption_attempt_frequency = variable_data_double
    ;
275             continue;
276         }
277         if (strcmp("etch_precursor.diffusion_energy",variable_name) == 0) {
278             (*etch_precursor).diffusion_energy = variable_data_double;
279             continue;
280         }
281         if (strcmp("etch_precursor.diffusion_attempt_frequency",variable_name) ==
     0) {
282             (*etch_precursor).diffusion_attempt_frequency = variable_data_double;
283             continue;
284         }
285         if (strcmp("etch_precursor.sticking_coefficient",variable_name) == 0) {
286             (*etch_precursor).sticking_coefficient = variable_data_double;
287             continue;
288         }
289         if (strcmp("etch_precursor.PE_electron_cross_section",variable_name) ==
    0) {
290             (*etch_precursor).PE_electron_cross_section = variable_data_double;
291             continue;
292         }
293         if (strcmp("etch_precursor.BSE_electron_cross_section",variable_name) ==
    0) {
294             (*etch_precursor).BSE_electron_cross_section = variable_data_double;
295             continue;
296         }
297         if (strcmp("etch_precursor.SE_electron_cross_section",variable_name) ==
    0) {
298             (*etch_precursor).SE_electron_cross_section = variable_data_double;
```

```
299             continue;
300         }
301         if (strcmp("deposit_precursor.gas_partial_pressure",variable_name) == 0)
    {
302             (*deposit_precursor).gas_partial_pressure = variable_data_double;
303             continue;
304         }
305         if (strcmp("deposit_precursor.reactive_product_molecular_mass",
    variable_name) == 0) {
306             (*deposit_precursor).reactive_product_molecular_mass =
    variable_data_double;
307             continue;
308         }
309         if (strcmp("deposit_precursor.surface_area",variable_name) == 0) {
310             (*deposit_precursor).surface_area = variable_data_double;
311             continue;
312         }
313         if (strcmp("deposit_precursor.desorption_energy",variable_name) == 0) {
314             (*deposit_precursor).desorption_energy = variable_data_double;
315             continue;
316         }
317         if (strcmp("deposit_precursor.desorption_attempt_frequency",variable_name
    ) == 0) {
318             (*deposit_precursor).desorption_attempt_frequency =
    variable_data_double;
319             continue;
320         }
321         if (strcmp("deposit_precursor.diffusion_energy",variable_name) == 0) {
322             (*deposit_precursor).diffusion_energy = variable_data_double;
323             continue;
324         }
325         if (strcmp("deposit_precursor.diffusion_attempt_frequency",variable_name)
     == 0) {
326             (*deposit_precursor).diffusion_attempt_frequency =
    variable_data_double;
327             continue;
328         }
329         if (strcmp("deposit_precursor.sticking_coefficient",variable_name) == 0)
    {
330             (*deposit_precursor).sticking_coefficient = variable_data_double;
331             continue;
332         }
333         if (strcmp("deposit_precursor.PE_electron_cross_section",variable_name)
    == 0) {
334             (*deposit_precursor).PE_electron_cross_section = variable_data_double
    ;
335             continue;
336         }
337         if (strcmp("deposit_precursor.BSE_electron_cross_section",variable_name)
    == 0) {
338             (*deposit_precursor).BSE_electron_cross_section =
    variable_data_double;
339             continue;
```

```
340         }
341         if (strcmp("deposit_precursor.SE_electron_cross_section",variable_name)
    == 0) {
342             (*deposit_precursor).SE_electron_cross_section = variable_data_double
    ;
343             continue;
344         }
345         if (strcmp("deposit_pinned_reaction_electron_cross_section",variable_name
    ) == 0) {
346             *deposit_pinned_reaction_electron_cross_section =
    variable_data_double;
347             continue;
348         }
349         if (strcmp("deposit_precursor_reaction_electron_cross_section",
    variable_name) == 0) {
350             *deposit_precursor_reaction_electron_cross_section =
    variable_data_double;
351             continue;
352         }
353         if (strcmp("triangle_wave_rate",variable_name) == 0) {
354             *triangle_wave_rate = variable_data_double;
355             continue;
356         }
357         if (strcmp("precursor_diffusion_tolerance",variable_name) == 0) {
358             *precursor_diffusion_tolerance = variable_data_double;
359             continue;
360         }
361         if (strcmp("substrate_temperature",variable_name) == 0) {
362             *substrate_temperature = variable_data_double;
363             continue;
364         }
365     }
366     fclose(fp);
367
368     //Read In Input Paremeters, Int Data Type
369     fp = fopen("./inputs/input_parameters.txt","r");
370     while (!feof(fp)) {
371         fscanf(fp,"%s = %d\n",variable_name,&variable_data_long_int);
372         if (strcmp("number_of_electron_trajectories",variable_name) == 0) {
373             *number_of_electron_trajectories = variable_data_long_int;
374             continue;
375         }
376         if (strcmp("number_of_surface_bins",variable_name) == 0) {
377             *number_of_surface_bins = variable_data_long_int;
378             continue;
379         }
380         if (strcmp("number_of_simulation_time_steps",variable_name) == 0) {
381             *number_of_simulation_time_steps = variable_data_long_int;
382             continue;
383         }
384         if (strcmp("run_MC_every_zero_point_X_percent",variable_name) == 0) {
385             *run_MC_every_zero_point_X_percent = variable_data_long_int;
386             continue;
```

```
387         }
388         if (strcmp("save_simulation_data_every_X_percent",variable_name) == 0) {
389             *save_simulation_data_every_X_percent = variable_data_long_int;
390             continue;
391         }
392         if (strcmp("
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array",
        variable_name) == 0) {
393             *
        length_of_electron_energy_deposited_and_electron_maximum_z_depth_array =
        variable_data_long_int;
394             continue;
395         }
396         if (strcmp("maximum_electron_track_length",variable_name) == 0) {
397             *maximum_electron_track_length = variable_data_long_int;
398             continue;
399         }
400     }
401     fclose(fp);
402 
403     //Read In Input Paremeters, Int Data Type
404     fp = fopen("./inputs/input_parameters.txt","r");
405     while (!feof(fp)) {
406         fscanf(fp,"%s = %d\n",variable_name,&variable_data_int);
407         if (strcmp("toggle.electron_trajectory_simulator",variable_name) == 0) {
408             (*toggle).electron_trajectory_simulator = variable_data_int;
409             continue;
410         }
411         if (strcmp("toggle.electron_trajectory_tracking",variable_name) == 0) {
412             (*toggle).electron_trajectory_tracking = variable_data_int;
413             continue;
414         }
415         if (strcmp("toggle.electron_beam_projection",variable_name) == 0) {
416             (*toggle).electron_beam_projection = variable_data_int;
417             continue;
418         }
419         if (strcmp("toggle.electron_beam_shape",variable_name) == 0) {
420             (*toggle).electron_beam_shape = variable_data_int;
421             continue;
422         }
423         if (strcmp("toggle.surface_evolution",variable_name) == 0) {
424             (*toggle).surface_evolution = variable_data_int;
425             continue;
426         }
427         if (strcmp("toggle.previous_simulation",variable_name) == 0) {
428             (*toggle).previous_simulation = variable_data_int;
429             continue;
430         }
431         if (strcmp("toggle.coverage",variable_name) == 0) {
432             (*toggle).coverage = variable_data_int;
433             continue;
434         }
435         if (strcmp("wave_type",variable_name) == 0) {
```

```
436             *wave_type = variable_data_int;
437             continue;
438         }
439         if (strcmp("square_wave_period",variable_name) == 0) {
440             *square_wave_period = variable_data_int;
441             continue;
442         }
443         if (strcmp("square_wave_min_current",variable_name) == 0) {
444             *square_wave_min_current = variable_data_int;
445             continue;
446         }
447         if (strcmp("square_wave_max_current",variable_name) == 0) {
448             *square_wave_max_current = variable_data_int;
449             continue;
450         }
451         if (strcmp("triangle_wave_min_current",variable_name) == 0) {
452             *triangle_wave_min_current = variable_data_int;
453             continue;
454         }
455         if (strcmp("triangle_wave_max_current",variable_name) == 0) {
456             *triangle_wave_max_current = variable_data_int;
457             continue;
458         }
459         if (strcmp("pulsing_period",variable_name) == 0) {
460             *pulsing_period = variable_data_int;
461             continue;
462         }
463         if (strcmp("pulse_on_time",variable_name) == 0) {
464             *pulse_on_time = variable_data_int;
465             continue;
466         }
467         if (strcmp("pulse_off_time",variable_name) == 0) {
468             *pulse_off_time = variable_data_int;
469             continue;
470         }
471         if (strcmp("number_of_points",variable_name) == 0) {
472             *number_of_points = variable_data_int;
473             continue;
474         }
475         if (strcmp("order",variable_name) == 0) {
476             *order = variable_data_int;
477             continue;
478         }
479         if (strcmp("no_etch_area",variable_name) == 0) {
480             *no_etch_area = variable_data_int;
481             continue;
482         }
483         if (strcmp("seed",variable_name) == 0) {
484             *seed = variable_data_int;
485             continue;
486         }
487         if (strcmp("time_delay",variable_name) == 0) {
488             *time_delay = variable_data_int;
```

```
489             continue;
490         }
491     }
492     fclose(fp);
493
494     //End, Read In Input Parameters
495 }
```

B.12 Read Previous Simulation Data

```
1  //
2  //  read_input_previous_simulation.c
3  //
4  //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5  //
6  //  This is the module reads in a previous simulation outputs from an input file,
    for the EBIED Simulator.
7  //
8  #ifdef XCODE
9  #include <stdio.h>
10 #include "structures.h"
11 #include "prototypes.h"
12 #endif
13
14 void read_input_previous_simulation(int number_of_surface_bins) {
15     FILE *fp;
16     int index,i;
17     double input_r,input_z,input_Nd,input_Ne,input_ND;
18     fp = fopen("./inputs/input_previous_simulation.txt","r");
19     while (!feof(fp)) {
20         fscanf(fp,"%d\t%lf\t%lf\t%lf\t%lf\t%lf\n",&index,&input_r,&input_z,&
    input_Nd,&input_Ne,&input_ND);
21         previous_r_coordinate_array[index-1] = input_r;
22         previous_z_coordinate_array[index-1] = input_z;
23         previous_deposit_precursor_gas_concentration_previous_time_step_array[
    index-1] = input_Nd;
24         previous_etch_precursor_gas_concentration_previous_time_step_array[index
    -1] = input_Ne;
25         previous_reactive_product_concentration_previous_time_step_array[index-1]
     = input_ND;
26     }
27     fclose(fp);
28     if (index<number_of_surface_bins) {
29         for (i=index;i<number_of_surface_bins;i++) {
30             previous_r_coordinate_array[i] = previous_r_coordinate_array[i-1]+(
    previous_r_coordinate_array[i-1]-previous_r_coordinate_array[i-2]);
31             previous_z_coordinate_array[i] = input_z;
32             previous_deposit_precursor_gas_concentration_previous_time_step_array
    [i] = input_Nd;
```

```
33              previous_etch_precursor_gas_concentration_previous_time_step_array[i]
     = input_Ne;
34              previous_reactive_product_concentration_previous_time_step_array[i] =
     input_ND;
35          }
36      }
37  }
```

B.13 Output Current Simulation Data

```
1   //
2   //  save_current_simulation_data.c
3   //
4   //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5   //
6   //  This function save the current simulation data for the EBIED Simulator.
7   //
8   #ifdef XCODE
9   #include <stdio.h>
10  #include "structures.h"
11  #include "prototypes.h"
12  #endif
13
14  void save_current_simulation_data(int
    length_of_electron_energy_deposited_and_electron_maximum_z_depth_array,double
     backscattered_electron_coefficient,double secondary_electron_coefficient,int
     number_of_surface_bins,int number_of_electron_trajectories,int
    current_time_step,int val1,int val2,int number_of_simulation_time_steps,int
    save_simulation_data_every_X_percent) {
15     int j,k;
16      FILE *fp;
17      //
18      fp=fopen("./outputs/output_electron_flux_profile.txt", "w");
19      for (k=0;k<number_of_surface_bins;k++) {
20          print_electron_flux_profile_array[val1][k] = r_coordinate_array[k];
21          print_electron_flux_profile_array[val2][k] = electron_flux_profile_array[
    k];
22      }
23      fprintf(fp,"Electron Flux Profile\n");
24      for (j=0;j<=val1/2;j++) {
25          fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
    number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
    ]);
26      }
27      fprintf(fp,"\n");
28      for (j=0;j<=val2/2;j++) {
29          fprintf(fp,"r(A)\teFlux(electrons/s)\t");
30      }
31      fprintf(fp,"\n");
32      for (k=0;k<number_of_surface_bins;k++) {
```

```
33          for (j=0;j<=val2;j++) {
34              fprintf(fp,"%g\t",print_electron_flux_profile_array[j][k]);
35          }
36          fprintf(fp,"\n");
37      }
38      fclose(fp);
39      //
40      fp=fopen("./outputs/output_N_little_e.txt", "w");
41      for (k=0;k<number_of_surface_bins;k++) {
42          print_N_little_e[val1][k] = r_coordinate_array[k];
43          print_N_little_e[val2][k] =
        etch_precursor_gas_concentration_previous_time_step_array[k];
44      }
45      fprintf(fp,"Etch Precursor Concentration\n");
46      for (j=0;j<=val1/2;j++) {
47          fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
        number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
        ]);
48      }
49      fprintf(fp,"\n");
50      for (j=0;j<=val2/2;j++) {
51          fprintf(fp,"r(A)\tConcentration(molecules/A2)\t");
52      }
53      fprintf(fp,"\n");
54      for (k=0;k<number_of_surface_bins;k++) {
55          for (j=0;j<=val2;j++) {
56              fprintf(fp,"%g\t",print_N_little_e[j][k]);
57          }
58          fprintf(fp,"\n");
59      }
60      fclose(fp);
61      //
62      fp=fopen("./outputs/output_N_little_d.txt", "w");
63      for (k=0;k<number_of_surface_bins;k++) {
64          print_N_little_d[val1][k] = r_coordinate_array[k];
65          print_N_little_d[val2][k] =
        deposit_precursor_gas_concentration_previous_time_step_array[k];
66      }
67      fprintf(fp,"Deposit Precursor Concentration\n");
68      for (j=0;j<=val1/2;j++) {
69          fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
        number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
        ]);
70      }
71      fprintf(fp,"\n");
72      for (j=0;j<=val2/2;j++) {
73          fprintf(fp,"r(A)\tConcentration(molecules/A2)\t");
74      }
75      fprintf(fp,"\n");
76      for (k=0;k<number_of_surface_bins;k++) {
77          for (j=0;j<=val2;j++) {
78              fprintf(fp,"%g\t",print_N_little_d[j][k]);
79          }
```

```
80          fprintf(fp,"\n");
81      }
82      fclose(fp);
83      //
84      fp=fopen("./outputs/output_N_big_d.txt", "w");
85      for (k=0;k<number_of_surface_bins;k++) {
86          print_N_big_d[val1][k] = r_coordinate_array[k];
87          print_N_big_d[val2][k] =
        reactive_product_concentration_previous_time_step_array[k];
88      }
89      fprintf(fp,"Deposited Molecule Concentration\n");
90      for (j=0;j<=val1/2;j++) {
91          fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
        number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
        ]);
92      }
93      fprintf(fp,"\n");
94      for (j=0;j<=val2/2;j++) {
95          fprintf(fp,"r(A)\tConcentration(molecules/A2)\t");
96      }
97      fprintf(fp,"\n");
98      for (k=0;k<number_of_surface_bins;k++) {
99          for (j=0;j<=val2;j++) {
100             fprintf(fp,"%g\t",print_N_big_d[j][k]);
101         }
102         fprintf(fp,"\n");
103     }
104     fclose(fp);
105     //
106     /*fp=fopen("./outputs/output_growth.txt", "w");
107     for (k=0;k<number_of_surface_bins;k++) {
108         print_growth_rate[val1][k] = r_coordinate_array[k];
109         print_growth_rate[val2][k] = growth_or_etch_rate_array[k];
110     }
111     fprintf(fp,"Growth/Etch Rate\n");
112     for (j=0;j<=val1/2;j++) {
113         fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
        number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
        ]);
114     }
115     fprintf(fp,"\n");
116     for (j=0;j<=val2/2;j++) {
117         fprintf(fp,"r(A)\tRate(molecules/s)\t");
118     }
119     fprintf(fp,"\n");
120     for (k=0;k<number_of_surface_bins;k++) {
121         for (j=0;j<=val2;j++) {
122             fprintf(fp,"%g\t",print_growth_rate[j][k]);
123         }
124         fprintf(fp,"\n");
125     }
126     fclose(fp);*/
127     //
```

```
128     fp=fopen("./outputs/output_surface.txt", "w");
129     for (k=0;k<number_of_surface_bins;k++) {
130         print_surface[val1][k] = r_coordinate_array[k];
131         print_surface[val2][k] = z_coordinate_array[k];
132     }
133     fprintf(fp,"Surface Height\n");
134     for (j=0;j<=val1/2;j++) {
135         fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
        number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
        ]);
136     }
137     fprintf(fp,"\n");
138     for (j=0;j<=val2/2;j++) {
139         fprintf(fp,"r(A)\tz(A)\t");
140     }
141     fprintf(fp,"\n");
142     for (k=0;k<number_of_surface_bins;k++) {
143         for (j=0;j<=val2;j++) {
144             fprintf(fp,"%g\t",print_surface[j][k]);
145         }
146         fprintf(fp,"\n");
147     }
148     fclose(fp);
149     //
150     fp=fopen("./outputs/output_e_SE.txt", "w");
151     for (k=0;k<number_of_surface_bins;k++) {
152         print_secondary_electrons[val1][k] = r_coordinate_array[k];
153         print_secondary_electrons[val2][k] = secondary_electron_location_array[k
        ];
154     }
155     fprintf(fp,"Secondary Electron Counts\n");
156     for (j=0;j<=val1/2;j++) {
157         fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
        number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
        ]);
158     }
159     fprintf(fp,"\n");
160     for (j=0;j<=val2/2;j++) {
161         fprintf(fp,"r(A)\tSE(counts)\t");
162     }
163     fprintf(fp,"\n");
164     for (k=0;k<number_of_surface_bins;k++) {
165         for (j=0;j<=val2;j++) {
166             fprintf(fp,"%g\t",print_secondary_electrons[j][k]);
167         }
168         fprintf(fp,"\n");
169     }
170     fclose(fp);
171     //
172     fp=fopen("./outputs/output_e_BSE.txt", "w");
173     for (k=0;k<number_of_surface_bins;k++) {
174         print_backscattered_electrons[val1][k] = r_coordinate_array[k];
```

```
175         print_backscattered_electrons[val2][k] =
    backscattered_electron_location_array[k];
176     }
177     fprintf(fp,"Backscattered Electron Counts\n");
178     for (j=0;j<=val1/2;j++) {
179         fprintf(fp,"Time(sec) %g\t\t",time_per_time_step_array[
    number_of_simulation_time_steps/(100/save_simulation_data_every_X_percent)*j
    ]);
180     }
181     fprintf(fp,"\n");
182     for (j=0;j<=val2/2;j++) {
183         fprintf(fp,"r(A)\tBSE(counts)\t");
184     }
185     fprintf(fp,"\n");
186     for (k=0;k<number_of_surface_bins;k++) {
187         for (j=0;j<=val2;j++) {
188             fprintf(fp,"%g\t",print_backscattered_electrons[j][k]);
189         }
190         fprintf(fp,"\n");
191     }
192     fclose(fp);
193     //
194     fp=fopen("./outputs/output_bksct_se.txt", "a");
195     fprintf(fp,"%g\t%g\n",backscattered_electron_coefficient,
    secondary_electron_coefficient);
196     fclose(fp);
197     //
198     //Save data for next simulation
199     fp=fopen("./inputs/input_previous_simulation.txt", "w");
200     for (k=0;k<number_of_surface_bins;k++) {
201         fprintf(fp,"%d\t%g\t%g\t%g\t%g\t%g\n",k+1,r_coordinate_array[k],
    z_coordinate_array[k],
    deposit_precursor_gas_concentration_current_time_step_array[k],
    etch_precursor_gas_concentration_current_time_step_array[k],
    reactive_product_concentration_current_time_step_array[k]);
202     }
203     fclose(fp);
204 }
```

B.14 Create Output Files

```
1 //
2 //  setup_save_files.c
3 //
4 //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5 //
6 //  This creates the save files for the EBIED Simulator.
7 //
8 #ifdef XCODE
9 #include <stdio.h>
```

```
#include <sys/stat.h>
#include "structures.h"
#include "prototypes.h"
#endif

void setup_save_files() {
    mkdir("./outputs",S_IRWXU|S_IRWXG);
   FILE *fp;
    fp=fopen("./outputs/output_electron_flux_profile.txt", "w");
    fprintf(fp,"EFlux\n");
    fclose(fp);
    fp=fopen("./outputs/output_e_SE.txt", "w");
    fprintf(fp,"e_SE\n");
    fclose(fp);
    fp=fopen("./outputs/output_N_little_e.txt", "w");
    fprintf(fp,"N_e\n");
    fclose(fp);
    fp=fopen("./outputs/output_N_little_d.txt", "w");
    fprintf(fp,"N_d\n");
    fclose(fp);
    fp=fopen("./outputs/output_N_big_D.txt", "w");
    fprintf(fp,"N_D\n");
    fclose(fp);
    /*fp=fopen("./outputs/output_growth.txt", "w");
    fprintf(fp,"Growth Rate\n");
    fclose(fp);*/
    fp=fopen("./outputs/output_surface.txt", "w");
    fprintf(fp,"Surface Height\n");
    fclose(fp);
    fp=fopen("./outputs/output_e_BSE.txt", "w");
    fprintf(fp,"e_BSE\n");
    fclose(fp);
    fp=fopen("./outputs/output_bksct_se.txt", "w");
    fprintf(fp,"BSE Coefficient\tSE Yield\n");
    fclose(fp);
    fp=fopen("./outputs/logfile.txt", "w");
    fprintf(fp,"--------Simulation Logfile-------\n");
    fclose(fp);
}
```

B.15 Crank-Nicholson Solver

```
//
//  solve_matrix.c
//
//  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
//
//  This is the tridiagonal matrix solver for the EBIED Simulator.
//
//  Inputs:     n - number of equations.
```

```
9  //               a - sub-diagonal (means it is the diagonal below the main
       diagonal,
10 //                   -- indexed from 1..n-1).
11 //               b - the main diagonal.
12 //               c - sup-diagonal (means it is the diagonal above the main
       diagonal
13 //                   -- indexed from 0..n-2).
14 //               v - right part.
15 //
16 //  Outputs:     x - the answer.
17 #ifdef XCODE
18 #include "structures.h"
19 #include "prototypes.h"
20 #endif
21
22 void solve_matrix(int n, double *a, double *b, double *c, double *v, double *x) {
23    int i;
24    for (i=1;i<n;i++) {
25       double m = a[i]/b[i-1];
26       b[i] = b[i] - m*c[i-1];
27       v[i] = v[i] - m*v[i-1];
28    }
29    x[n-1] = v[n-1]/b[n-1];
30    for (i=n-2;i>=0;i--) {
31       x[i]=(v[i]-c[i]*x[i+1])/b[i];
32    }
33 }
```

B.16 Surface Evolution

```
1  //
2  //  surface_evolution.c
3  //
4  //  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
5  //
6  //  This is the module which performs surface evolution and interpolation, for
       the EBIED Simulator.
7  //
8  #ifdef XCODE
9  #include <stdlib.h>
10 #include <stdio.h>
11 #include <math.h>
12 #include <assert.h>
13 #include "structures.h"
14 #include "prototypes.h"
15 #endif
16
17 void surface_evolution(Precursor deposit_precursor,
18                        Precursor etch_precursor,
19                        Material lower_material,
```

```
20                          Material upper_material,
21                          int evolve_surface_normal,
22                          int number_of_points,
23                          int order,
24                          int number_of_surface_bins,
25                          double delta_t,
26                          double delta_r,
27                          double layered_material_interface_depth,int time_step,
    Toggle toggle)
28 {
29     int j;
30 
31     int number_of_points_original = number_of_points;
32 
33     if (evolve_surface_normal) {
34         // advance surface in the normal direction
35         for (j=0;j<number_of_surface_bins;j++) {
36             dC[j] = reactive_product_concentration_current_time_step_array[j] -
37                     reactive_product_concentration_previous_time_step_array[j];
38             growth_or_etch_rate_array[j] = dC[j] / delta_t;
39         }
40         int p1 = (number_of_points-1)/2;
41         int p2 = (number_of_points-1)/2;
42         double z_gradient;
43 
44         for (j=0;j<p1;j++) {
45             z_gradient = (z_coordinate_array[j+p2]-z_coordinate_array[-(j-p1)])/(
    r_coordinate_array[j+p2]+r_coordinate_array[-(j-p1)]);
46 
47             if (z_coordinate_array[j] >= layered_material_interface_depth) {
48                 deposit_precursor.reactive_product_density = upper_material.
    density*1.0E-27;
49             } else {
50                 deposit_precursor.reactive_product_density = lower_material.
    density*1.0E-27;
51             }
52 
53             double deposit_atomic_volume = deposit_precursor.
    reactive_product_molecular_mass/deposit_precursor.reactive_product_density;
54             double distance_to_move = dC[j]*deposit_atomic_volume;
55 
56             double angle = atan(-z_gradient);
57             double test_z = distance_to_move*cos(angle)+z_coordinate_array[j];
58 
59             double test_r = distance_to_move*sin(-angle)+r_coordinate_array[j];
60 
61             temporary_z_coordinate_array[j] = test_z;
62             temporary_r_coordinate_array[j] = test_r;
63             //
64 
65             if ((isnan(temporary_z_coordinate_array[j])) || (
    temporary_r_coordinate_array[j] < 0.0)) {
66                 // z_gradient was zero, so evolve in z only
```

```
67                  temporary_z_coordinate_array[j] =
    reactive_product_concentration_current_time_step_array[j]*
    deposit_atomic_volume;
68                  temporary_r_coordinate_array[j] = r_coordinate_array[j];
69              }
70          }
71          for (j=p1;j<number_of_surface_bins;j++) {
72              if (j+p2 > number_of_surface_bins) {
73                  p2--;
74              }
75              z_gradient = (z_coordinate_array[j+p2]-z_coordinate_array[(j-p1)])/(
    r_coordinate_array[j+p2]-r_coordinate_array[(j-p1)]);
76
77              if (z_coordinate_array[j] >= layered_material_interface_depth) {
78                  deposit_precursor.reactive_product_density = upper_material.
    density*1.0E-27;
79              } else {
80                  deposit_precursor.reactive_product_density = lower_material.
    density*1.0E-27;
81              }
82
83              double deposit_atomic_volume = deposit_precursor.
    reactive_product_molecular_mass/deposit_precursor.reactive_product_density;
84              double distance_to_move = dC[j]*deposit_atomic_volume;
85
86              double angle = atan(-z_gradient);
87              double test_z = distance_to_move*cos(angle)+z_coordinate_array[j];
88
89              double test_r = distance_to_move*sin(-angle)+r_coordinate_array[j];
90
91              temporary_z_coordinate_array[j] = test_z;
92              temporary_r_coordinate_array[j] = test_r;
93              //
94
95              if ((isnan(temporary_z_coordinate_array[j])) || (
    temporary_r_coordinate_array[j] < 0.0)) {
96                  // z_gradient was zero, so evolve in z only
97                  temporary_z_coordinate_array[j] =
    reactive_product_concentration_current_time_step_array[j]*
    deposit_atomic_volume;
98                  temporary_r_coordinate_array[j] = r_coordinate_array[j];
99              }
100         }
101         // completed moving points in the surface normal direction
102
103         //remap points in decending r, effectively sorting the array.
104         double temp_r,temp_z,temp_Nd,temp_Ne,temp_ND;
105         int max_index = 0;
106         double maxz = temporary_z_coordinate_array[0];
107         for (j=1; j<number_of_surface_bins; j++) {
108             if (maxz < temporary_z_coordinate_array[j]) {
109                 maxz = temporary_z_coordinate_array[j];
110                 max_index = j;
```

```
111             }
112         }
113         if (max_index > 0) {
114             for (j=1;j<max_index;j++) {
115                 if (temporary_r_coordinate_array[j] <
    temporary_r_coordinate_array[j-1]) {
116                     temp_r = temporary_r_coordinate_array[j-1];
117                     temp_z = temporary_z_coordinate_array[j-1];
118                     temp_Nd =
    deposit_precursor_gas_concentration_current_time_step_array[j-1];
119                     temp_Ne =
    etch_precursor_gas_concentration_current_time_step_array[j-1];
120                     temp_ND =
    reactive_product_concentration_current_time_step_array[j-1];
121                     //
122                     temporary_r_coordinate_array[j-1] =
    temporary_r_coordinate_array[j];
123                     temporary_z_coordinate_array[j-1] =
    temporary_z_coordinate_array[j];
124                     deposit_precursor_gas_concentration_current_time_step_array[j
    -1] = deposit_precursor_gas_concentration_current_time_step_array[j];
125                     etch_precursor_gas_concentration_current_time_step_array[j-1]
     = etch_precursor_gas_concentration_current_time_step_array[j];
126                     reactive_product_concentration_current_time_step_array[j-1] =
     reactive_product_concentration_current_time_step_array[j];
127                     //
128                     temporary_r_coordinate_array[j] = temp_r;
129                     temporary_z_coordinate_array[j] = temp_z;
130                     deposit_precursor_gas_concentration_current_time_step_array[j
    ] = temp_Nd;
131                     etch_precursor_gas_concentration_current_time_step_array[j] =
     temp_Ne;
132                     reactive_product_concentration_current_time_step_array[j] =
    temp_ND;
133                 }
134             }
135         }
136
137         // interpolation which remaps the data point locations based
138         //  on the correct area to an annulus to a flat surface
139         // stash values into temporary arrays
140         for (j=0;j<number_of_surface_bins;j++) {
141             temporary_etch_precursor_gas_concentration_current_time_step_array_y[
    j] = etch_precursor_gas_concentration_current_time_step_array[j];
142
    temporary_deposit_precursor_gas_concentration_current_time_step_array_y[j] =
    deposit_precursor_gas_concentration_current_time_step_array[j];
143             temporary_reactive_product_concentration_current_time_step_array_y[j]
     = reactive_product_concentration_current_time_step_array[j];
144             temporary_etch_precursor_gas_concentration_current_time_step_array_x[
    j] = r_coordinate_array[j];
```

```
145
    temporary_deposit_precursor_gas_concentration_current_time_step_array_x[j] =
    r_coordinate_array[j];
146             temporary_reactive_product_concentration_current_time_step_array_x[j]
     = r_coordinate_array[j];
147         }
148
149         // move the surface
150         primary_interpolation(&temporary_r_coordinate_array,
151                               &temporary_z_coordinate_array,
152                               number_of_surface_bins,
153                               delta_r,
154                               number_of_points_original);
155
156         // move all the concentrations the same amount the surface moved
157         secondary_interpolation(
    temporary_etch_precursor_gas_concentration_current_time_step_array_x,
158                                 &
    temporary_etch_precursor_gas_concentration_current_time_step_array_y,
159
    temporary_etch_precursor_gas_concentration_current_time_step_array_y,
160                                 temporary_r_coordinate_array,
161                                 number_of_surface_bins,number_of_points_original,
    order,delta_r);
162
163         secondary_interpolation(
    temporary_deposit_precursor_gas_concentration_current_time_step_array_x,
164                                 &
    temporary_deposit_precursor_gas_concentration_current_time_step_array_y,
165
    temporary_deposit_precursor_gas_concentration_current_time_step_array_y,
166                                 temporary_r_coordinate_array,
167                                 number_of_surface_bins,number_of_points_original,
    order,delta_r);
168
169         secondary_interpolation(
    temporary_reactive_product_concentration_current_time_step_array_x,
170                                 &
    temporary_reactive_product_concentration_current_time_step_array_y,
171
    temporary_reactive_product_concentration_current_time_step_array_y,
172                                 temporary_r_coordinate_array,
173                                 number_of_surface_bins,number_of_points_original,
    order,delta_r);
174
175
176         // copy values back to original arrays
177         for (j=0;j<number_of_surface_bins;j++) {
178             z_coordinate_array[j] = temporary_z_coordinate_array[j];
179             etch_precursor_gas_concentration_previous_two_time_step_array[j] =
    etch_precursor_gas_concentration_previous_time_step_array[j];
180             etch_precursor_gas_concentration_previous_time_step_array[j] =
    temporary_etch_precursor_gas_concentration_current_time_step_array_y[j];
```

```
181             deposit_precursor_gas_concentration_previous_two_time_step_array[j] =
     deposit_precursor_gas_concentration_previous_time_step_array[j];
182             deposit_precursor_gas_concentration_previous_time_step_array[j] =
    temporary_deposit_precursor_gas_concentration_current_time_step_array_y[j];
183             reactive_product_concentration_previous_time_step_array[j] =
    temporary_reactive_product_concentration_current_time_step_array_y[j];
184             r_coordinate_array[j] = temporary_r_coordinate_array[j];
185         }
186
187
188
189     } else {
190         //
191         // advance surface in z only (straight up or down)
192         //
193         for (j=0;j<number_of_surface_bins;j++) {
194             growth_or_etch_rate_array[j] = (
    reactive_product_concentration_current_time_step_array[j]-
    reactive_product_concentration_previous_time_step_array[j])/delta_t;
195             etch_precursor_gas_concentration_previous_time_step_array[j] =
    etch_precursor_gas_concentration_current_time_step_array[j];
196             deposit_precursor_gas_concentration_previous_time_step_array[j] =
    deposit_precursor_gas_concentration_current_time_step_array[j];
197             reactive_product_concentration_previous_time_step_array[j] =
    reactive_product_concentration_current_time_step_array[j];
198             if (z_coordinate_array[j] >= layered_material_interface_depth) {
199                 deposit_precursor.reactive_product_density = upper_material.
    density*1.0E-27;
200             } else {
201                 deposit_precursor.reactive_product_density = lower_material.
    density*1.0E-27;
202             }
203
204             double deposit_atomic_volume = deposit_precursor.
    reactive_product_molecular_mass/deposit_precursor.reactive_product_density;
205             z_coordinate_array[j] =
    reactive_product_concentration_current_time_step_array[j]*
    deposit_atomic_volume;
206         }
207
208         // (no remapping is needed here) - this matches the functionality of the
    Mathematica code from Charlene Lobo
209         // (no interpolation needed here)
210     }
211 }
```

B.17 Print to Logfile

```
1 //
2 //  logfile_printf.c
```

```
//
//  Copyright (c) 2015 University of Technology, Sydney. All rights reserved.
//
//  this module outputs the printf command to a logfile aswell as the default
    print to the terminal.
//
#ifdef XCODE
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "structures.h"
#include "prototypes.h"
#endif

void logfile_printf(const char *fmt, ...) {
    va_list args;
    size_t  len;
    char    *space;
    FILE *fp;

    va_start(args, fmt);
    len = vsnprintf(0, 0, fmt, args);
    va_end(args);
    if ((space = malloc(len + 1)) != 0) {
        va_start(args, fmt);
        vsnprintf(space, len+1, fmt, args);
        va_end(args);
        fp=fopen("./outputs/logfile.txt", "a");
        fprintf(fp,"%s",space);
        printf("%s",space);
        fflush(NULL);
        fclose(fp);
        free(space);
    }
}
```

Appendix C

Model Derivations

C.1 EBID/EBIE - Uniform Grid Spacing Derivation

This is the derivation of the EBID model with uniform grid spacing in the form required by the Crank-Nicholson method.

$$\frac{\partial N_d(r,t)}{\partial t} = s_d F_d[1 - N_d(r,t)A_d] - \frac{N_d(r,t)}{\tau_d} + D_d\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right] - \sigma_d f(r) N_d(r,t) \tag{C.1}$$

First we simplify the equation by grouping together like terms in Equation C.2.

$$\frac{\partial N_d(r,t)}{\partial t} = s_d F_d + N_d(r,t)\left[-s_d F_d A_d - \frac{1}{\tau_d} - \sigma_d f(r)\right] + D_d\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right] \tag{C.2}$$

We must then discretise the partial differentials using the forward and backward time centered space scheme.

$$
\begin{aligned}
\frac{N_d(r,t+1)-N_d(r,t)}{\Delta t} = {} & s_dF_d + \left(\frac{N_d(r,t+1)+N_d(r,t)}{2}\right)\left[-s_dF_dA_d - \frac{1}{\tau_d} - \sigma_d f(r)\right] \\
& + \frac{D_d}{2}\left[\frac{N_d(r+1,t+1)-2N_d(r,t+1)+N_d(r-1,t+1)}{(\Delta r)^2}\right. \\
& \qquad \left. + \frac{N_d(r+1,t)-2N_d(r,t)+N_d(r-1,t)}{(\Delta r)^2}\right] \\
& + \frac{D_d}{2r}\left[\frac{N_d(r+1,t+1)-N_d(r-1,t+1)}{2\Delta r}\right. \\
& \qquad \left. + \frac{N_d(r+1,t)-N_d(r-1,t)}{2\Delta r}\right]
\end{aligned}
\tag{C.3}
$$

The forward time step terms are positioned on the LHS of the equation and the backward time step terms on the RHS, in Equation C.4

$$
\begin{aligned}
& N_d(r,t+1)\left[\frac{1}{\Delta t} + \frac{s_dF_dA_d}{2} + \frac{1}{2\tau_d} + \frac{\sigma_d f(r)}{2}\right] \\
& - \frac{D_d}{2}\left[\frac{N_d(r+1,t+1)-2N_d(r,t+1)+N_d(r-1,t+1)}{(\Delta r)^2}\right] \\
& - \frac{D_d}{2r}\left[\frac{N_d(r+1,t+1)-N_d(r-1,t+1)}{2\Delta r}\right] \\
& = \\
& s_dF_d + N_d(r,t)\left[\frac{1}{\Delta t} - \frac{s_dF_dA_d}{2} - \frac{1}{2\tau_d} - \frac{\sigma_d f(r)}{2}\right] \\
& + \frac{D_d}{2}\left[\frac{N_d(r+1,t)-2N_d(r,t)+N_d(r-1,t)}{(\Delta r)^2}\right] \\
& + \frac{D_d}{2r}\left[\frac{N_d(r+1,t)-N_d(r-1,t)}{2\Delta r}\right]
\end{aligned}
\tag{C.4}
$$

Like terms in space are then grouped together in Equation C.5

$$
\begin{aligned}
&N_d(r+1,t+1)\left[-\frac{D_d}{2(\Delta r)^2}-\frac{D_d}{4r\Delta r}\right]+\\
&N_d(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}+\frac{D_d}{(\Delta r)^2}\right]+\\
&N_d(r-1,t+1)\left[-\frac{D_d}{2(\Delta r)^2}+\frac{D_d}{4r\Delta r}\right]\\
&=\\
s_dF_d+&N_d(r+1,t)\left[\frac{D_d}{2(\Delta r)^2}+\frac{D_d}{4r\Delta r}\right]+\\
&N_d(r,t)\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}-\frac{D_d}{(\Delta r)^2}\right]+\\
&N_d(r-1,t)\left[\frac{D_d}{2(\Delta r)^2}-\frac{D_d}{4r\Delta r}\right]
\end{aligned}
\tag{C.5}
$$

Equation C.5 is solved as a tridiagonal matrix, with the most common form being $A \cdot x = B$, where the LHS is contained within A and the RHS is contained within B. The breakdown of Equation C.5 into the required matrix form is done in Equation C.6.

$$
[A][N(t+1)] = [s_dF_d] + [B][N(t)] \tag{C.6}
$$

where the matrices $[A]$, $[B]$, and $[s_dF_d]$ are

$$
[A] = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ a & b & c & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & a & b & c \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix} \tag{C.7}
$$

$$
a = -\frac{D_d}{2(\Delta r)^2} + \frac{D_d}{4r\Delta r} \tag{C.8}
$$

$$
b = \frac{1}{\Delta t} + \frac{s_dF_dA_d}{2} + \frac{1}{2\tau_d} + \frac{\sigma_d f(r)}{2} + \frac{D_d}{(\Delta r)^2} \tag{C.9}
$$

$$
a = -\frac{D_d}{2(\Delta r)^2} - \frac{D_d}{4r\Delta r} \tag{C.10}
$$

$$[B] = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ d & e & f & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & d & e & f \\ 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{C.11}$$

$$d = \frac{D_d}{2(\Delta r)^2} - \frac{D_d}{4r\Delta r} \tag{C.12}$$

$$e = \frac{1}{\Delta t} - \frac{s_d F_d A_d}{2} - \frac{1}{2\tau_d} - \frac{\sigma_d f(r)}{2} - \frac{D_d}{(\Delta r)^2} \tag{C.13}$$

$$f = \frac{D_d}{2(\Delta r)^2} + \frac{D_d}{4r\Delta r} \tag{C.14}$$

$$[s_d F_d] = \begin{bmatrix} 0 \\ s_d F_d \\ \vdots \\ s_d F_d \\ 0 \end{bmatrix} \tag{C.15}$$

The first and last points in each matrix are set boundary conditions required for a zero gradient concentration change at these end points.

C.1.1 Von Neumann Stability

The stability of the EBID model is derived in the following section based on the Von Neumann stability criteria, that the amplification factor $|\epsilon|$ is less than 1[84].

The derivation of the stability follows on from Equation C.5 where $N_d(r,t)$ is replaced by $\epsilon^t e^{ikr\Delta r}$, where k is the spatial wave number.

$$\begin{aligned}
&\epsilon^{t+1}e^{ik(r+1)\Delta r}\left[-\frac{D_d}{2(\Delta r)^2}-\frac{D_d}{4r\Delta r}\right]+\\
&\epsilon^{t+1}e^{ikr\Delta r}\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}+\frac{D_d}{(\Delta r)^2}\right]+\\
&\epsilon^{t+1}e^{ik(r-1)\Delta r}\left[-\frac{D_d}{2(\Delta r)^2}+\frac{D_d}{4r\Delta r}\right]\\
&=\\
s_dF_d+&\epsilon^{t}e^{ik(r+1)\Delta r}\left[\frac{D_d}{2(\Delta r)^2}+\frac{D_d}{4r\Delta r}\right]+\\
&\epsilon^{t}e^{ikr\Delta r}\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}-\frac{D_d}{(\Delta r)^2}\right]+\\
&\epsilon^{t}e^{ik(r-1)\Delta r}\left[\frac{D_d}{2(\Delta r)^2}-\frac{D_d}{4r\Delta r}\right]
\end{aligned} \tag{C.16}$$

Equation C.16 is divided by $e^{ikr\Delta r}$. Note, the s_dF_d constant term is assumed to be ignored and therefore removed.

$$\begin{aligned}
&\epsilon^{t+1}e^{ik\Delta r}\left[-\frac{D_d}{2(\Delta r)^2}-\frac{D_d}{4r\Delta r}\right]+\\
&\epsilon^{t+1}\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}+\frac{D_d}{(\Delta r)^2}\right]+\\
&\epsilon^{t+1}e^{-ik\Delta r}\left[-\frac{D_d}{2(\Delta r)^2}+\frac{D_d}{4r\Delta r}\right]\\
&=\\
&\epsilon^{t}e^{ik\Delta r}\left[\frac{D_d}{2(\Delta r)^2}+\frac{D_d}{4r\Delta r}\right]+\\
&\epsilon^{t}\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}-\frac{D_d}{(\Delta r)^2}\right]+\\
&\epsilon^{t}e^{-ik\Delta r}\left[\frac{D_d}{2(\Delta r)^2}-\frac{D_d}{4r\Delta r}\right]
\end{aligned} \tag{C.17}$$

The $e^{ik\Delta r}$ terms are replaced to be more consistent by using the identities in Equations C.18, C.19, and C.20.

$$\cos(k\Delta r)=\frac{e^{ik\Delta r}+e^{-ik\Delta r}}{2} \tag{C.18}$$

$$\sin^2(\frac{k\Delta r}{2})=\frac{1-\cos(k\Delta r)}{2} \tag{C.19}$$

$$\sin(k\Delta r) = \frac{e^{ik\Delta r} - e^{-ik\Delta r}}{2i} \tag{C.20}$$

$$\begin{gathered}\epsilon^{t+1}\left[\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) - i\frac{D_d}{2r\Delta r}\sin(k\Delta r) + \frac{1}{\Delta t} + \frac{s_dF_dA_d}{2} + \frac{1}{2\tau_d} + \frac{\sigma_d f(r)}{2}\right]\\ = \\ \epsilon^{t}\left[-\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) + i\frac{D_d}{2r\Delta r}\sin(k\Delta r) + \frac{1}{\Delta t} - \frac{s_dF_dA_d}{2} - \frac{1}{2\tau_d} - \frac{\sigma_d f(r)}{2}\right]\end{gathered} \tag{C.21}$$

Equation C.21 is simplified by performing the substitution, $\beta = \frac{s_dF_dA_d}{2} + \frac{1}{2\tau_d} + \frac{\sigma_d f(r)}{2}$, on these constant terms.

$$\begin{gathered}\epsilon^{t+1}\left[\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) - i\frac{D_d}{2r\Delta r}\sin(k\Delta r) + \frac{1}{\Delta t} + \beta\right]\\ = \\ \epsilon^{t}\left[-\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) + i\frac{D_d}{2r\Delta r}\sin(k\Delta r) + \frac{1}{\Delta t} - \beta\right]\end{gathered} \tag{C.22}$$

By rearranging the equation the amplification factor can be determined using the equation, $|\epsilon| = |\frac{\epsilon^{t+1}}{\epsilon^t}|$

$$\frac{\epsilon^{t+1}}{\epsilon^t} = \frac{-\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) + i\frac{D_d}{2r\Delta r}\sin(k\Delta r) + \frac{1}{\Delta t} - \beta}{\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) - i\frac{D_d}{2r\Delta r}\sin(k\Delta r) + \frac{1}{\Delta t} + \beta} \tag{C.23}$$

However, it is noted that this equation contains a complex number in the denominator, therefore to further this derivation we must multiply by the complex conjugate to remove it. The following series of algebraic steps are rather complex, so to provide some perspective on the process see Equation C.24

$$\frac{a+ib}{c+id} = \frac{a+ib}{c+id}\cdot\frac{c-id}{c-id} = \frac{(ac+bd)+i(bc-ad)}{c^2+d^2} = \frac{ac+bd}{c^2+d^2} + i\frac{bc+ad}{c^2+d^2} \tag{C.24}$$

where

$$a = -\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) + \frac{1}{\Delta t} - \beta \tag{C.25}$$

$$b = \frac{D_d}{2r\Delta r}\sin(k\Delta r) \tag{C.26}$$

$$c = \frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r) + \frac{1}{\Delta t} + \beta \tag{C.27}$$

$$d = -\frac{D_d}{2r\Delta r}\sin(k\Delta r) \tag{C.28}$$

First the denominator is calculated

$$(\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}+\beta)^2+(-\frac{D_d}{2r\Delta r}\sin(k\Delta r))^2 \tag{C.29}$$

$$\frac{4D_d^2}{(\Delta r)^4}\sin^4(k\Delta r)+\frac{4D_d}{(\Delta r)^2\Delta t}\sin^2(k\Delta r)+\frac{4D_d\beta}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{(\Delta t)^2}+\frac{2\beta}{\Delta t}+\beta^2+\frac{D_d^2}{4r^2(\Delta r)^2}\sin^2(k\Delta r) \tag{C.30}$$

$$\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}+\beta+\frac{D_d}{16r^2}\right]+\frac{1}{(\Delta t)^2}+\frac{2\beta}{\Delta t}+\beta^2 \tag{C.31}$$

and second the numerator

$$\begin{aligned}&\left[(-\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}-\beta)\cdot(\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}+\beta)\right.\\&\left.+(\frac{D_d}{2r\Delta r}\sin(k\Delta r))\cdot(-\frac{D_d}{2r\Delta r}\sin(k\Delta r))\right]\\&+i\left[(\frac{D_d}{2r\Delta r}\sin(k\Delta r))\cdot(\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}+\beta)\right.\\&\left.-(-\frac{2D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}-\beta)\cdot(-\frac{D_d}{2r\Delta r}\sin(k\Delta r))\right]\end{aligned} \tag{C.32}$$

$$\begin{aligned}&\left[-\frac{4D_d^2}{(\Delta r)^4}\sin^4(k\Delta r)-\frac{4D_d\beta}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{(\Delta t)^2}-\beta^2-\frac{D_d^2}{4r^2(\Delta r)^2}\sin^2(k\Delta r)\right]\\&+i\left[\frac{D_d}{r\Delta r\Delta t}\sin(k\Delta r)\right]\end{aligned} \tag{C.33}$$

$$\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[-\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r)-\beta-\frac{D_d}{16r^2}\right]+\frac{1}{(\Delta t)^2}-\beta^2+i\left[\frac{D_d}{r\Delta r\Delta t}\sin(k\Delta r)\right] \tag{C.34}$$

Finally the previous two sections are combined with the real component

$$\frac{\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[-\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r)-\beta-\frac{D_d}{16r^2}\right]+\frac{1}{(\Delta t)^2}-\beta^2}{\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}+\beta+\frac{D_d}{16r^2}\right]+\frac{1}{(\Delta t)^2}+\frac{2\beta}{\Delta t}+\beta^2} \tag{C.35}$$

and the imaginary component

$$i\frac{\frac{D_d}{r\Delta r\Delta t}\sin(k\Delta r)}{\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r)+\frac{1}{\Delta t}+\beta+\frac{D_d}{16r^2}\right]+\frac{1}{(\Delta t)^2}+\frac{2\beta}{\Delta t}+\beta^2} \tag{C.36}$$

Since we have eliminated the imaginary term from the denominator the magnitude of the amplification factor can be calculated using Equation C.37, where x is the real component

and y is the imaginary component

$$|\epsilon| = \sqrt{x^2 + y^2} \tag{C.37}$$

We will first calculate the x^2 term

$$\frac{\left[\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[-\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r) - \beta - \frac{D_d}{16r^2}\right] + \frac{1}{(\Delta t)^2} - \beta^2\right]^2}{\left[\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r) + \frac{1}{\Delta t} + \beta + \frac{D_d}{16r^2}\right] + \frac{1}{(\Delta t)^2} + \frac{2\beta}{\Delta t} + \beta^2\right]^2} \tag{C.38}$$

$$\frac{\begin{array}{r}(4r^2((\Delta r)^4(-1+\beta^2(\Delta t)^2) + 4\beta D_d(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r) + 4D_d^2(\Delta t)^2\sin^2(k\Delta r)) \\ + D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r))^2\end{array}}{(4r^2((\Delta r)^2(1+\beta\Delta t) + 2D_d\Delta t\sin^2(k\Delta r))^2 + D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r))^2} \tag{C.39}$$

and then the y^2 term

$$\frac{\left[\frac{D_d}{r\Delta r\Delta t}\sin(k\Delta r)\right]^2}{\left[\frac{4D_d}{(\Delta r)^2}\sin^2(k\Delta r)\left[\frac{D_d}{(\Delta r)^2}\sin^2(k\Delta r) + \frac{1}{\Delta t} + \beta + \frac{D_d}{16r^2}\right] + \frac{1}{(\Delta t)^2} + \frac{2\beta}{\Delta t} + \beta^2\right]^2} \tag{C.40}$$

$$\frac{16D_d^2(\Delta r)^6(\Delta t)^2 r^2\sin^2(k\Delta r)}{(4r^2((\Delta r)^2(1+\beta\Delta t) + 2D_d\Delta t\sin^2(k\Delta r))^2 + D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r))^2} \tag{C.41}$$

Adding x^2 and y^2 together we obtain

$$\frac{4r^2((\Delta r)^2(-1+\beta\Delta t) + 2D_d\Delta t\sin^2(k\Delta r))^2 + D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r)}{4r^2((\Delta r)^2(1+\beta\Delta t) + 2D_d\Delta t\sin^2(k\Delta r))^2 + D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r)} \tag{C.42}$$

Equation C.42 is still a very complex equation with a large number of terms, therefore to provide some simplicity we expand and rearrange the equation also re-introducing the

square root term

$$|\epsilon| = \sqrt{\frac{D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r) + 16D_d^2(\Delta t)^2 r^2\sin^4(k\Delta r) + 16\beta D_d(\Delta r)^2(\Delta t)^2 r^2\sin^2(k\Delta r) + 4\beta^2(\Delta r)^4(\Delta t)^2 r^2 + 4(\Delta r)^4 r^2 - 16D_d(\Delta r)^2\Delta t r^2\sin^2(k\Delta r) - 8\beta(\Delta r)^4\Delta t r^2}{D_d^2(\Delta r)^2(\Delta t)^2\sin^2(k\Delta r) + 16D_d^2(\Delta t)^2 r^2\sin^4(k\Delta r) + 16\beta D_d(\Delta r)^2(\Delta t)^2 r^2\sin^2(k\Delta r) + 4\beta^2(\Delta r)^4(\Delta t)^2 r^2 + 4(\Delta r)^4 r^2 + 16D_d(\Delta r)^2\Delta t r^2\sin^2(k\Delta r) + 8\beta(\Delta r)^4\Delta t r^2}} \tag{C.43}$$

From Equation C.43 the condition $|\epsilon| \leq 1$ is only true when two conditions are met. First, the numerator must always be less than the denominator, which is evident with two negative terms only appearing in the numerator. Second, that these negative terms do not result in a negative numerator, which is true under the following reasonable assumptions (with explanations):

- $\Delta t > 0$, the size of the simulation time step can only have a positive non-zero value
- $\Delta r > 0$, the size of the surface bins can only have a positive non-zero value
- $D_d > 0$, regardless of how slow diffusion may be at low temperatures it will still have a value greater than zero
- $\beta > 0$, from earlier $\beta = \frac{s_d F_d A_d}{2} + \frac{1}{2\tau_d} + \frac{\sigma_d f(r)}{2}$, even at no precursor flux, $F_d = 0$, and no electron flux $F(r) = 0$, desorption will still occur no matter the temperature resulting in a value greater than zero
- $r > 0$, the first surface bin is always $1 \times \Delta r$ and therefore greater than zero

C.2 EBID/EBIE - Non-Uniform Grid Spacing Derivation

This is the derivation of the EBID model with non-uniform grid spacing in the form required by the Crank-Nicholson Finite Difference Method [81], which employs the ideas presented in Section 3.3.2.1 and Sobey [94]. The complete derivation by Sobey [94] of non-uniform grid spacings is omitted here but the prominent equation, 15, is shown below

in Equation C.44, which expresses the discretisation of second partial derivative.

$$\frac{\partial^2 C}{\partial x^2}\Big|^n = \frac{1}{(\Delta x)^2}\left[\frac{1}{1+A_i}C_{i-1}^n - \frac{2}{1-A_i^2}C_i^n + \frac{1}{1-A_i}C_{i+1}^n\right] \tag{C.44}$$

Applying this asymmetry parameter to the first spatial derivative, we obtain Equation C.45.

$$\frac{\partial C}{\partial x}\Big|^n = \frac{1}{2\Delta x}\left[\frac{1}{1+A_i}C_{i-1}^n - \frac{1}{1-A_i}C_{i+1}^n\right] \tag{C.45}$$

With the definition of the first and second spatial derivatives, the derivation of the EBID model with non-uniform grid spacing is as follows, where r is equivalent to x and t is equivalent to n, however in the radial coordinate system.

$$\frac{\partial N_d(r,t)}{\partial t} = s_d F_d[1 - N_d(r,t)A_d] - \frac{N_d(r,t)}{\tau_d} + D\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right] - \sigma_d f(r) N_d(r,t) \tag{C.46}$$

We begin again by simplifing Equation C.46 by grouping together like terms in Equation C.47.

$$\frac{\partial N_d(r,t)}{\partial t} = s_d F_d + N_d(r,t)\left[-s_d F_d A_d - \frac{1}{\tau_d} - \sigma_d f(r)\right] + D\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right] \tag{C.47}$$

We again discretise the partial differential using the forward and backward time centered space scheme.

$$\begin{aligned}
\frac{N_d(r,t+1) - N_d(r,t)}{\Delta t} &= s_d F_d + \left(\frac{N_d(r,t+1) + N_d(r,t)}{2}\right)\left[-s_d F_d A_d - \frac{1}{\tau_d} - \sigma_d f(r)\right] \\
&+ \frac{D}{2(\Delta r)^2}\Bigg[\frac{1}{1-A_i}N_d(r+1,t+1) - \frac{2}{1-A_i^2}N_d(r,t+1) \\
&\qquad + \frac{1}{1+A_i}N_d(r-1,t+1) \\
&\qquad + \frac{1}{1-A_i}N_d(r+1,t) - \frac{2}{1-A_i^2}N_d(r,t) \\
&\qquad + \frac{1}{1+A_i}N_d(r-1,t)\Bigg] \\
&+ \frac{D}{4r\Delta r}\Bigg[\frac{1}{1-A_i}N_d(r+1,t+1) - \frac{1}{1+A_i}N_d(r-1,t+1) \\
&\qquad + \frac{1}{1-A_i}N_d(r+1,t) - \frac{1}{1+A_i}N_d(r-1,t)\Bigg]
\end{aligned} \tag{C.48}$$

The forward time step terms are positioned on the LHS of the equation and the backward time step terms on the RHS, in Equation C.49

$$
\begin{aligned}
&N_d(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}\right] \\
&-\frac{D}{2(\Delta r)^2}\left[\frac{1}{1-A_i}N_d(r+1,t+1)-\frac{2}{1-A_i^2}N_d(r,t+1)+\frac{1}{1+A_i}N_d(r-1,t+1)\right] \\
&-\frac{D}{4r\Delta r}\left[\frac{1}{1-A_i}N_d(r+1,t+1)-\frac{1}{1+A_i}N_d(r-1,t+1)\right] \\
&=s_dF_d+N_d(r,t)\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}\right] \\
&\quad+\frac{D}{2(\Delta r)^2}\left[\frac{1}{1-A_i}N_d(r+1,t)-\frac{2}{1-A_i^2}N_d(r,t)+\frac{1}{1+A_i}N_d(r-1,t)\right] \\
&\quad+\frac{D}{4r\Delta r}\left[\frac{1}{1-A_i}N_d(r+1,t)-\frac{1}{1+A_i}N_d(r-1,t)\right]
\end{aligned}
\tag{C.49}
$$

Like terms in space are again grouped together in Equation C.50

$$
\begin{aligned}
&N_d(r+1,t+1)\left[\frac{1}{1-A_i}\right]\left[-\frac{D}{2(\Delta r)^2}-\frac{D}{4r\Delta r}\right]+ \\
&N_d(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}+\frac{2}{1-A_i^2}\frac{D}{2(\Delta r)^2}\right]+ \\
&N_d(r-1,t+1)\left[\frac{1}{1+A_i}\right]\left[-\frac{D}{2(\Delta r)^2}+\frac{D}{4r\Delta r}\right] \\
&= \\
&s_dF_d+N_d(r+1,t)\left[\frac{1}{1-A_i}\right]\left[\frac{D}{2(\Delta r)^2}+\frac{D}{4r\Delta r}\right]+ \\
&N_d(r,t)\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}-\frac{2}{1-A_i^2}\frac{D}{2(\Delta r)^2}\right]+ \\
&N_d(r-1,t)\left[\frac{1}{1+A_i}\right]\left[\frac{D}{2(\Delta r)^2}-\frac{D}{4r\Delta r}\right]
\end{aligned}
\tag{C.50}
$$

Equation C.50 is now in a form similar to Equation C.5 that can be expressed in a tridiagonal matrix form required for the Crank-Nicholson method.

C.3 EBIED - Derivation

This is the derivation of the EBIED model in the form required by the Crank-Nicholson method. The derivation is split into three sections, N_d, deposit precursor gas concentration, N_e, etchant precursor gas concentration and N_D, deposited molecule concentration.

C.3.1 Deposit Precursor Gas Concentration

$$\begin{aligned}\frac{\partial N_d(r,t)}{\partial t} =& s_d F_d[1-(N_e(r,t)A_e + N_d(r,t)A_d)] - \frac{N_d(r,t)}{\tau_d} + D\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right] \\ & - \sigma_d f(r) N_d(r,t) - (\sigma_e f(r) N_e(r,t))\sigma_{rd} N_d(r,t)\end{aligned} \tag{C.51}$$

First we simplify Equation C.51 by grouping together like terms in Equation C.52.

$$\begin{aligned}\frac{\partial N_d(r,t)}{\partial t} =& s_d F_d + N_d(r,t)\left[-s_d F_d A_d - \frac{1}{\tau_d} - \sigma_d f(r) - \sigma_e f(r) N_e(r,t)\sigma_{rd}\right] + N_e(r,t)[-s_d F_d A_e] \\ & + D\left[\frac{\partial^2 N_d(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial N_d(r,t)}{\partial r}\right]\end{aligned} \tag{C.52}$$

We must then discretise the partial differentials using the forward and backward time centered space scheme.

$$\begin{aligned}&\frac{N_d(r,t+1) - N_d(r,t)}{\Delta t} = \\ & s_d F_d + \left(\frac{N_d(r,t+1) + N_d(r,t)}{2}\right)\left[-s_d F_d A_d - \frac{1}{\tau_d} - \sigma_d f(r) - \sigma_e f(r)\sigma_{rd}\left(\frac{N_e(r,t+1) + N_e(r,t)}{2}\right)\right] \\ & \quad + \left(\frac{N_e(r,t+1) + N_e(r,t)}{2}\right)[-s_d F_d A_e] \\ & \quad + \frac{D}{2}\left[\frac{N_d(r+1,t+1) - 2N_d(r,t+1) + N_d(r-1,t+1)}{(\Delta r)^2} \right. \\ & \qquad \left. + \frac{N_d(r+1,t) - 2N_d(r,t) + N_d(r-1,t)}{(\Delta r)^2}\right] \\ & \quad + \frac{D}{2r}\left[\frac{N_d(r+1,t+1) - N_d(r-1,t+1)}{2\Delta r} \right. \\ & \qquad \left. + \frac{N_d(r+1,t) - N_d(r-1,t)}{2\Delta r}\right]\end{aligned} \tag{C.53}$$

The forward time step terms are positioned on the LHS of the equation and the backward time step terms on the RHS, in Equation C.54

$$
\begin{aligned}
&N_d(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}+\frac{\sigma_e f(r)\sigma_{rd}(N_e(r,t+1)+N_e(r,t))}{4}\right] \\
&-\frac{D}{2}\left[\frac{N_d(r+1,t+1)-2N_d(r,t+1)+N_d(r-1,t+1)}{(\Delta r)^2}\right] \\
&-\frac{D}{2r}\left[\frac{N_d(r+1,t+1)-N_d(r-1,t+1)}{2\Delta r}\right] \\
&= \\
&s_dF_d+N_d(r,t)\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}-\frac{\sigma_e f(r)\sigma_{rd}(N_e(r,t+1)+N_e(r,t))}{4}\right] \\
&+(N_e(r,t+1)+N_e(r,t))\left[\frac{-s_dF_dA_e}{2}\right] \\
&+\frac{D}{2}\left[\frac{N_d(r+1,t)-2N_d(r,t)+N_d(r-1,t)}{(\Delta r)^2}\right] \\
&+\frac{D}{2r}\left[\frac{N_d(r+1,t)-N_d(r-1,t)}{2\Delta r}\right]
\end{aligned}
\tag{C.54}
$$

Like terms in space are grouped together in Equation C.55

$$
\begin{aligned}
&N_d(r+1,t+1)\left[-\frac{D}{2(\Delta r)^2}-\frac{D}{4r\Delta r}\right]+ \\
&N_d(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_dF_dA_d}{2}+\frac{1}{2\tau_d}+\frac{\sigma_d f(r)}{2}+\frac{\sigma_e f(r)\sigma_{rd}(N_e(r,t+1)+N_e(r,t))}{4}+\frac{D}{(\Delta r)^2}\right]+ \\
&N_d(r-1,t+1)\left[-\frac{D}{2(\Delta r)^2}+\frac{D}{4r\Delta r}\right] \\
&= \\
&s_dF_d+N_d(r+1,t)\left[\frac{D}{2(\Delta r)^2}+\frac{D}{4r\Delta r}\right]+ \\
&N_d(r,t)\left[\frac{1}{\Delta t}-\frac{s_dF_dA_d}{2}-\frac{1}{2\tau_d}-\frac{\sigma_d f(r)}{2}-\frac{\sigma_e f(r)\sigma_{rd}(N_e(r,t+1)+N_e(r,t))}{4}-\frac{D}{(\Delta r)^2}\right]+ \\
&N_d(r-1,t)\left[\frac{D}{2(\Delta r)^2}-\frac{D}{4r\Delta r}\right]+ \\
&(N_e(r,t+1)+N_e(r,t))\left[\frac{-s_dF_dA_e}{2}\right]
\end{aligned}
\tag{C.55}
$$

Equation C.55 is now in a form similar to Equation C.5 that can be expressed in a tridiagonal matrix form required for the Crank-Nicholson method.

C.3.2 Etchant Precursor Gas Concentration

$$\begin{aligned}\frac{\partial N_e(r,t)}{\partial t} =& s_eF_e[1-(N_e(r,t)A_e+N_d(r,t)A_d)]-\frac{N_e(r,t)}{\tau_e}+D\left[\frac{\partial^2 N_e(r,t)}{\partial r^2}+\frac{1}{r}\frac{\partial N_e(r,t)}{\partial r}\right]\\ &-\sigma_e f(r)N_e(r,t)\end{aligned} \tag{C.56}$$

First we simplify the equation by grouping together like terms in Equation C.57.

$$\begin{aligned}\frac{\partial N_e(r,t)}{\partial t} =& s_eF_e+N_e(r,t)\left[-s_eF_eA_e-\frac{1}{\tau_e}-\sigma_e f(r)\right]+N_d(r,t)[-s_eF_eA_d]\\ &+D\left[\frac{\partial^2 N_e(r,t)}{\partial r^2}+\frac{1}{r}\frac{\partial N_e(r,t)}{\partial r}\right]\end{aligned} \tag{C.57}$$

We must then discretise the partial differentials using the forward and backward time centered space scheme.

$$\begin{aligned}\frac{N_e(r,t+1)-N_e(r,t)}{\Delta t} =& s_eF_e+\left(\frac{N_e(r,t)+N_e(r,t)}{2}\right)\left[-s_eF_eA_e-\frac{1}{\tau_e}-\sigma_e f(r)\right]\\ &+\left(\frac{N_d(r,t)+N_d(r,t)}{2}\right)[-s_eF_eA_d]\\ &+\frac{D}{2}\left[\frac{N_e(r+1,t+1)-2N_e(r,t+1)+N_e(r-1,t+1)}{(\Delta r)^2}\right.\\ &\left.\quad+\frac{N_e(r+1,t)-2N_e(r,t)+N_e(r-1,t)}{(\Delta r)^2}\right]\\ &+\frac{D}{2r}\left[\frac{N_e(r+1,t+1)-N_e(r-1,t+1)}{2\Delta r}\right.\\ &\left.\quad+\frac{N_e(r+1,t)-N_e(r-1,t)}{2\Delta r}\right]\end{aligned} \tag{C.58}$$

The forward time step terms are positioned on the LHS of the equation and the backward time step terms on the RHS, in Equation C.59

$$
\begin{aligned}
&N_e(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_eF_eA_e}{2}+\frac{1}{2\tau_e}+\frac{\sigma_e f(r)}{2}\right]\\
&-\frac{D}{2}\left[\frac{N_e(r+1,t+1)-2N_e(r,t+1)+N_e(r-1,t+1)}{(\Delta r)^2}\right]\\
&-\frac{D}{2r}\left[\frac{N_e(r+1,t+1)-N_e(r-1,t+1)}{2\Delta r}\right]\\
&=\\
&s_eF_e+N_e(r,t)\left[\frac{1}{\Delta t}-\frac{s_eF_eA_e}{2}-\frac{1}{2\tau_e}-\frac{\sigma_e f(r)}{2}\right]\\
&+(N_d(r,t+1)+N_d(r,t))\left[\frac{-s_eF_eA_d}{2}\right]\\
&+\frac{D}{2}\left[\frac{N_e(r+1,t)-2N_e(r,t)+N_e(r-1,t)}{(\Delta r)^2}\right]\\
&+\frac{D}{2r}\left[\frac{N_e(r+1,t)-N_e(r-1,t)}{2\Delta r}\right]
\end{aligned}
\tag{C.59}
$$

Like terms in space are grouped together in Equation C.60

$$
\begin{aligned}
&N_e(r+1,t+1)\left[-\frac{D}{2(\Delta r)^2}-\frac{D}{4r\Delta r}\right]+\\
&N_e(r,t+1)\left[\frac{1}{\Delta t}+\frac{s_eF_eA_e}{2}+\frac{1}{2\tau_e}+\frac{\sigma_e f(r)}{2}+\frac{D}{(\Delta r)^2}\right]+\\
&N_e(r-1,t+1)\left[-\frac{D}{2(\Delta r)^2}+\frac{D}{4r\Delta r}\right]\\
&=\\
&s_eF_e+N_e(r+1,t)\left[\frac{D}{2(\Delta r)^2}+\frac{D}{4r\Delta r}\right]+\\
&N_e(r,t)\left[\frac{1}{\Delta t}-\frac{s_eF_eA_e}{2}-\frac{1}{2\tau_e}-\frac{\sigma_e f(r)}{2}-\frac{D}{(\Delta r)^2}\right]+\\
&N_e(r-1,t)\left[\frac{D}{2(\Delta r)^2}-\frac{D}{4r\Delta r}\right]+\\
&(N_d(r,t+1)+N_d(r,t))\left[\frac{-s_eF_eA_d}{2}\right]
\end{aligned}
\tag{C.60}
$$

Equation C.60 is now in a form similar to Equation C.5 that can be expressed in a tridiagonal matrix form required for the Crank-Nicholson method. It is noted that the final term in Equation C.60 cannot be solved as it depends on the same time step being calculated,

therefore an approximation is made where the final term becomes,

$$N_d(r,t)[-s_e F_e A_d] \tag{C.61}$$

this approximation is then fed back into the EBIED equations and the correct result is calculated.

C.3.3 Deposited Molecule Concentration

$$\frac{\partial N_D(r,t)}{\partial t} = [\sigma_d f(r) N_d(r,t)] - [\sigma_e f(r) N_e(r,t)][1 - \sigma_{rd} N_d(r,t)][\sigma_{rD} N_D(r,t)] \tag{C.62}$$

Due to the first order partial differential we only discretise the equation to be centred in time, in Equation C.63.

$$\frac{N_D(r,t+1) - N_D(r,t)}{\Delta t} = [\sigma_d f(r) N_d(r,t)] - [\sigma_e f(r) N_e(r,t)][1 - \sigma_{rd} N_d(r,t)][\sigma_{rD} N_D(r,t)] \tag{C.63}$$

Collecting the forward and backward terms together, in Equation C.64

$$\frac{N_D(r,t+1)}{\Delta t} = [\sigma_d f(r) N_d(r,t)] - [\sigma_e f(r) N_e(r,t)][1 - \sigma_{rd} N_d(r,t)][\sigma_{rD} N_D(r,t)] + [\frac{N_D(r,t)}{\Delta t}] \tag{C.64}$$

Simplifying, in Equation C.65

$$N_D(r,t{+}1) = \Delta t[[\sigma_d f(r) N_d(r,t)] - [\sigma_e f(r) N_e(r,t)][1 - \sigma_{rd} N_d(r,t)][\sigma_{rD} N_D(r,t)] + [\frac{N_D(r,t)}{\Delta t}]] \tag{C.65}$$

Bibliography

[1] Robert W Christy. Formation of Thin Polymer Films by Electron Bombardment. *J. Appl. Phys.*, 31(9):1680–1683, 1960.

[2] Matthew G Lassiter, Philip D Rack, Home Search, Collections Journals, About Contact, My Iopscience, I P Address, Matthew G Lassiter, and Philip D Rack. Nanoscale electron beam induced etching: a continuum model that correlates the etch profile to the experimental parameters. *Nanotechnology*, 19(45):455306, October 2008.

[3] Milos Toth, Charlene J. Lobo, Gavin Hartigan, and W. Ralph Knowles. Electron flux controlled switching between electron beam induced etching and deposition. *J. Appl. Phys.*, 101(5):54309, 2007.

[4] Ivo Utke, Vinzenz Friedli, Martin Purrucker, and Johann Michler. Resolution in Focused Electron- and Ion-Beam Induced Processing. *J. Vac. Sci. Technol. B*, 25 (6):2219–2223, 2007.

[5] Milos Toth, Charlene Lobo, Vinzenz Friedli, Aleksandra Szkudlarek, and Ivo Utke. Continuum models of focused electron beam induced processing. *Beilstein J. Nanotechnol.*, 6:1518–1540, 2015.

[6] Daryl A Smith, Jason D Fowlkes, and Philip D Rack. Understanding the Kinetics and Nanoscale Morphology of Electron-Beam-Induced Deposition via a Three-Dimensional Monte Carlo Simulation: The Effects of the Precursor Molecule and the Deposited Material. *Small*, 4(9):1382–1389, September 2008.

[7] J. D. Fowlkes, S. J. Randolph, and P. D. Rack. Growth and simulation of high-aspect ratio nanopillars by primary and secondary electron-induced deposition. *J. Vac. Sci. Technol. B Microelectron. Nanom. Struct.*, 23(6):2825, 2005.

[8] Zhi-Quan Liu, Kazutaka Mitsuishi, and Kazuo Furuya. A Dynamic Monte Carlo Study of the in situ Growth of a Substance Deposited Using Electron-Beam-Induced Deposition. *Nanotechnology*, 17(15):3832–3837, July 2006.

[9] N Silvis-Cividjian, C W Hagen, and P Kruit. Spatial Resolution Limits in Electron-Beam-Induced Deposition. *J. Appl. Phys.*, 98(8):84905, 2005.

[10] J Cullen, A Bahm, C J Lobo, M J Ford, and M Toth. Localized Probing of Gas Molecule Adsorption Energies and Desorption Attempt Frequencies. *J. Phys. Chem. C*, 119(28):15948–15953, 2015.

[11] Jared Cullen, Charlene J. Lobo, Michael J. Ford, and Milos Toth. Electron-Beam-Induced Deposition as a Technique for Analysis of Precursor Molecule Diffusion Barriers and Prefactors. *ACS Appl. Mater. Interfaces*, September 2015.

[12] Steven Randolph, Milos Toth, Jared Cullen, Clive Chandler, and Charlene Lobo. Kinetics of Gas Mediated Electron Beam Induced Etching. *Appl. Phys. Lett.*, 99 (21):213103, 2011.

[13] P Hoffmann, I Utke, and V Friedli. Focused electron-and ion-beam induced processes. *Thesis*, 2008.

[14] C Desjonqueres and D Spanjaard. *Concepts in Surface Physics: 2nd Edition*. Springer series in surface sciences. Springer Berlin Heidelberg, 1996. ISBN 9783540586227.

[15] James Bishop, Charlene J. Lobo, Aiden Martin, Mike Ford, Matthew Phillips, and Milos Toth. Role of activated chemisorption in gas-mediated electron beam induced deposition. *Phys. Rev. Lett.*, 109(14):3–6, 2012.

[16] Dongbo Li, Mosha H Zhao, J Garra, A M Kolpak, A M Rappe, D A Bonnell, and J M Vohs. Direct in situ determination of the polarization dependence of physisorption on ferroelectric surfaces. *Nat. Mater.*, 7(6):473–477, May 2008.

[17] Leszek Czepirski, M Balys, and E Nomorowska-Czepirska. Some generalizations of Langmuir adsorption isotherm. *Internet J. Chem.*, 3(14):1099–8292, 2000.

[18] Stephen Brunauer, Paul Hugh Emmett, and Edward Teller. Adsorption of gases in multimolecular layers. *J. Am. Chem. Soc.*, 60(2):309–319, 1938.

[19] Ivo Utke, Patrik Hoffmann, and John Melngailis. Gas-Assisted Focused Electron Beam and Ion Beam Processing and Fabrication. *J. Vac. Sci. {&} Technol. B Microelectron. Nanom. Struct.*, 26(4):1197, 2008.

[20] V Scheuer, H Koops, and T Tschudi. Electron Beam Decomposition of Carbonyls on Silicon. *Microelectron. Eng.*, 5(1-4):423–430, 1986.

[21] a. D. Dubner and a. Wagner. The role of gas adsorption in ion-beam-induced deposition of gold. *J. Appl. Phys.*, 66(2):870–874, 1989.

[22] Kristen Fichthorn and Radu Miron. Thermal Desorption of Large Molecules from Solid Surfaces. *Phys. Rev. Lett.*, 89(19):196103, October 2002.

[23] R Storch, H Stolz, and H W Wassmuth. Desorption kinetics and surface diffusion of potassium, rubidium and cesium on a silicon(111)7 7-surface. *Ann. Phys.*, 504(5): 315–320, 1992.

[24] Nigel Mason, Yukikazu Itikawa, Nigel Mason, and Yukikazu Itikawa. Cross Sections for Electron Collisions with Water Molecules. *J. Phys. Chem. Ref. Data*, 34(1):1, 2005.

[25] Michael Huth, Fabrizio Porrati, Christian Schwalb, Marcel Winhold, Roland Sachser, Maja Dukic, Jonathan Adams, and Georg Fantner. Focused Electron Beam Induced Deposition: A Perspective. *Beilstein J. Nanotechnol.*, 3:597–619, 2012.

[26] a. Chutjian, a. Garscadden, and J. M. Wadehra. Electron attachment to molecules at low electron energies. *Phys. Rep.*, 264(6):393–470, 1996.

[27] S Feil, P Sulzer, a Mauracher, M Beikircher, N Wendt, a Aleem, S Denifl, F Zappa, S Matt-Leubner, a Bacher, S Matejcik, M Probst, P Scheier, and T D Märk. Electron Impact Ionization/Dissociation of Molecules: Production of Energetic Radical Ions and Anions. *J. Phys. Conf. Ser.*, 86:012003, 2007.

[28] R.L. Lariviere Stewart. Insulating Films Formed Under Electron and Ion Bombardment. *Phys. Rev.*, 45(1931):488, April 1934.

[29] C. W. Oatley. The early history of the scanning electron microscope. *J. Appl. Phys.*, 53(2):R1, 1982.

[30] E F BURTON, R S SENNETT, and S G ELLIS. Specimen changes due to electron bombardment in the electron microscope. *Nature*, 160(4069):565–567, October 1947.

[31] A N Broers, W W Molzen, J J Cuomo, and N D Wittels. Electron-beam fabrication of 80-{Å} metal structures. *Appl. Phys. Lett.*, 29(9):596, 1976.

[32] A Botman, M Hesselberth, and J J L Mulders. Improving the conductivity of platinum-containing nano-structures created by electron-beam-induced deposition. *Microelectron. Eng.*, 85(5-6):1139–1142, May 2008.

[33] Willem F van Dorp, Bob van Someren, Cornelis W Hagen, Pieter Kruit, Peter A Crozier, Willem F Van Dorp, Bob Van Someren, Cornelis W Hagen, Pieter Kruit, and Peter A Crozier. Approaching the Resolution Limit of Nanometer-Scale Electron Beam-Induced Deposition. *Nano Lett.*, 5(7):1303–1307, July 2005.

[34] M Toth, C J Lobo, W R Knowles, M R Phillips, M T Postek, and A E Vladar. Nanostructure Fabrication by Ultra-High-Resolution Environmental Scanning Electron Microscopy. *Nano Lett.*, 7(2):525–530, 2007.

[35] Konrad Rykaczewski, Owen J Hildreth, Dhaval Kulkarni, Matthew R Henry, Song-kil Kim, Ching Ping Wong, Vladimir V Tsukruk, and Andrei G Fedorov. Maskless and Resist-Free Rapid Prototyping of Three-Dimensional Structures Through Electron Beam Induced Deposition (EBID) of Carbon in Combination with Metal-Assisted Chemical Etching (MaCE) of Silicon. 2(4):969–973, 2010.

[36] Amalio Fernández-Pacheco, Luis Serrano-Ramón, Jan M Michalik, M Ricardo Ibarra, José M De Teresa, Liam O'Brien, Dorothée Petit, Jihyun Lee, and Russell P Cowburn. Three Dimensional Magnetic Nanowires Grown by Focused Electron-Beam Induced Deposition. *Sci. Rep.*, 3:1–5, March 2013.

[37] G Martin, Damiana Lerose, Christoph Niederberger, Johann Michler, Silke Christiansen, Ivo Utke, Martin Günter Jenke, Damiana Lerose, Christoph Niederberger, Johann Michler, Silke Christiansen, and Ivo Utke. Toward Local Growth of Individual Nanowires on Three-Dimensional Microstructures by Using a Minimally Invasive Catalyst Templating Method. *Nano Lett.*, 11(10):4213–4217, October 2011.

[38] Anastasia V Riazanova, Yuri G M Rikers, Johannes J L Mulders, and Lyubov M Belova. Pattern Shape Control for Heat Treatment Purification of Electron-Beam-Induced Deposition of Gold from the Me 2Au(acac) Precursor. *Langmuir*, 28(14): 6185–6191, April 2012.

[39] Fangfang Wen, Jian Ye, Na Liu, Pol Van Dorpe, Peter Nordlander, and Naomi J Halas. Plasmon Transmutation: Inducing New Modes in Nanoclusters by Adding Dielectric Nanoparticles. *Nano Lett.*, 12(9):5020–5026, September 2012.

[40] V Gopal, V R Radmilovic, C Daraio, S Jin, P D Yang, and E A Stach. Rapid Prototyping of Site-Specific Nanocontacts by Electron and Ion Beam Assisted Direct-Write Nanolithography. *Nano Lett.*, 4(11):2059–2063, January 2004.

[41] Katja Höflich, Ren Bin Yang, Andreas Berger, Gerd Leuchs, and Silke Christiansen. The Direct Writing of Plasmonic Gold Nanostructures by Electron-Beam-Induced Deposition. *Adv. Mater.*, 23(22-23):2657–2661, 2011.

[42] S. J. Randolph, J. D. Fowlkes, and P. D. Rack. Focused, Nanoscale Electron-Beam-Induced Deposition and Etching. *Crit. Rev. Solid State Mater. Sci.*, 31(3):55–89, September 2006.

[43] W. F. van Dorp and C. W. Hagen. A Critical Literature Review of Focused Electron Beam Induced Deposition. *J. Appl. Phys.*, 104(081301):1–42, 2008.

[44] Ivo Utke, Stanislav Moshkalev, and Phillip Russell. *Nanofabrication Using Focused Ion and Electron Beams*. Principles and Applications. Oxford University Press, USA, 2012.

[45] P. D. Rack, S. Randolph, Y. Deng, J. Fowlkes, Y. Choi, and D. C. Joy. Nanoscale electron-beam-stimulated processing. *Appl. Phys. Lett.*, 82(14):2326, 2003.

[46] Jianhua Wang, D P Griffis, R Garcia, and P E Russell. Etching characteristics of chromium thin films by an electron beam induced surface reaction. *Semicond. Sci. Technol.*, 18(4):199–205, April 2003.

[47] S J Randolph, J D Fowlkes, and P D Rack. Focused electron-beam-induced etching of silicon dioxide. *J. Appl. Phys.*, 98(3):34902, 2005.

[48] Theodore E. Madey and J T Yates. Electron-Stimulated Desorption as a Tool for Studies of Chemisorption: A Review. *J. Vac. Sci. Technol.*, 8(4):525, July 1971.

[49] Peter Roediger, Heinz D Wanzenboeck, Gottfried Hochleitner, and Emmerich Bertagnolli. Crystallinity-retaining removal of germanium by direct-write focused electron beam induced etching. *J. Vac. Sci. Technol. B*, 29(4):41801, 2011.

[50] F J Schoenaker, R Córdoba, R Fernández-Pacheco, C Magén, O Stéphan, C Zuriaga-Monroy, M R Ibarra, and J M De Teresa. Focused electron beam induced etching of titanium with XeF2. *Nanotechnology*, 22(26):265304, May 2011.

[51] Aiden A Martin, Matthew R Phillips, and M Toth. Dynamic Surface Site Activation: A Rate Limiting Process in Electron Beam Induced Etching. *ACS Appl. Mater. Interfaces*, 5(16):8002–8007, August 2013.

[52] Aiden A Martin and Milos Toth. Cryogenic Electron Beam Induced Chemical Etching. *ACS Appl. Mater. Interfaces*, 6(21):18457–18460, 2014.

[53] A Shih, J Yater, C Hor, and R Abrams. Secondary electron emission studies. *Appl. Surf. Sci.*, 111:251–258, February 1997.

[54] David C Joy. A Database on Electron-Solid Interactions. *Scanning*, 17(5):270–275, April 2008.

[55] Robert Winkler, Jason Fowlkes, Aleksandra Szkudlarek, Ivo Utke, Philip D Rack, and Harald Plank. The Nanoscale Implications of a Molecular Gas Beam during Electron Beam Induced Deposition. *ACS Appl. Mater. Inter.*, 6(4):2987–2995, February 2014.

[56] P. C. Post, A. Mohammadi-Gheidari, C. W. Hagen, and P. Kruit. Parallel electron-beam-induced deposition using a multi-beam scanning electron microscope. *J. Vac. Sci. {&} Technol. B Microelectron. Nanom. Struct.*, 29(6):06F310, 2011.

[57] Nicholas A Roberts, Jason D Fowlkes, Gregory a Magel, and Philip D Rack. Enhanced material purity and resolution via synchronized laser assisted electron beam induced deposition of platinum. *Nanoscale*, 5(1):408–15, 2013.

[58] L M Belova, Olav Hellwig, Elizabeth Dobisz, and E Dan Dahlberg. Rapid preparation of electron beam induced deposition Co magnetic force microscopy tips with 10 nm spatial resolution. *Rev. Sci. Instrum.*, 83(9):93711, 2012.

[59] a. Notargiacomo, E. Giovine, and L. Di Gaspare. Ion and electron beam deposited masks for pattern transfer by reactive ion etching. *Microelectron. Eng.*, 88(8):2710–2713, August 2011.

[60] Young R Y R Choi, Philip D P D Rack, Steven J S J Randolph, Daryl A D A Smith, and David C D C Joy. Pressure Effect of Growing with Electron Beam-Induced Deposition with Tungsten Hexafluoride and Tetraethylorthosilicate Precursor. *Scanning*, 28(6):311–318, October 2006.

[61] A Ganczarczyk, M Geller, and a Lorke. XeF2 gas-assisted focused-electron-beam-induced etching of GaAs with 30 nm resolution. *Nanotechnology*, 22(4):45301, December 2010.

[62] Young R. Choi, Philip D. Rack, Bernhard Frost, and David C. Joy. Effect of electron beam-induced deposition and etching under bias. *Scanning*, 29(4):171–176, June 2007.

[63] J Bishop, C J Lobo, A Martin, M Ford, M R Phillips, and M Toth. The Role of Activated Chemisorption in Electron Beam Induced Deposition. *Phys. Rev. Lett.*, 109:146103, 2012.

[64] Jason D Fowlkes and Philip D Rack. Fundamental Electron-Precursor-Solid Interactions Derived from Time-Dependent Electron-Beam-Induced Deposition Simulations and Experiments. *ACS Nano*, 4(3):1619–1629, March 2010.

[65] Aleksandra Szkudlarek, Mihai Gabureac, and Ivo Utke. Determination of the Surface Diffusion Coefficient and the Residence Time of Adsorbates via Local Focused Electron Beam Induced Chemical Vapour Deposition. *J. Nanosci. Nanotechnol.*, 11 (9):8074–8078, August 2011.

[66] W. F. van Dorp, J. D. Wnuk, J. M. Gorham, D. H. Fairbrother, T. E. Madey, and C. W. Hagen. Electron Induced Dissociation of Trimethyl (Methylcyclopentadienyl) Platinum (IV): Total Cross Section as a Function of Incident Electron Energy. *J. Appl. Phys.*, 106(7):74903, 2009.

[67] S. J. Randolph, J. D. Fowlkes, and P. D. Rack. Effects of Heat Generation During Electron-Beam-Induced Deposition of Nanostructures. *J. Appl. Phys.*, 97(12): 124312, 2005.

[68] Milos Toth, Charlene J. Lobo, Michael J. Lysaght, András E. Vladár, and Michael T. Postek. Contamination-free imaging by electron induced carbon volatilization in environmental scanning electron microscopy. *J. Appl. Phys.*, 106(3):34306, 2009.

[69] Charlene J Lobo, Milos Toth, Raymond Wagner, Bradley L Thiel, and Michael Lysaght. High Resolution Radially Symmetric Nanostructures from Simultaneous Electron Beam Induced Etching and Deposition. *Nanotechnology*, 19(2):25303, December 2007.

[70] Willem F. van Dorp, Thomas W. Hansen, Jakob B. Wagner, and JeffT T M De Hosson. The role of electron-stimulated desorption in focused electron beam induced deposition. *Beilstein J. Nanotechnol.*, 4(1):474–480, 2013.

[71] Wei Li and David C. Joy. Study of Temperature Influence on Electron Beam Induced Deposition. *J. Vac. Sci. Technol. A*, 24(3):431, 2006.

[72] Steven J Randolph, Aurelien Botman, and Milos Toth. Deposition of Highly Porous Nanocrystalline Platinum on Functionalized Substrates Through Fluorine-Induced Decomposition of \ce{Pt(PF3)4} Adsorbates. *Part. Part. Syst. Charact.*, pages 672–677, June 2013.

[73] L Bernau, M Gabureac, R Erni, and I Utke. Tunable Nanosynthesis of Composite Materials by Electron-Impact Reaction. *Angew Chem Int Ed.*, 49(47):8880–8884, 2010.

[74] D A Smith, J D Fowlkes, and P D Rack. A Nanoscale Three-Dimensional Monte Carlo Simulation of Electron-Beam-Induced Deposition with Gas Dynamics. *Nanotechnology*, 18(26):265308, June 2007.

[75] Daryl A Smith, Jason D Fowlkes, and Philip D Rack. Simulating the Effects of Surface Diffusion on Electron Beam Induced Deposition via a Three-Dimensional Monte Carlo Simulation. *Nanotechnology*, 19(41):415704, September 2008.

[76] N Silvis-Cividjian, C W Hagen, L H A Leunissen, and P Kruit. The role of secondary electrons in electron-beam-induced- deposition spatial resolution. *Microelectron. Eng.*, 61-62:693–699, May 2002.

[77] K Mitsuishi, Z Q Liu, M Shimojo, M Han, and K Furuya. Dynamic profile calculation of deposition resolution by high-energy electrons in electron-beam-induced deposition. *Ultramicroscopy*, 103(1):17–22, April 2005.

[78] Zhi Quan Liu, Kazutaka Mitsuishi, and Kazuo Furuya. Modeling the process of electron-beam-induced deposition by dynamic Monte Carlo simulation. *Japanese J. Appl. Physics, Part 1 Regul. Pap. Short Notes Rev. Pap.*, 44(7 B):5659–5663, 2005.

[79] Konrad Rykaczewski, William B. White, and Andrei G. Fedorov. Analysis of electron beam induced deposition (EBID) of residual hydrocarbons in electron microscopy. *J. Appl. Phys.*, 101(5):1–12, 2007.

[80] Pierre Hovington, Dominique Drouin, Raynald Gauvin, David C. Joy, and Neal Evens. CASINO: A New Monte Carlo Code in C Language for Electron Beam Interaction—Part I: Description of the Program. *Scanning*, 19(1):1–14, 20–28, 29–35, 1997.

[81] J Crank and P Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Adv. Comput. Math.*, 6 (1):207–226, December 1996.

[82] G Sun, C.W. W Trueman, C. Sun, C.W. W Trueman, G Sun, and C.W. W Trueman. Unconditionally stable Crank-Nicolson scheme for solving two-dimensional Maxwell's equations. *Electron. Lett.*, 39(7):595–597, 2003.

[83] G. D. (Gordon D.) Smith. *Numerical solution of partial differential equations : finite difference methods*. Oxford [Oxfordshire] : Clarendon Press ; New York : Oxford University Press, third edition, 1985. ISBN 0198596413.

[84] John D Anderson. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill, Inc., 1 edition, February 1995. ISBN 0-07-001685-2.

[85] H L Thomas. *Elliptic Problems in Linear Differential Equations over a Network*. Watson Sci. Comput. Lab Report, Columbia University, New York, January 1949.

[86] W T Lee. Tridiagonal Matrices: Thomas Algorithm. *MS6021, Sci. Comput. Univ. Limerick*, pages 1–3, November 2011.

[87] Richard Ghez. *Diffusion Phenomena: Cases and Studies* . Springer, January 2011.

[88] Electron Microscope Facility University of Tennessee David C. Joy Oak Ridge National Laboratory Distinguished Scientist and Director, Knoxville and David C Joy. *Monte Carlo Modeling for Electron Microscopy and Microanalysis*. Oxford University Press, April 1995. ISBN 9780195088748.

[89] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000. ISBN 3662042479.

[90] D E Newbury and R L Myklebust. \emph{*Analytical Electron Microscopy*}. Ed. R. H. Geiss, January 1981.

[91] H E Bishop. The History and Development of Monte Carlo Methods for Use in X-Ray Microanalysis. In Kurt F J Heinrich, D E Newbury, and Harvey Yakowitz, editors, *Use Monte Carlo Calc. electron probe Microanal. scanning electron Microsc.*, page 169. National Bureau of Standards Special Publication 460, December 1976.

[92] D C Joy and S Luo. An empirical stopping power relationship for low-energy electrons. *Scanning*, 11(4):176–180, 1989.

[93] M J Berger and S M Seltzer. *Studies in Penetration of Charged Particles in Matter.* Nuclear Science Series Report Number 39. National Academies, December 1964.

[94] R J Sobey. An optimized solution for the diffusion equation on a nonuniform grid. *Int. J. Numer. Methods Eng.*, 20(3):465–477, 1984.

[95] Eric W. Weisstein. Conical Frustum, 2015.

[96] V P Zhdanov. Arrhenius Parameters for Rate-Processes on Solid-Surfaces. *Surf. Sci. Rep.*, 12(5):183–242, January 1991.

[97] T E Madey and J T Yates Jr. Desorption Methods as Probes of Kinetics and Bonding at Surfaces. *Surf. Sci.*, 63(C):203–231, 1977.

[98] Yinghong Lin and David C Joy. A New Examination of Secondary Electron Yield Data. *Surf. Interface Anal.*, 37(11):895–900, 2005.

[99] Juan Shen, Kaliappan Muthukumar, Harald O Jeschke, and Roser Valenti. Physisorption of an Organometallic Platinum Complex on Silica: An Ab Initiostudy. *New J. Phys.*, 14(7):73040, July 2012.

[100] M Toth. Advances in Gas-Mediated Electron Beam-Induced Etching and Related Material Processing Techniques. *Appl. Phys. A*, 117:1623–1629, 2014.

[101] V Friedli and I Utke. Optimized Molecule Supply from Nozzle-Based Gas Injection Systems for Focused Electron- and Ion-Beam Induced Deposition and Etching: Simulation and Experiment. *J. Phys. D*, 42(12):125305, January 2009.

[102] T Bret, I Utke, and P Hoffmann. Influence of the Beam Scan Direction During Focused Electron Beam Induced Deposition of 3D Nanostructures. *Microelectron. Eng.*, 78-79:307–313, January 2005.

[103] J Bishop, M Toth, M Phillips, and C Lobo. Effects of Oxygen on Electron Beam Induced Deposition of \ce{SiO2} Using Physisorbed and Chemisorbed Tetraethoxysilane. *Appl. Phys. Lett.*, 101(21):211605, November 2012.

[104] M J Drinkwine and D Lichtman. Electron Stimulated Desorption: A Critical Review. *Prog. Surf. Sci.*, 8(3):123–142, 1977.

[105] R D Ramsier and J T Yates. Electron-Stimulated Desorption - Principles and Applications. *Surf. Sci. Rep.*, 12(6-8):243–378, 1991.

[106] E G Seebauer and C E Allen. Estimating Surface Diffusion Coefficients. *Prog. Surf. Sci.*, 49(3):265–330, July 1995.

[107] J V Barth. Transport of Adsorbates at Metal Surfaces: From Thermal Migration to Hot Precursors. *Surf. Sci. Rep.*, 40(3):75–149, December 1999.

[108] J C Kuhr and H J Fitting. Monte Carlo Simulation of Electron Emission from Solids. *J. Electron Spectrosc. Relat. Phenom.*, 105(2-3):257–273, January 1999.

[109] M Bresin, M Toth, and K A Dunn. Direct-Write 3D Nanolithography at Cryogenic Temperatures. *Nanotechnology*, 24(3):35301, January 2013.

[110] M Bresin, B L Thiel, M Toth, and K A Dunn. Focused Electron Beam-Induced Deposition at Cryogenic Temperatures. *J. Mater. Res.*, 26(3):357–364, 2011.

[111] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.