Simulating Radial Dendrite Growth

by

Ryan Foss

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2016 by the
Graduate Supervisory Committee:

Michael Kozicki, Chair
Hugh Barnaby
David Allee

ARIZONA STATE UNIVERSITY

December 2016

ABSTRACT

The formation of dendrites in materials is usually seen as a failure-inducing defect in devices. Naturally, most research views dendrites as a problem needing a solution while focusing on process control techniques and post-mortem analysis of various stress patterns with the ultimate goal of total suppression of the structures. However, programmable metallization cell (PMC) technology embraces dendrite formation in chalcogenide glasses by utilizing the nascent conductive filaments as its core operative element. Furthermore, exciting More-than-Moore capabilities in the realms of device watermarking and hardware encryption schema are made possible by the random nature of dendritic branch growth. While dendritic structures have been observed and are well-documented in solid state materials, there is still no satisfactory theoretical model that can provide insight and a better understanding of how dendrites form. Ultimately, what is desired is the capability to predict the final structure of the conductive filament in a PMC device so that exciting new applications can be developed with PMC technology.

This thesis details the results of an effort to create a first-principles MATLAB simulation model that uses configurable physical parameters to generate images of dendritic structures. Generated images are compared against real-world samples. While growth has a significant random component, there are several reliable characteristics that form under similar parameter sets that can be monitored such as the relative length of major dendrite arms, common branching angles, and overall growth directionality.

The first simulation model that was constructed takes a Newtonian perspective of the problem and is implemented using the Euler numerical method. This model has several shortcomings stemming majorly from the simplistic treatment of the problem, but is highly performant. The model is then revised to use the Verlet numerical method, which increases the simulation accuracy, but still does not fully resolve the issues with the theoretical background. The final simulation model returns to the Euler method, but is a stochastic model based on Mott-Gurney’s ion hopping theory applied to solids. The results from this model are seen to match real samples the closest of all simulations.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

In these waning days of Moore's law [1], many foundational semiconductor technologies face uncertain futures. Nowhere is this more visible than in the realm of memory cell technology. Floating gate Flash memory and derivatives have served as faithful workhorses in the semiconductor industry for more than 25 years [2], exhibiting commendable reliability in both application usage and rate of technological growth. However, with traditional device scaling becoming increasingly more difficult to obtain, entirely new device technologies have appeared to supplant Flash [3]. These new technologies fight a tough battle in the marketplace because not only must they scale at least as well as current Flash, but they must also provide some other interesting characteristic that sets them apart from other solutions. For example, where the world of memory design is currently split between applications requiring high speed volatile memory, and low speed non-volatile memory, a next generation memory technology could bridge that divide and offer a universal solution. This direction of thinking is the ethos of the More than Moore philosophy: economic gains by value-added features [4].

Of particular interest is programmable metallization cell (PMC) technology. Like other hopeful contenders, PMC devices have been proven to be viable memory elements [5]. But PMC differs from other solutions in the extent of More than Moore applications available, with the same cell architecture offering very diverse and valuable opportunities. This wealth of possibilities comes from the unique operating principle that is the namesake of PMC: the formation (and dissolution) of a thin metallic bridge, or conductive filament, between an anode and cathode. Close inspection of this filament reveals a complicated structure with dendritic branches exhibiting fractal patterning, as can be seen in Figure 1.1. Due to the fundamentally random properties inherent in dendritic growth, device watermarking or even hardware encryption schema can be developed which can be manufactured in a production setting using the same processing steps as a bulk memory panel – with obvious economic benefits.

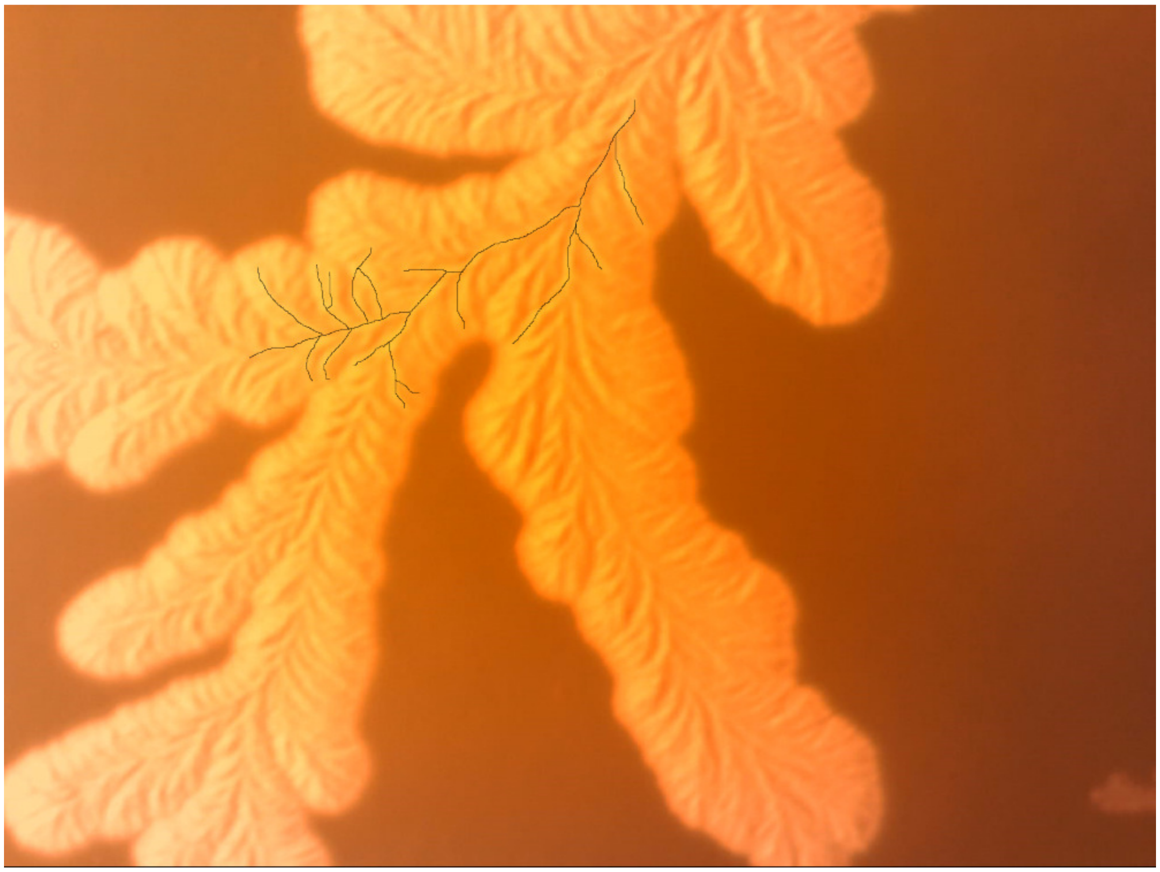


Figure 1.1: Magnified view of a dendrite sample showing fractal patterning. A small extent of the branches inside a major arm is traced in black to highlight the similar bifurcation occurring at smaller scales.

The biggest hurdle yet to overcome in realizing the More than Moore capabilities inherent to PMC technology is not the manufacturing of the device itself. This, fortunately, is a problem that has been solved [5]. The remaining challenge lies in understanding and reliably influencing the random structure of the conductive filament that forms in a stressed cell. With this capability, a manufacturer could uniquely generate a dendritic structure that is easy to verify as conforming to certain parameters, but very difficult to forge.

1.2 PMC Operation

A prototypical device is a circular electrolyte, typically a solid chalcogenide, with an anodic outer ring and cathodic center point. When a positive bias is applied between the anode and cathode, metal atoms at the anode oxidize and the resulting ions become mobile inside the electrolytic medium. Ions supplied from the anode reduce at the cathode after following the electric field and radially branching dendritic structures form as more ions enter the system. These dendrites eventually grow all the way back to the anode and create a low resistance conductive path. The redox reactions with a generic metal M can be simply written as

$$M \rightarrow M^{+} + e^{-} \tag{1.1}$$

$$M^{+} + e^{-} \rightarrow M \tag{1.2}$$

The dendritic filaments persist when the bias is removed, which gives PMC its non-volatility that is crucial as a long-term memory element. This process is reversible by applying a negative bias across the same terminals, and the dendrites will dissolve back into the anode. Devices are generally envisioned as 2D, but do possess a small third dimensional extent when constructed. This does not change the fundamentals of operation and small amounts of growth will occur on the dendrite arms along the vertical direction, but this can be considered negligible with regards to the growth in the lateral direction so long as the device is significantly more planar than cylindrical. Because devices in this thesis are considered to be much wider than they are tall, 3D growth is disregarded.

1.3 Thesis Outline

This thesis will detail the creation of a MATLAB-based computer model that attempts to provide a realistic simulation of dendrite formation at the atomic scale in a prototypical PMC device. Simulation output is compared against real samples. Chapter 2 begins the discussion with the theoretical mechanics behind dendrite formation. Primary focus is on the kinematic theory of Mott-Gurney [6], but attention is also given to a dynamics viewpoint as results from this perspective serve as a valuable foil to the favored model. Chapter 3 explores several different implementations of the mechanics in the simulation environment and also includes conceptualizations of the physical scene. Chapter 4 pits the simulation against reality and compares the results. Chapter 5 concludes this thesis with a summary of results and final remarks.

CHAPTER 2

THEORY

The key to simulating the formation of the dendritic structure in a PMC device is to understand how the individual ions behave inside the active medium. A continuum-based model is inadequate because the expectation is that the randomized branching structure organically arises from the behavior of individual ions when the device is subjected to particular stimulus, and that a continuum approach will miss this subtlety. More specifically then, ion traversal through a chalcogenide medium is a particle transport problem with sensitivity to drift, from the applied bias across the cell, and to diffusion, from the concentration gradient created by atoms oxidizing at the anode and reducing at the cathode or a dendritic structure connected to the cathode. If, at any time, an ion comes in contact with a dendrite branch during its journey, there is a chance that the ion will bind to the structure. These few conjectures serve as a blueprint for describing the life of an ion during the simulation, and much insight can be gained through experimentation and observation to corroborate these assumptions.

Two different theoretical approaches are evaluated for describing ionic motion following two of the pillars of mechanics: kinematics and dynamics. The kinematics approach makes use of Mott-Gurney's ion hopping theory applied to solids and offers a heuristic for envisioning particle interactions that serves as support for what is referred to as the stochastic simulation model. The dynamics approach draws from Newtonian mechanics, realizing that the particles here are large enough to disregard quantum effects and slow enough to disregard relativistic effects, and supports a pair of models that more closely follow classical mechanics.

2.1 Mott-Gurney Approach

The motion of a migrating ion is driven by a combination of drift and diffusion. In the stochastic model, diffusion is responsible for providing the simulation with a large part of the randomness that manifests in the dendritic branches. In order to have atomic scale resolution, simulated device sizes are in the nanoscale regime and at this size the effects of thermal motion are readily visible. Thermal motion is commonly described as the seemingly random motion of atoms in a medium stemming from interactions between many other atoms and the quantum uncertainties in position of their constituent electrons. A random walk model is adopted to describe the ionic motion attributed to diffusion, whereby a new direction of travel is selected at every simulation time step from a uniform random distribution. Meanwhile, the magnitude of the distance the ion will travel per time step is discretized to certain values corresponding to the intrinsic bond lengths of a molecule composed of the ion of interest, which in this case is Ag+ [7]. Diffusion is also characterized in the ion replenishment mechanic whereby the reduction of an ion at a cathode-connected point spurs the oxidation of a new ion at the anode along the same direction that the reduction occurred. If there is an applied bias across the cell, this creates an electric field which causes the ions to be influenced by drift effects as well. Drift is realized in the model as a modification to the uniform distribution from which the diffusion direction is selected, such that the constructed probability mass function is augmented along the direction of the electric field and mitigated in the opposite direction. The end result of these effects is a many-participant, biased random walk.

Silver ions in chalcogenides are known to traverse their medium via both Schottky and Frenkel defects [8]. This allows for essentially a full range of movement along the plane of motion. From the perspective of an ion, the chalcogenide medium is seen as a lattice of potential wells that can each hold a single ion at a time. Ions are loosely-bound to their wells and an applied bias or an abundance of thermal energy can be sufficient impetus for migration to occur [9]. In the potential well picture, an electric field, *E*, preferentially lowers a portion of the well's barrier which increases the odds that the ion will escape in that direction and into an adjacent well. In a similar manner, increasing temperature raises the resting level of the ion inside the well which effectively lowers the barrier height overall. Figure 2.1 illustrates these effects.

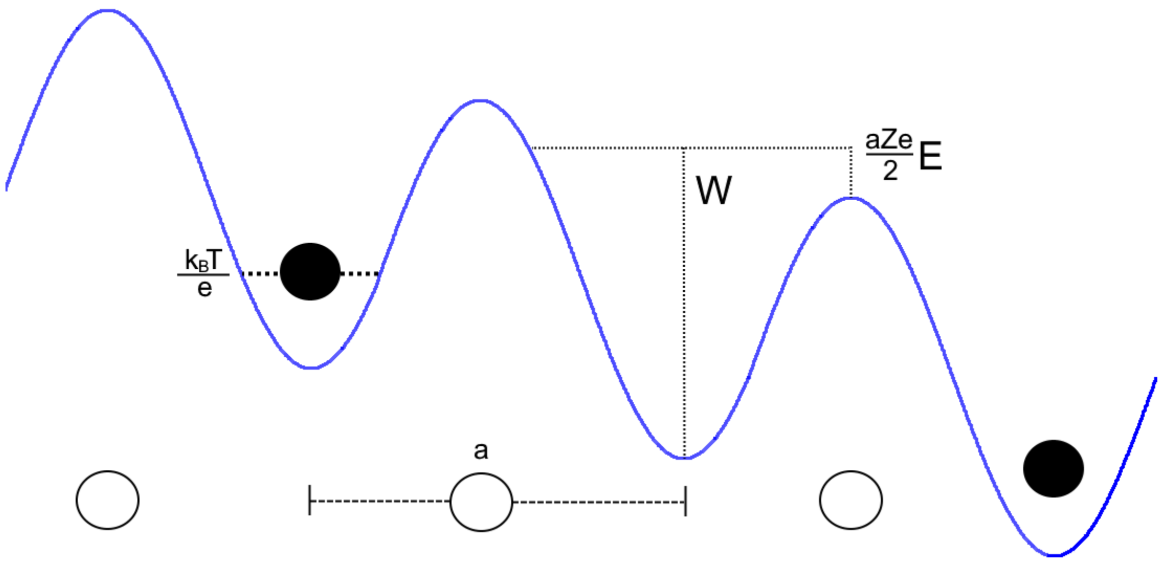


Figure 2.1: 2D sketch of ion potential wells with barrier height modification effects. Ions, in black, rest in energetic potential wells that exist in the lattice between bulk atoms in the medium. Ions may randomly move to adjacent wells, generally in the direction where the migration barrier is the lowest.

The basis for this view comes from Mott-Gurney's theory of ion hopping with a discrete model equation for ionic current density in a lattice developed by Fromhold and Cook [10]. Briefly repeating the derivation, consider the 2D potential well picture of Figure 2.1. There exist two current densities, $j_{forward}$ and $j_{reverse}$, which manifest from ions travelling with and against the electric field, respectively. The sum of these currents creates the net current density, j_{hop},

$$j_{hop} = j_{forward} - j_{reverse} \tag{2.1}$$

The forward and reverse currents can be further described with the aid of Boltzmann statistics and by defining several parameters. Let *c* be the ion concentration in the direction perpendicular to each well (assumed to be constant in this application), *f* be the ionic hopping attempt frequency, *a* be the mean ionic hopping distance, *Ze* be the effective ionic charge (which may not equal simply the elementary charge *e*, see Fromhold and Cook for further discussion), and *W* be the energetic migration barrier. This results in the following definitions

$$j_{forward,reverse} = aZecf exp\left(-\frac{W}{k_B T}\right) \tag{2.2}$$

Clearly, this would mean that by following Equation 2.1 there is no net current flow in an arbitrary cross-section of the device, but this is exactly what would be expected for the no-applied-bias setup in which the ionic motion is purely diffusive. By adding a bias across the device, *W* is asymmetrically modified by an amount equal to

$$\Delta W = \pm \frac{aZe}{2} E \tag{2.3}$$

where the two results are applied diametrically to the barrier profile, with the negative result applied to direction following the electric field. This equation makes use of the previously mentioned electric field, *E,* resulting from the applied bias. The forward and reverse currents can now be modified to incorporate field effects,

$$j_{forward} = aZecf exp\left(-\frac{W - \frac{aZe}{2} E}{k_B T}\right)$$

$$j_{reverse} = aZecfexp\left(-\frac{W + \frac{aZe}{2}E}{k_B T}\right) \tag{2.4}$$

and the final net current density is then,

$$j_{hop} = aZecf \exp\left(-\frac{W}{k_B T}\right)\left[\exp\left(\frac{aZe}{2k_B T}E\right) - \exp\left(-\frac{aZe}{2k_B T}E\right)\right] \tag{2.5a}$$

which may also be written as,

$$j_{hop} = aZecf \exp\left(-\frac{W}{k_B T}\right)\sinh\left(\frac{aZe}{2k_B T}E\right) \tag{2.5b}$$

This model as presented by Fromhold and Cook is a discrete model that is very applicable to the present situation, but a continuum model is also presented which Dignam [11] refines as he describes a phenomenological continuum transport equation that reconciles the high field results of both approaches. Dignam's reconciliation of the two types of models is encouraging as it provides support to the use of either of the approaches in a practical setting.

The leading exponential term in Equation 2.5b provides a positive temperature coefficient for ion hopping, while the hyperbolic sin term can be considered an exponentially increasing enhancement factor for large fields. Even at small fields, the term increases linearly which suggests that modulating the applied bias should always cause a noticeable change in dendritic structure. The hopping frequency, *f*, and barrier height, *W*, are both material parameters and are of particular interest in this theory because they significantly contribute to the stochastic nature of the simulation model due to their influence on ionic movement.

Hopping frequency directly relates to the rate of traversal of an ion, but becomes even more impactful when considering ion replenishment. Older ions (in terms of global simulation time) in the medium tend to be more exploratory as diffusion is dominant and progress towards the cell cathode is slow and meandering. These ions seed the beginning of the dendrite branch growth radially around the cell cathode. Younger ions are more significantly influenced by drift and tend to be more reactionary as they are captured by growing dendrite branches. The ionic hopping frequency parameter affects this trend by modifying how frequently an ion attempts to move versus how long it remains in place while surrounded by the activity of other ions.

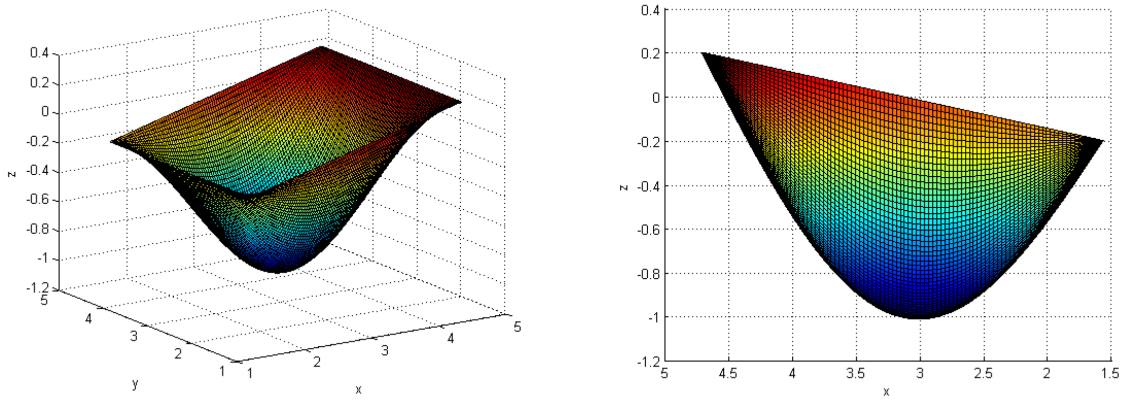


Figure 2.2: 3D visualization of a modified potential well profile. In this scenario, the barrier height is 1.0 eV with the z-axis zeroed to the unmodified barrier height. An applied bias in the -x direction has resulted in a barrier lowering of 0.2 eV, with a corresponding barrier rising of 0.2 eV in the +x direction.

While the derivation above focused on a series of 2D potential wells, it is important to remember that the wells are truly three dimensional. To capture this aspect, the 2D theory is applied as a cross sectional analysis of the complete well and rotated radially to define the entire well profile. The significant factor here is the barrier height which directly influences the direction an ion travels as it escapes its well and hops to an adjacent well. A lower barrier height in a direction means that ions can diffuse easier and are more likely to continue along that path. Field-assisted barrier modification from drift effects is applied with respect to the barrier height taken at the unbiased state, and is modified according to Equation 2.3. In a device, a mobile ion will be influenced by the many atoms that compose a dendritic structure, both near and far. This results in an overall radial barrier modification with clear directionality but also a cosine-patterned transition between the minimum and maximum height, as can be seen in Figure 2.2. The augmented barrier has no energetic upper limit, but 0eV is considered to be the lowest a barrier can be lowered. If no barrier height is defined in any direction, the particle's motion is unfettered and purely diffusive. While the unmodified barrier height value is a material parameter, the final barrier height profile is somewhat of an empirical parameter that contains energetic information about the formation and dissolution of the vacancies or other defects in the chalcogenide as the ion migrates and even physical information about the crystallinity of the medium that can appear in the radial inhomogeneity of the barrier.

2.2 Classical Approach

The classical models make use of elementary Newtonian mechanics to calculate familiar parameters of interest, such as position, velocity, acceleration, and force. Just as in the stochastic model, drift and diffusion are the motivators of ionic motion. Though even here, some elements of the Mott-Gurney theory are used as a basis for the simulation, including certain limitations such as the fact that ion velocity cannot exceed the speed of sound [12]. This limitation firmly plants the problem in the realm of Newton, and far away from any relativistic concerns as typical speeds sound in chalcogenide materials range in the few thousands of meters per second [13].

Drift in the classical model follows from the force felt by a charged particle as described by Coulomb's Law,

$$F = k_e \frac{|q_1 q_2|}{r^2} \tag{2.6}$$

where k_e is Coulomb's constant and r is the distance between charges q_1 and q_2. In application, every mobile ion is evaluated as q_1 with a pairwise interaction calculated for every bound ion, q_2, which is part of a forming dendrite branch. This is the most computationally intensive part of the classical model because one aggregate force vector must be derived from hundreds or thousands of component sources. Diffusion once again takes the form of a random walk and is realized by a force vector with a constant magnitude but a direction drawn randomly from a uniform distribution. The final drift force vector is linearly combined with the diffusion force vector to create a single net force that spurs ion movement.

Space in the classical model is discretized to emulate the existence of ion hopping sites from Mott-Gurney and to provide the positional exclusivity necessary for ion structures to form. At every simulation time step, the projected position of an ion is snapped into the nearest available site. This does raise a small concern for conservation of energy in that an ion that lands anywhere but the very center of a hopping site will experience an anomalous force that adjusts the ion's position in the grid. Intuition leads that the number of these pushes and pulls should approximately cancel out by the time the ion comes to rest, but the concern is still valid.

CHAPTER 3

IMPLEMENTATION

Defining the background theory for the simulation model is only half of the solution. The issue of proper implementation is also an important matter that sometimes warrants multiple explorations. Two different implementations of the classical approach were developed utilizing two different established numerical methods as the backbone. The implementations are named after the numerical methods for ease of identification, but note that “model” will be used to refer to the implementation while “method” will be used to refer to the numerical method itself. Section 3.1 discusses the Euler model, which despite some short-comings, serves as a fast, simple, and generic look at dendrite formation. Section 3.2 discusses the Verlet model, which addresses some of the issues with the Euler method in the classical approach setting and allows for more customization. The implementation of the Mott-Gurney approach is discussed in Section 3.3 with the stochastic model and is considered the preferred model overall. The most customization of any model is available here and the simulation output is qualitatively the most similar to real samples. All simulations were fully developed in MATLAB.

3.1 Euler Model

Choosing a method of numerical analysis depends greatly on the form of the problem. From a classical mechanics view, dendritic growth can be envisioned as an ordinary differential equation through the mechanics of ion transport: an ion has an initial position given at time zero, and a new position is calculated at every time step. In aggregate, the procession of ions through time directly corresponds to the rate of dendritic growth. Runge-Kutta methods are well-known tools for evaluating discrete-time differential equations such as this one, with many variations developed to focus on elements such as error mitigation and stability. The simplest of these methods is the explicit Euler method.

The Euler method is used to solve problems in the form of,

$$p'(t) = v(t) \tag{3.1}$$

$$p(t_0) = p_0 \tag{3.2}$$

where we let p represent a specific particle's position inside the active area and t represent the time since the simulation's start. $p'(t)$ is then clearly the velocity of the particle, $v(t)$. This method is iterative in that,

$$t_{n+1} = t_n + \Delta t \tag{3.3}$$

$$p_{n+1} = p_n + v_n \Delta t \tag{3.4}$$

where Δt is the step size which here represents the advancement of time, and is used to find successive particle positions [14]. Initial position p_0 is determined one of two ways depending on when the particle is spawned. If $t = 0$, then $t_0 = 0$ and the particle is placed somewhere inside the active area to realize a uniform distribution with respect to other particles spawned at $t = 0$. At all $t > 0$, $t_0 = t$ and p_0 is sampled from a uniform distribution of available positions along the anodic ring of the active area. Because the metal ions in a PMC device traverse the chalcogenide through site hopping, the Euler model visualizes the active region as a Cartesian grid of single-occupancy potential wells. Available positions for a spawned particle then are the grid locations $s(x,y)$ that do not contain a particle at time t. It is important to note that representing the radial geometry of the device with a rectilinear grid was seen to cause a conspicuous cross-like structure to form consistently under high field simulations, as can be seen in Figure 3.1b. The root cause of this behavior is the limited number of potential binding angles that a mobile ion can attach to on the dendrite structure when dealing with a Cartesian grid. Section 3.3 discusses changing to a polar coordinate system to address this issue.

As a discretized representation of a continuous time problem, there is unavoidably an amount of error introduced into the result at every simulation step. While it is difficult, even impossible, to examine a simulation output and determine where inherent simulation error caused a deviation from reality, quantified error is still a useful metric to use when comparing similar simulation methods. To determine the amount of error, we begin by first finding the exact solution for the particle's position at time $t = t_0 + \Delta t$. This can be done by taking the Taylor expansion of $p(t_0 + \Delta t)$ about $p(t_0)$,

$$p(t_0 + \Delta t) = p(t_0) + p'(t_0)\Delta t + \frac{1}{2}p''(t_0)\Delta t^2 + O(\Delta t^3) \tag{3.5}$$

Subtracting the exact solution in Equation 3.5 from the Euler approximation in Equation 3.4, we get the local truncation error,

$$p(t_0 + \Delta t) - p_1 = \frac{1}{2}p''(t_0)\Delta t^2 + O(\Delta t^3) \tag{3.6}$$

which shows that there is an accumulation of error proportional to Δt^2 committed by each step when Δt is small. The global truncation error at the end of the simulation is the product of the local truncation error and the total number of steps in the simulation. With the total number of simulation steps inversely proportional to the step size Δt, we realize the final result will have an error proportional to Δt, making the Euler method a first order method [14]. In other words, the Euler method benefits from increasing the resolution of the simulation, but the benefits scale linearly. There exist other, more computationally intensive methods that generate error at a higher order where the cost of the increased sim time is balanced by greater benefits in sim precision. Some examples include extending the explicit Euler method to the Adams-Bashforth method by calculating two steps in the future using the current step and the following step, the midpoint method which estimates results at half time step increments for the first derivate calculation, and the Verlet method which is discussed in Section 3.2.

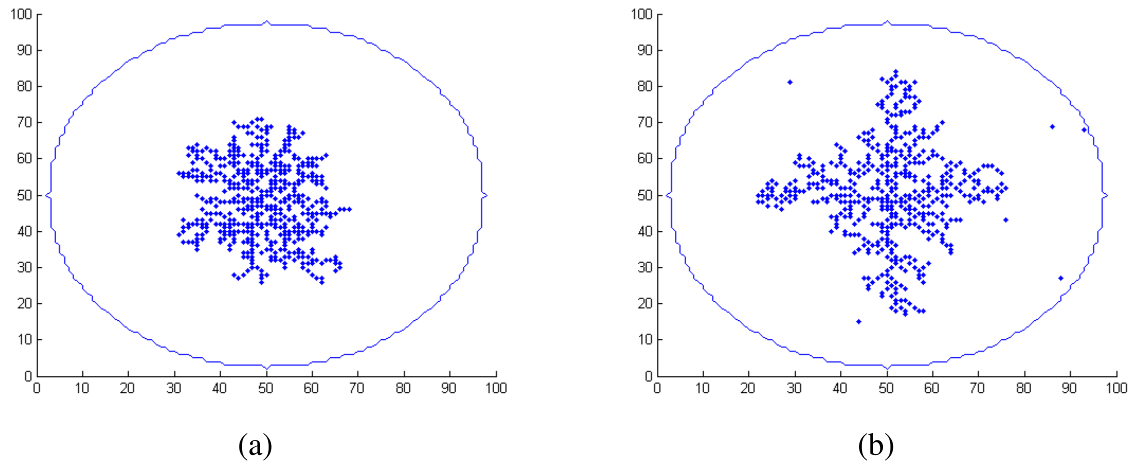


Figure 3.1: Sample images of Euler model results for 500 ions in a cell of 50nm radius. (a) shows the results for a "low field" sim while (b) shows the results for a "high field" sim. Note that (b) also shows several mobile ions that are not yet captured by the dendritic structure.

Configurability of the Euler model simulation is limited to geometry changes with regards to the radius of the PMC device, the number and distribution of ions inside the active area at initialization, and the strength of the field across the cell for matters of drift. Ion replenishment is also an option, which may be enabled or disabled. Overall, this is the simplest implementation because there is no material or physical parameter available to tune towards a real world setup.

Ions that are not bound to a dendrite are mobile and must have their position updated at every time step. At the beginning of each iteration of the main simulation loop, the previous loop iteration's velocity is used to determine the next position for the ion, and the ion is snapped into the nearest free grid location to that position. Then, a check is made to see if the ion is touching a dendrite at any of the eight adjacent sites. If it is, the ion is captured and bound to the structure. A bound ion, which is now a neutralized atom, no longer iterates its position and can immediately capture other mobile ions if they land in an adjacent site. If the ion is not touching a dendrite, then the next velocity vector direction is calculated by weighing together the random direction originating from diffusion and the field-following direction originating from drift. To calculate drift, a virtual charged particle is placed a distance away from the ion under evaluation that represents an amalgamation of the field effects felt from the entire dendritic structure. The weight between drift and diffusion is assigned such that far away from the virtual particle, drift has negligible effect, but has a quadratically increasing and eventually dominating effect as the ion approaches. The magnitude of the ion's velocity vector, the speed, is held constant, while the direction of the velocity vector changes at every time step in response to drift and diffusion effects. In the context of this simulation, a constant speed corresponds to ions hopping between sites at a constant rate. The simulation is terminated successfully when all mobile ions are captured by the dendritic structure and become bound.

The largest disadvantage of using the explicit Euler method for a classical physics simulation comes from its simplicity in implementation. While Equation 3.4 is sensitive to velocity, the Euler method does not calculate acceleration. This lack of concern for acceleration is troublesome because the background theory that supports this model is force-based, but the Euler method is position-focused. This results in ions essentially being shuttled around the site grid by an “invisible hand” of sorts, and without proper acceleration data to satisfy the theoretical background, the final result is just an approximation of physical phenomena.

3.2 Verlet Model

The Verlet model is a more robust implementation of the classical mechanics theory that is not significantly more computationally expensive than the Euler model, calculates force terms which can be used in acceleration calculations, and experiences less error as a second order method. Unlike the Euler method, the Verlet method is not a Runge-Kutta method. As such, the Verlet method is used to solve differential equations of the form,

$$p''(t) = a(t) \tag{3.8}$$

$$p(t_0) = p_0 \tag{3.9}$$

$$p'(t_0) = v_0 \tag{3.10}$$

where $p''(t)$ is the particle’s acceleration, $a(t)$, and all other definitions follow from the Euler method. Iterations of the Verlet method advance time as in Equation 3.3, with position and velocity terms calculated by

$$p_{n+1} = p_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2 \tag{3.11}$$

$$v_{n+1} = v_n + \frac{a_n + a_{n+1}}{2} \Delta t \tag{3.12}$$

Acceleration at each iteration is derived from Newton’s second law by dividing the force applied to the ion by the ion’s mass. Once again, the net force on the ion is a combination of drift (via Coulomb's Law, Equation 2.6) and diffusion. Computational complexity is increased compared to the Euler method, but calculations are still made using only the current time step data which is a performance benefit compared against other possible methods.

Error in the Verlet method can be calculated in a similar manner to the Euler method. First, we begin again with the Taylor expansion of $p(t_0 + \Delta t)$ about $p(t_0)$, but then we also look at the expansion of $p(t_0 - \Delta t)$ about $p(t_0)$ due to the Verlet method's time symmetry,

$$p(t_0 + \Delta t) = p(t_0) + p'(t_0)\Delta t + \frac{1}{2}p''(t_0)\Delta t^2 + \frac{1}{6}p'''(t_0)\Delta t^3 + O(\Delta t^4) \quad (3.13)$$

$$p(t_0 - \Delta t) = p(t_0) - p'(t_0)\Delta t + \frac{1}{2}p''(t_0)\Delta t^2 - \frac{1}{6}p'''(t_0)\Delta t^3 + O(\Delta t^4) \quad (3.14)$$

This time, we bring the expansion out to the third derivative of $p(t_0)$ and when Equation 3.13 and Equation 3.14 are summed together, the odd terms cancel leaving just,

$$p(t_0 + \Delta t) + p(t_0 - \Delta t) = 2p(t_0) + p''(t_0)\Delta t^2 + 2O(\Delta t^4) \quad (3.15)$$

This gives the local truncation error in position a 4th order, a very strong estimation. However, unlike the Euler method where each time step influences the global error by $1/\Delta t$, the Verlet method's higher order terms in the position calculation lead to a $1/\Delta t^2$ factor contributed by each step. This gives the Verlet method a second order global truncation error [15]. In comparison to the Euler method, this is the superior method for simulation accuracy.

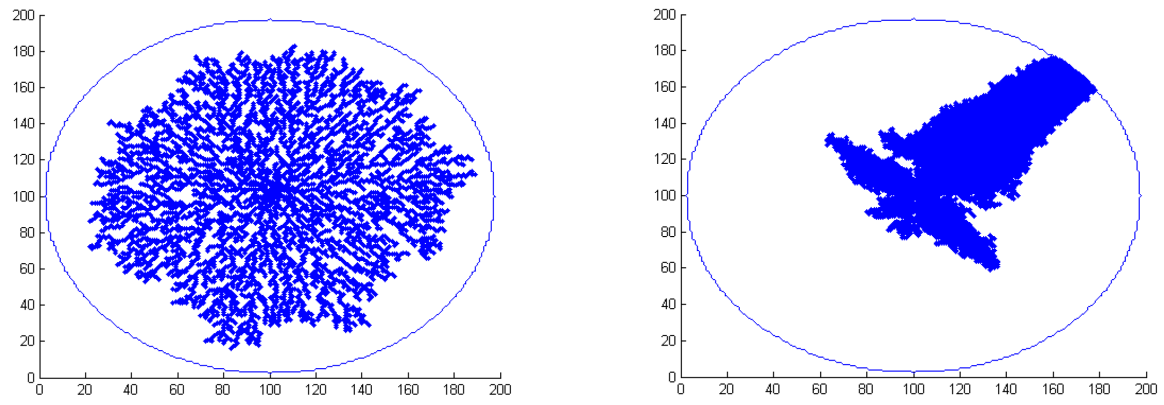


Figure 3.2: Sample images of Verlet model results for 5000 ions in a cell of 100nm radius. (a) shows the results for a “low field” sim while (b) shows the results for a “high field” sim.

The Verlet model has similar configurability to the Euler model as far as defining the size of the cell and the ion population at initialization. But, due to the fact that the calculated acceleration is derived from Newton’s law, the ion mass is a configurable parameter. Additionally, the voltage applied at the cathode must be provided in order to determine the force in Coulomb’s law. This lets the Verlet model generate much more dynamic results and also allows for tuning of the simulation to a real-world material. Figure 3.2 shows results from the Verlet model.

Much of the Verlet model’s implementation is the same as the Euler model with respect to the Cartesian grid representation of the chalcogenide medium and the rules of mobile ion capture by the forming dendrite. However, as mentioned above, this model is fundamentally force-based and instead of the Euler model’s “invisible hand”, there is a real motivator for ion motion. The force felt from drift is calculated per interacting pair of ions by Coulomb’s Law and consolidated into a single net drift force. The diffusion force is constructed so that the ion behaves in accordance with the random walk model. The magnitude of the diffusion force is constant, but the force is only large enough to propel the ion to an adjacent site in the grid. A constant force will theoretically cause the ion to travel with an increasing velocity at every time step, but the randomization of the force's direction means this effect is lessened in practice. This concern is further mitigated by the fact that the diffusion force is only significant at large distances away from sources of potential. As a mobile ion nears a cluster of bound ions, the drift force can be orders of magnitude larger than the diffusion force and dominate the net force. This overpowering nature of the drift force also provides an ion site hopping frequency analogue as the net applied force can be sufficient to propel an ion multiple sites in one time step.

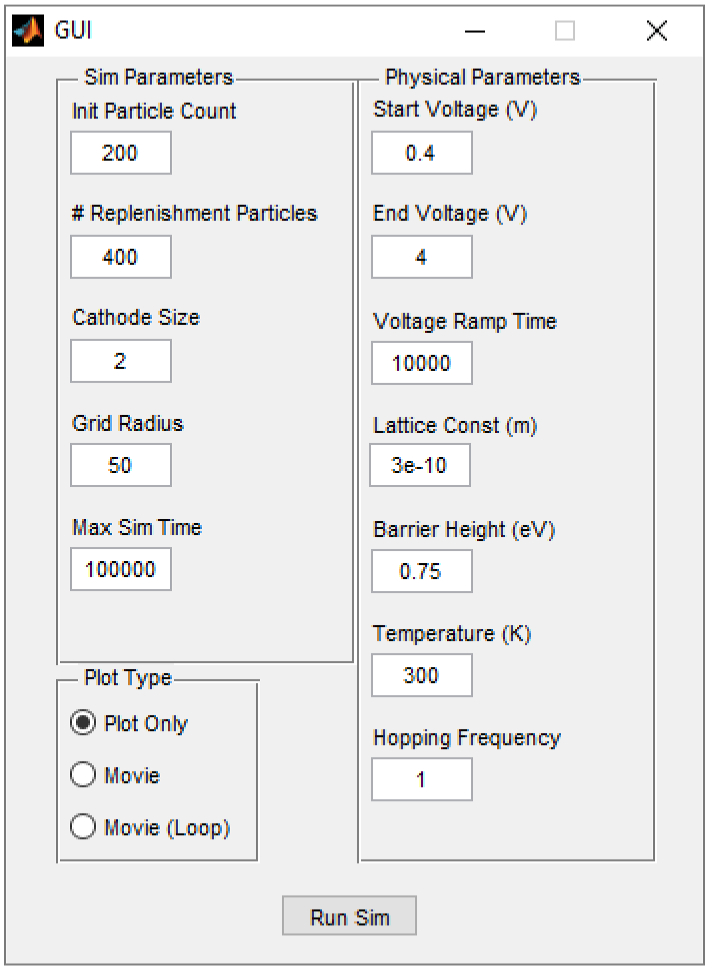
While the Verlet model is more accurate than Euler in tracking Newtonian motion, the end result is not without its quirks. As mentioned above, a force consistently applied in a direction leads to an ever-increasing velocity. As an ion nears a dendrite, the ion's velocity must be artificially clamped, lest superluminal motion be breached. The varied ion hopping distance brought on by high velocity is not quite correct as the hopping frequency in Equation 2.5b is a constant, and not dependent on proximity to sources of potential. The magnitude of the applied bias is also of concern because it is unfortunately entangled with the size of the simulation time step. Computationally reasonable time step sizes end up requiring very small applied biases on the order of nanovolts to prevent position updates that throw the ion far outside the simulation grid area, whereas real PMC devices are known to program with hundreds of millivolts. Due to these concerns, it was realized that the simulation method was not the only limiting factor to the success of a realistic simulation model. This Verlet model is the last attempt to use classical mechanics to govern the ionic motion, and further work continues with a different theoretical background.

3.3 Stochastic Model

The numerical method at the core of the stochastic model is the Euler method. Despite the challenges faced in the classical model, the Euler method is sufficiently accurate for a more kinematic model. In the classical model, the motivator of ionic motion is the interaction of large applied forces that are recalculated at every time step and the various ion speeds that result are implicitly taken to represent something akin to the ion hopping frequency. The stochastic model depends instead on field-assisted diffusion, where a constant ion speed is desired and an explicit hopping frequency parameter exists. The lack of calculated acceleration terms which leads to a constant ionic speed in the earlier Euler model is embraced here by permitting ions to hop only to their nearest neighboring site by definition in the theoretical background. The computational simplicity of the Euler method is taken advantage of here by defining the nearest neighbor more fluidly in this model. The simple Cartesian grid with eight nearest neighboring sites is replaced by a polar coordinate system with potential for far greater radial symmetry for movement, as can be seen in the result plots of Figure 3.4.

The stochastic model is the most configurable simulation. Cell size and particle count at initialization are configurable, as always. Ion replenishment from the anode is optional and can be configured to cease after a specified number of ions have spawned so that different cell dynamics can be evaluated. The size of the cell’s cathode is a new geometry feature which is implemented as a collection of ions bound in a circle around the center of the cell at initialization. Applied bias is present again, but ionic mass is not.

Instead, parameters from Equation 2.5b are present such as the mean hopping distance, barrier height, hopping frequency, and temperature. The applied bias can also be linearly ramped. In the classical model, ionic mass was the best descriptor of an ion's atomic identity, but here the hopping distance and barrier height attributes together fill that role. Figure 3.3 shows the configuration GUI wrapper developed to handle user input to the simulation.

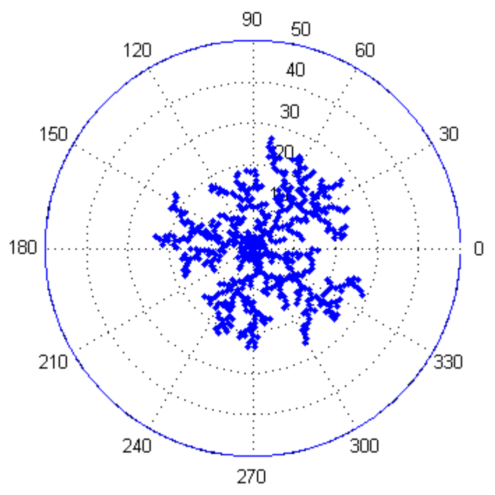

GUI

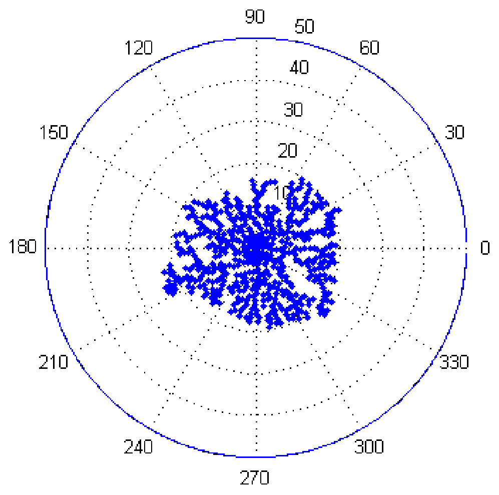
Sim Parameters
Init Particle Count: 200
Replenishment Particles: 400
Cathode Size: 2
Grid Radius: 50
Max Sim Time: 100000

Physical Parameters
Start Voltage (V): 0.4
End Voltage (V): 4
Voltage Ramp Time: 10000
Lattice Const (m): 3e-10
Barrier Height (eV): 0.75
Temperature (K): 300
Hopping Frequency: 1

Plot Type
[✓ Plot Only]
Movie
Movie (Loop)

Figure 3.3: The GUI wrapper, also developed in MATLAB, sends user data to the core simulation code for ease of use. Multiple result formats are available including static final image and movie playback.





(a) (b)

Figure 3.4: Sample images of the stochastic model results for 500 ions in a cell of 50nm radius, following the physical parameter configuration shown in Figure 3.3 (with voltage ramp disabled). (a) shows the results for a "low field" sim (0.4V applied bias) while (b) shows the results for a "high field" sim (4.0V applied bias).

Figure 3.6 shows a flowchart depicting the simulation flow from initialization to termination. The general simulation loop framework described in Section 3.1 remains the same for this model, but there are several functionally significant changes related to determining the next particle position that make this a truly distinct model. As mentioned earlier, an ion is permitted to move the distance of one well-width per simulation time step. To determine the direction of travel, a probability mass function is constructed to represent the probability that an ion will move in an arbitrary direction and the ion's heading is sampled from the PMF at every time step. Initially, the PMF reflects a uniform distribution to model purely diffusive motion, but then drift effects are used to modify elements of the PMF in a manner consistent with the potential well barrier height modification in Mott-Gurney's theory. Figure 3.5 displays the PMF generated for a single ion under the effects of a moderately high electric field.

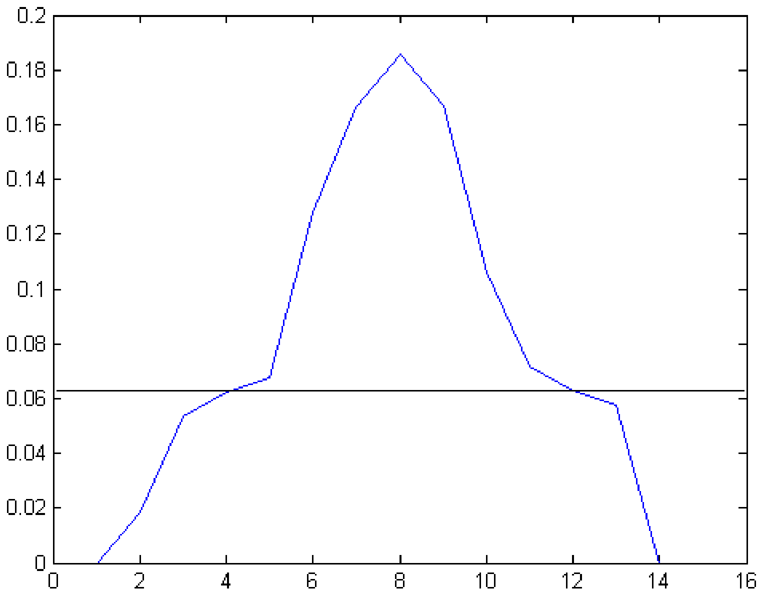


Figure 3.5: A modified PMF. The ion is given 16 evenly, radially distributed directions to choose from corresponding to 0 to 2π rad, with the solid horizontal line in the image demarking the base probabilities of the unmodified PMF. The peak at direction 8 in the center and the troughs at the periphery show that this ion is strongly inclined to move in the +pi direction. Note that directions 4 and 12, corresponding with the π/2 and 3π/2 directions respectively, intersect with the base PMF line. This shows that the ion's tendency to move laterally with respect to the electric field is unperturbed by the applied bias, as expected.

Drift is accounted for by first determining the direction from the current mobile ion to each bound atom on the dendritic structure. Then for each bound atom, a ray is traced from the bound atom through the mobile ion back to the anode. The user-provided applied bias is divided by the distance from the anode to the bound atom to calculate the instantaneous electric field felt by the ion for that time step. Equation 2.3 is used to calculate the value of the energetic barrier modification, and this value is divided by the effective barrier height, W - k_BT, to determine the percentage change in barrier height.

Finally, the PMF is modified by augmenting the probability in the direction of the electric field and reducing the probability in the opposite direction, and the ion’s direction of travel is sampled. The PMF is reset to the default uniform distribution for each mobile ion at every time step, and modified again to capture the dynamically evolving field effects inside the cell as the dendritic structure continues to expand.

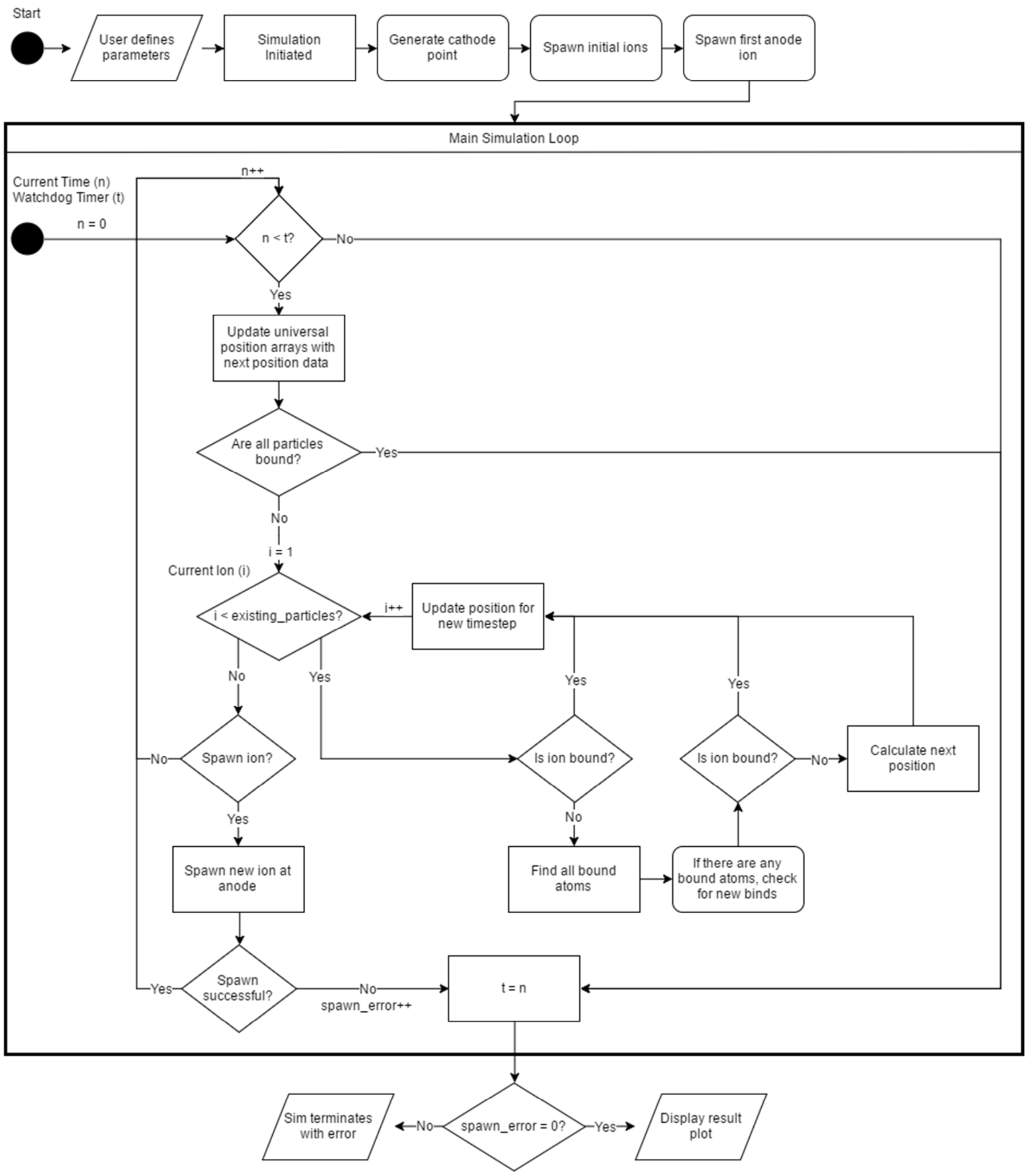


Figure 3.6: Stochastic model flowchart covering the major decision elements from initialization to termination.

CHAPTER 4

VALIDATION

Growing dendritic structures is heavily dependent on random processes. This presents something of a challenge when it comes to verifying simulation results against real-world samples because while the global concentration of ions in the sample is considered to be constant, minor fluctuations in local ion concentration can result in wildly varying branch orientation that may make two identically prepared samples appear quite different. Fortunately, there do exist some qualitative and quantitative metrics than can be used to verify simulation results from a theoretical perspective and also to directly compare result images.

Branch growth angle is a quantitative metric that verifies simulation results and is partially based off of the crystallographic growth angles of the metal's atomic species. For Ag+ ions, branch angles around 38° should be well-represented, while angles at approximately 45° and 65° should also be present [16]. It is expected that major branches will occur at angles similar to these. Qualitatively speaking, the magnitude of the applied bias across the cell should noticeably affect the branch density. Initial expectations from planar device experiments suggested that high biases would lead to elongated dendrites with minimal branching while low biases would lead to a more uniform buildup of metal at the cathode. However, the stochastic simulation shows the opposite behavior with a circular device geometry. Therefore, the expectation for this series of experiments is that under low electric fields the dendritic structure is more sparse, while under high fields the dendritic structure is much more condensed and intricate.

There are two ways to attempt to quantify the density of a dendritic structure. One is to count the number of dendrite arms of a certain length where major bifurcations occur and perform a relative comparison. The other is to calculate the fractal dimensionality factor of an entire structure or a subset thereof. ImageJ is a software program developed by the U.S. National Institutes of Health that can be used to perform fractal dimension analysis. In the following sections, simulation results from the stochastic model are analyzed and compared against dendrite samples grown on a SiO_2 substrate.

4.1 Low Field

The first comparison and discussion will evaluate a so-called “low field” experiment. The physical sample is a 1cm dendritic structure that was grown for 4000s under an applied bias of 10V, shown in Figure 4.1. In the interest of time and computational resources available, simulated samples are grown under a comparatively much higher field and at a much smaller scale, which also results in a much shorter simulated growth time on the order of nanoseconds. It is important to note that the simulations here are not meant to be taken as simulations of the physical sample specifically, but rather they are simulations of a similar device and the differences in dendrite morphology that result from modifying the applied bias is the major point of concern. A pair of simulation results differing in ion count and device radius, shown in Figure 4.2, are presented in order to show that the discussed comparison metrics are consistent for similar simulation setups and not the result of one fluke simulation run.

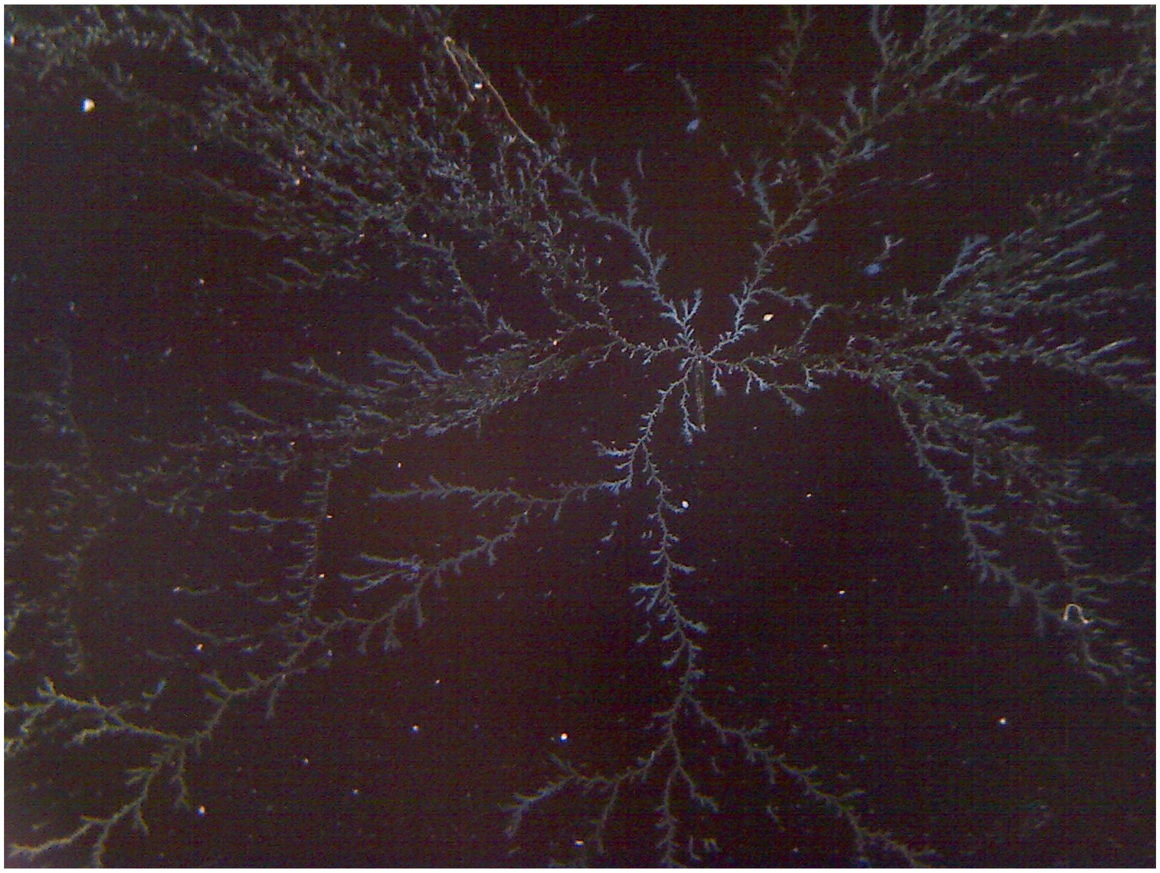
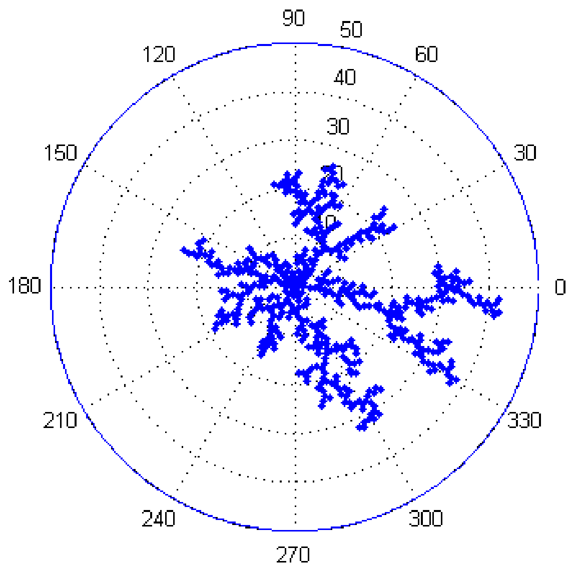
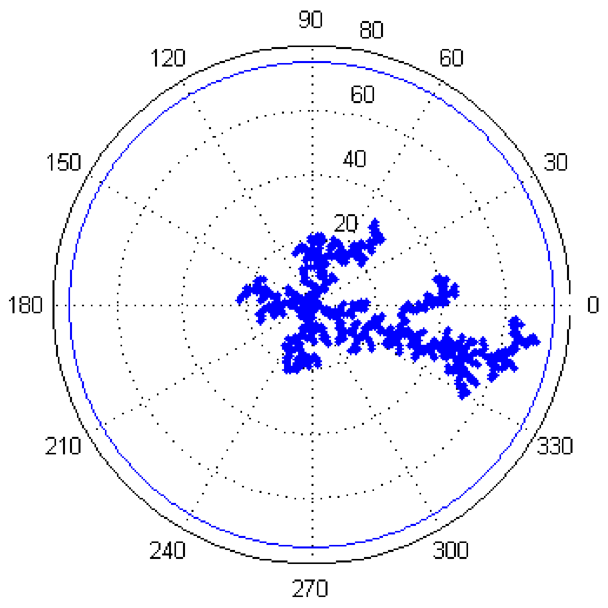


Figure 4.1: "Low field" physical sample at minimum magnification. Approximately 1cm wide, grown for 4000s with an applied bias of 10V.



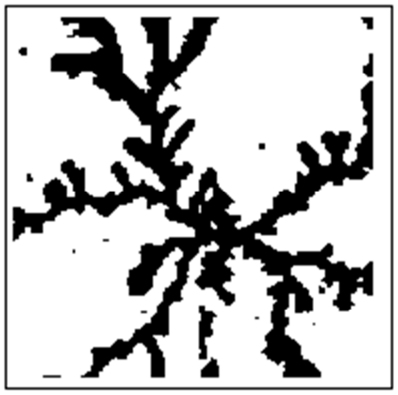
(a)



(b)

Figure 4.2: “Low field” simulated samples. (a) Radius 50nm, 500 ions, grown for several nanoseconds with an applied bias of 1V. (b) Radius 75nm, 600 ions, grown under the same conditions.

Fractal dimension is a numerical way to describe how complicated a shape is by dividing the log of the number of self-similar components in an image by the log of the magnification factor required to isolate a single one of the components. A straight line has a fractal dimension of 1 while a perfectly flat plane has a fractal dimension of 2. Any line that is not perfectly straight will then have a non-integer fractal dimension somewhere between 1 and 2. In the case of analyzing dendrites, this is a useful metric because a dense morphology will have a larger fractal dimension, closer to 2, than a sparse morphology. As mentioned in the introduction to this section, the ImageJ analysis tool provides the functionality to easily calculate the fractal dimension of any binary colored image. Figure 4.3 and Figure 4.4 show the processed dendrite images and their calculated fractal dimensions.



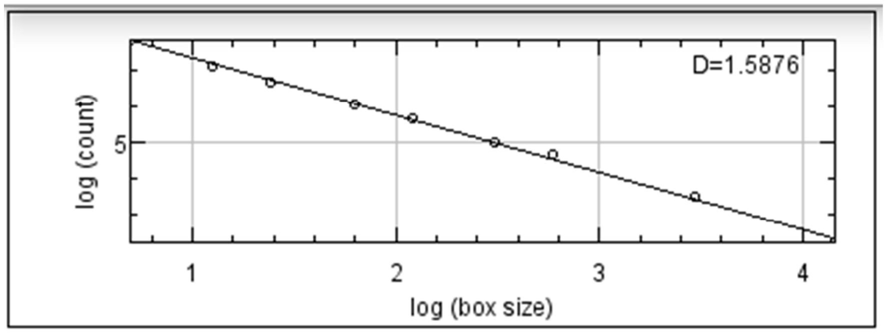
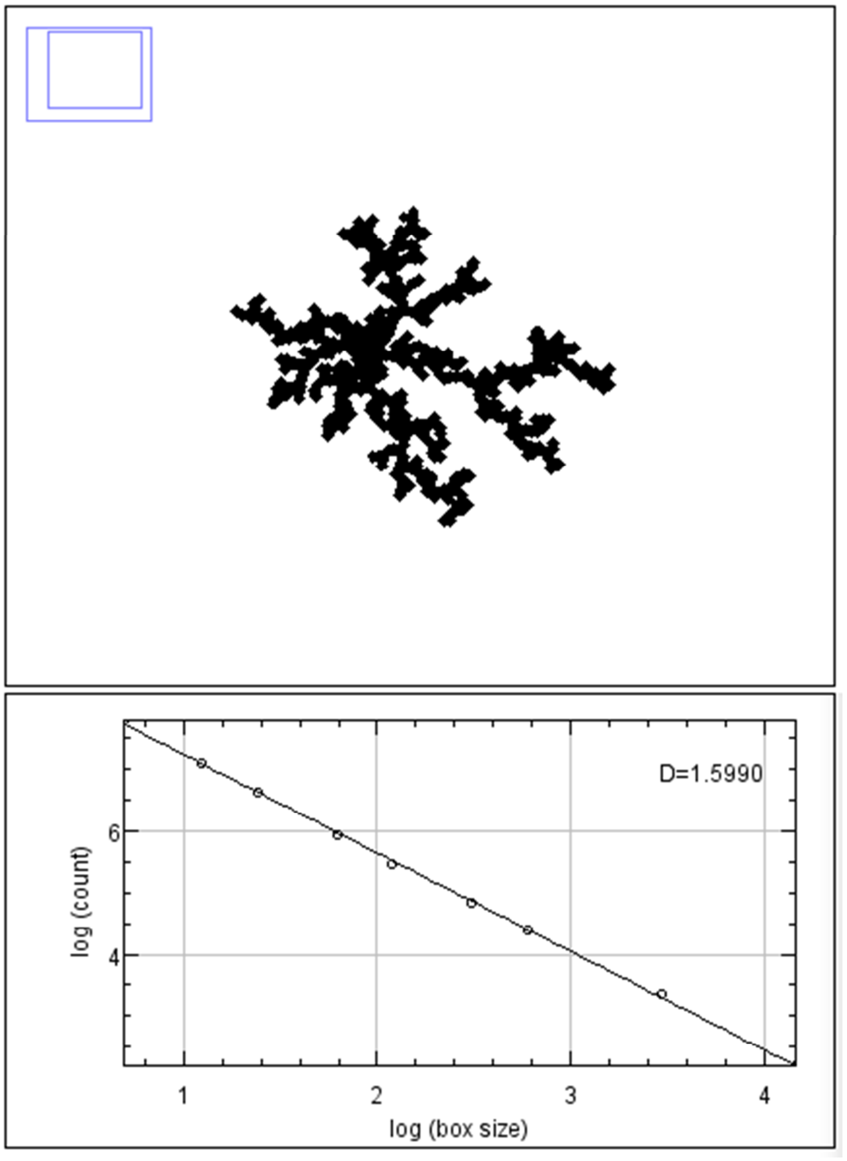
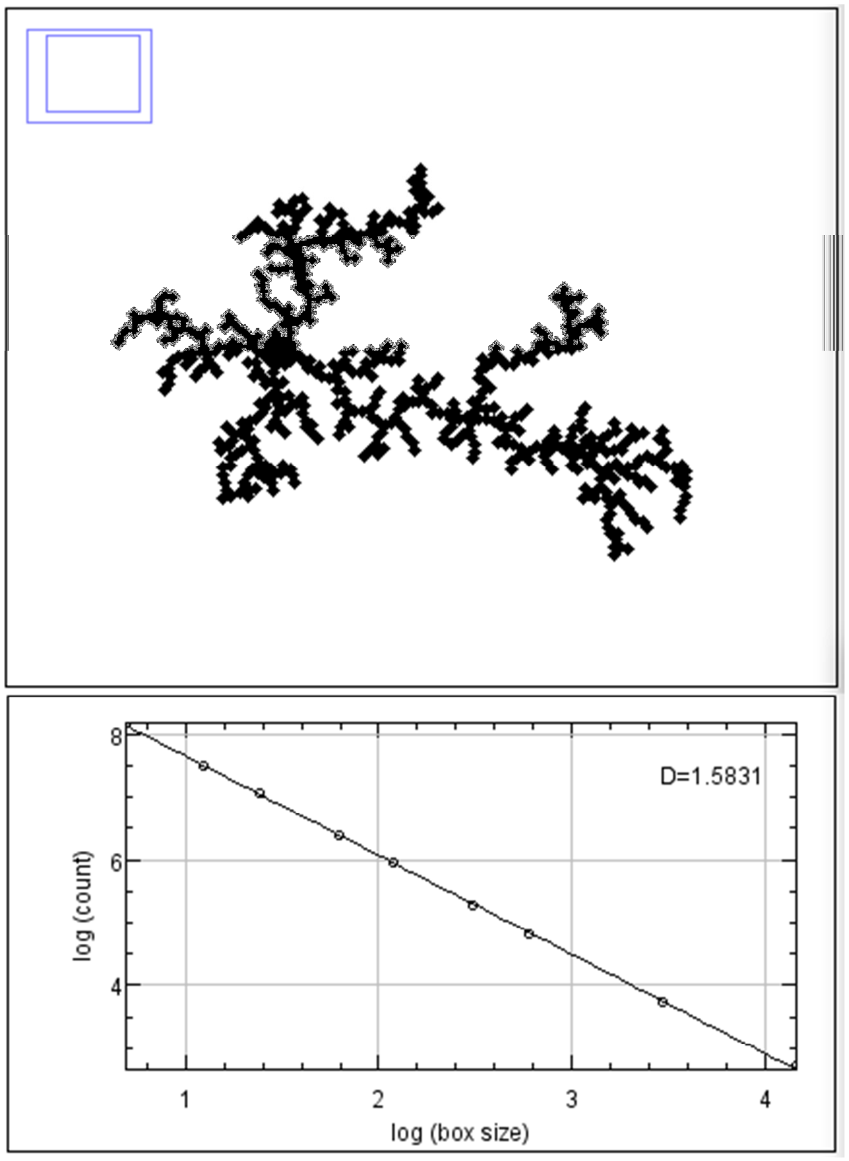


Figure 4.3: Fractal dimension analysis of the “low field” physical sample. Image has been cropped to focus on the region near the cathode point.



(a)



(b)

Figure 4.4: Fractal dimension analysis of the “low field” simulations. (a) The smaller, 50nm radius sample. (b) The larger, 75nm radius sample.

The two simulated dendrites have fractal dimensions of 1.5990 and 1.5831 for the smaller and larger device simulation, respectively. In between the two is the fractal dimension of the physical sample, at 1.5876. This comparison shows that the physical sample and the generated images are similar in complexity, but does not speak to how similar the structure is to a dendrite. For reference, a Pascal triangle modulo 2 (Figure

4.5) has a fractal dimension of 1.5849. Therefore, it is important to also analyze the branches.

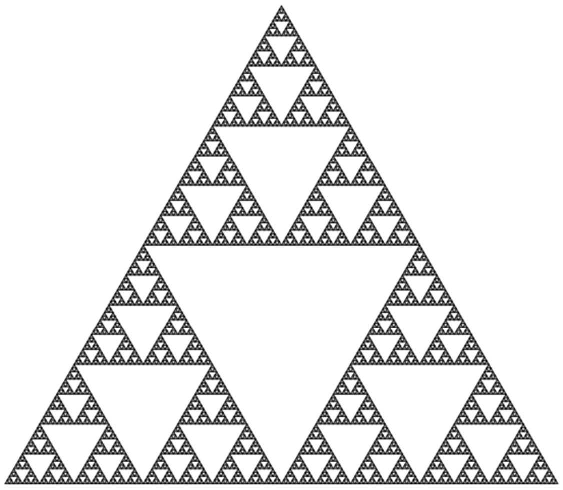


Figure 4.5: A Pascal triangle modulo 2 possess a fractal dimension of 1.5876.

The physical sample in Figure 4.3 can be seen to have four initial branches extending from the center cathode point, with first bifurcation angles of 80°, 34°, 55°, and 66° along the major arms starting in the top left and continuing counter-clockwise. The two simulated samples in Figure 4.4 each have four initial branches as well with very similar angles represented including angles near 80° and 35°. This wide range of angles can be attributed to the diffusion-dominated nature of the ions under a low electric field. The lengths of the major arms in these low field samples are noticeably larger than the minor branches with at least a 2:1 size ratio of the "successful" branches that continue growing compared against the "failed" branches that terminate shortly after division. Overall, the simulated dendrites appear to correlate with the physical sample in the low field experiment.

4.2 High Field

The next comparison is of the "high field" experiment. The physical sample, Figure 4.6, was grown on the same wafer as the low field sample in order to minimize concerns of process variation and was grown for 1000s under an applied bias of 40V resulting in another dendritic structure that was approximately 1cm in diameter. The simulated samples were in turn given an increased applied bias of 100V, shown in Figure 4.7.

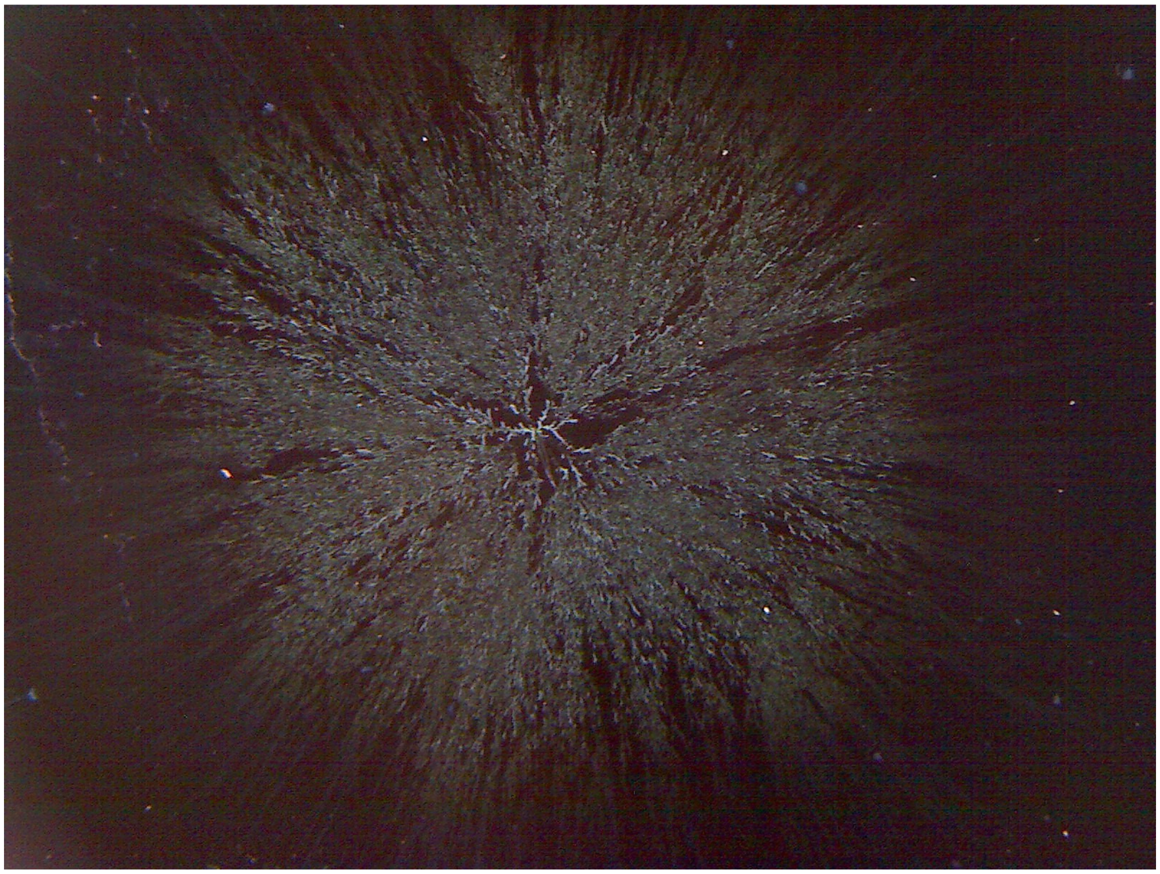
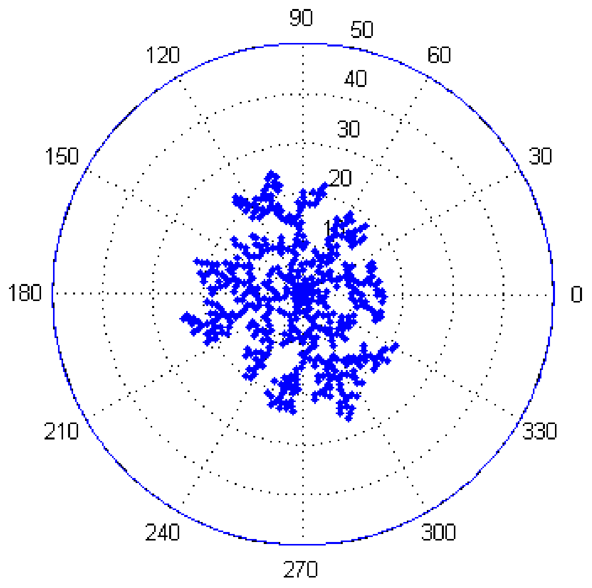
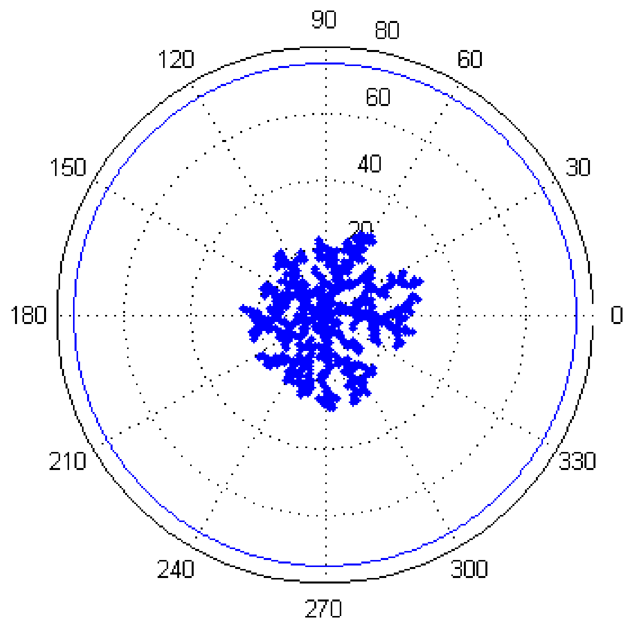


Figure 4.6: "High field" physical sample at minimum magnification. Approximately 1cm wide, grown for 1000s with an applied bias of 40V.



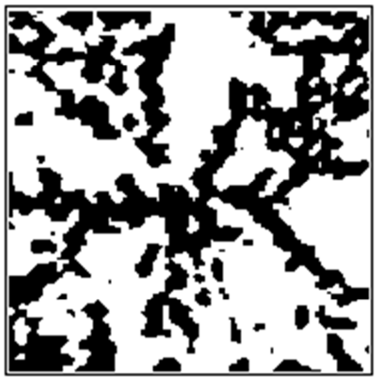
(a)



(b)

Figure 4.7: "Low field" simulated samples. (a) Radius 50nm, 500 ions, grown for several nanoseconds with an applied bias of 100V. (b) Radius 75nm, 600 ions, grown under the same conditions.

Even from cursory visual inspection, it is readily apparent that these higher field samples are much more compact and have many more intricate branches than the low field experiments. This validates the stochastic simulation's results qualitatively. The fractal dimension analysis is repeated on all of the samples and Figure 4.8 shows the result for the physical sample while Figure 4.9 shows the results for the smaller and larger simulated samples.



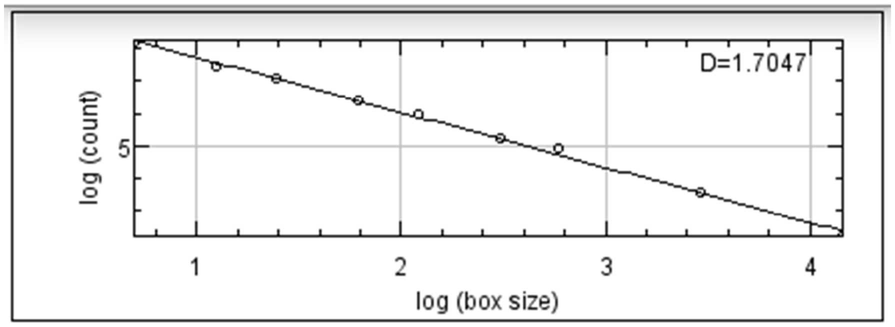
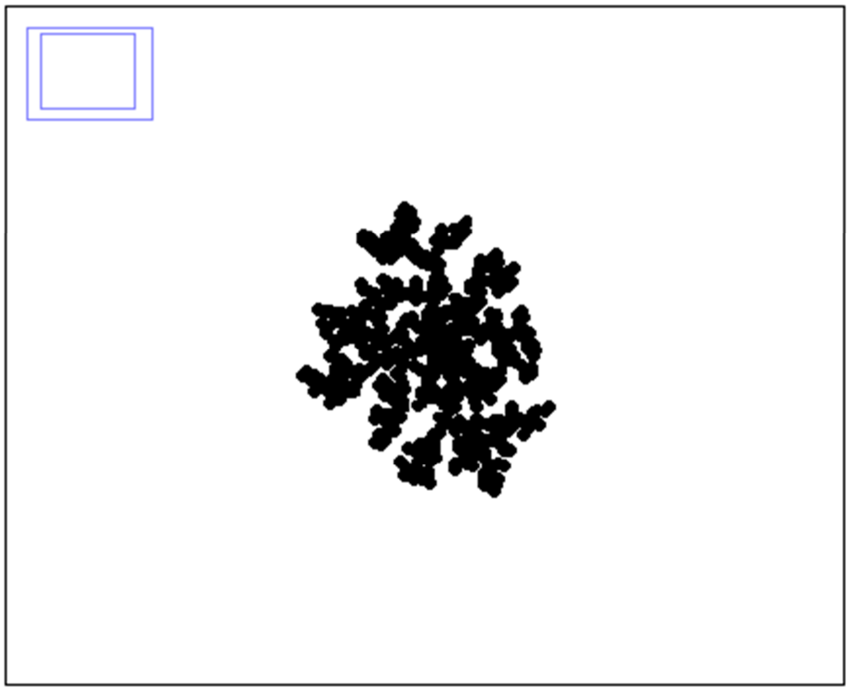
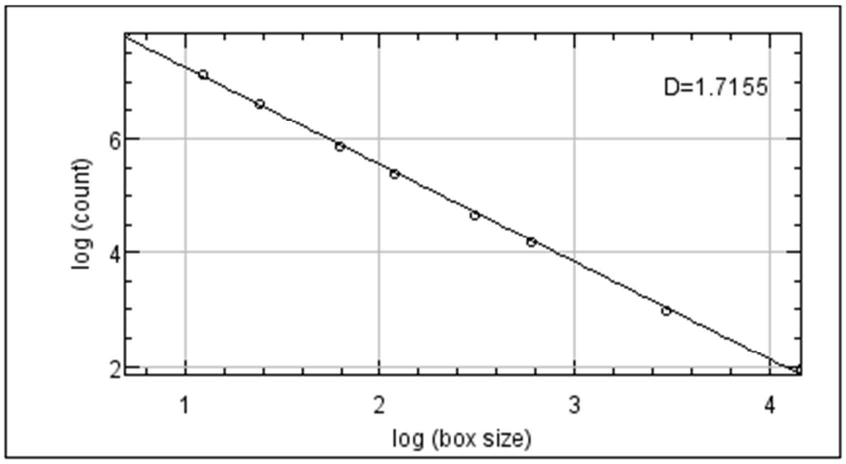
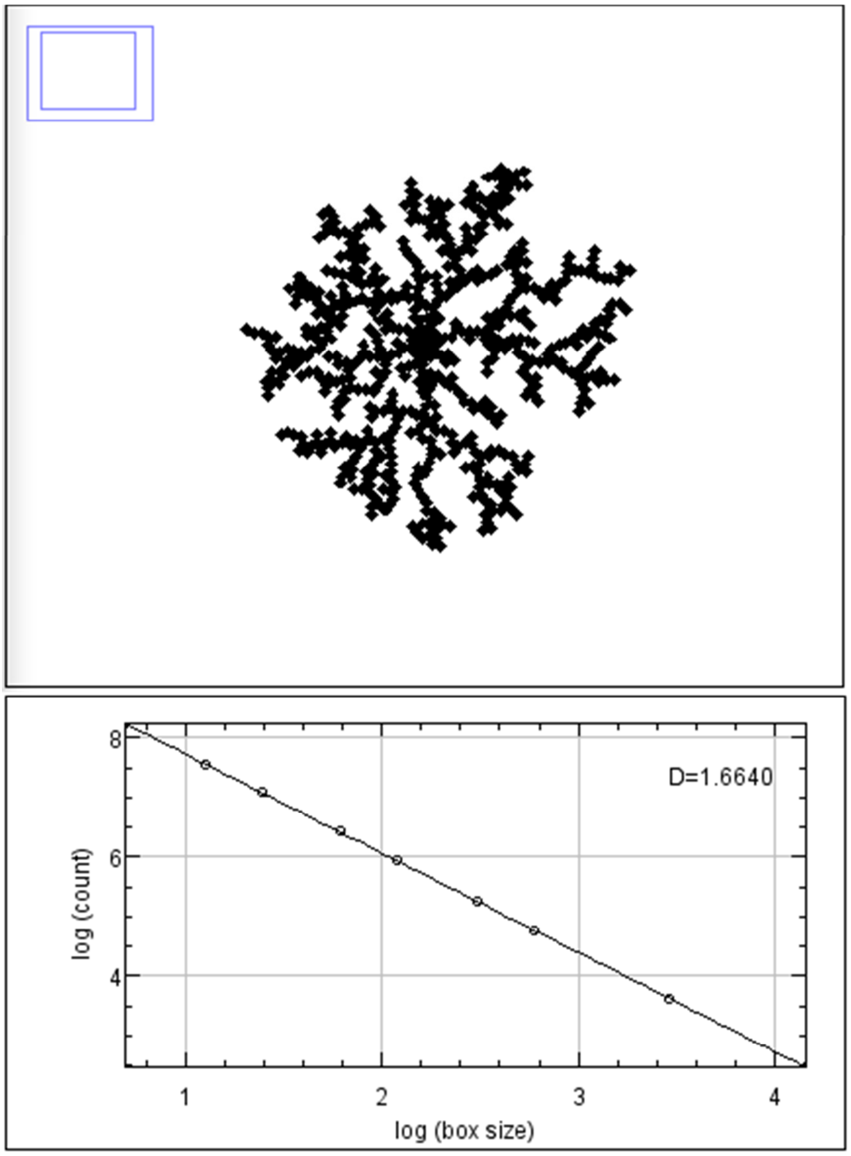


Figure 4.8: Fractal dimension analysis of the “high field” physical sample. Image has been cropped to focus on the region near the cathode point.





(a)



(b)

Figure 4.9: Fractal dimension analysis of the “high field” simulations. (a) The smaller, 50nm radius sample. (b) The larger, 75nm radius sample.

Once again the physical sample with a fractal dimension of 1.7047 is in between the two simulated samples with dimensions of 1.7155 and 1.6640. These complexity measures objectively show that the high field experiments result in a denser and more intricate dendrite morphology.

Analyzing the branches of the high field samples is more challenging than the low field samples due to this increased intricacy. While the low field samples clearly have what can be considered major branches and minor branches, the high field samples are much more uniform. Resolution of the images to analyze is also a larger concern. These difficulties notwithstanding, the physical sample has at least six branches stemming from the cathode while the simulated samples both have six to seven branches. This increase in initial branch count with higher field may be due to the greater dependence upon drift over diffusion in these setups. In the low field samples, it takes a lucky ion to diffuse to the cathode to start a branch and then that branch starts to become like a net of sorts that captures other diffusing ions. In contrast, the ions in a high field sample will be more motivated to reach the cathode from the beginning and more initial branches will form before ions are predominantly captured. Branch angles are also more homogenous in the high field samples, with the majority of bifurcation angles occurring around 60° to 80° and comparatively few angles less than 45°. Here again, the lessened influence of diffusion, and therefore randomness, in these experiments may be the cause of this behavior.

CHAPTER 5

CONCLUSION

In this thesis, three models were developed and evaluated to serve the need for a simulation of metallic dendrite growth in a PMC device. All models were developed in MATLAB. The Euler model uses Newtonian mechanics to govern ion motion and the Euler numerical method to calculate particle behavior. This model is highly performant, but suffers in realistic accuracy due to the limited number of physical parameters involved. The Verlet model was developed to address the limitations of the Euler model by incorporating more physical parameters, but this necessitated a more accurate numerical method for tracking high-velocity particles. The Verlet numerical method was applied successfully, but the simulation as whole is still divergent from reality in that the magnitudes of certain parameters must be set to unreasonable values. Finally, a completely different approach was pursued using Mott-Gurney's ion hopping theory applied to solids to govern particle motion and the Euler numerical method for calculations. This stochastic model also makes use of the most robust set of physical parameters to create the most-favored simulation overall.

The accuracy claims of the stochastic model are backed by comparisons to real world samples of grown dendritic structures. Initial expectations assumed that high field growth would result in a lower dendrite branch density due to ions stacking predominantly along earlier established chains. The classical model results followed this expectation. However, the stochastic model followed the opposite trend and it was the low field growth that showed the sparsest branching. These results were vindicated by real samples grown at 10V (low field) and 40V (high field) which clearly show a difference in dendrite morphology through analysis of fractal dimension that is consistent with the stochastic model’s results. Further comparisons evaluating the number of initial dendrite branches and common bifurcation angles not only showed good agreement between the physical samples and the simulation results, but also provided further evidence to support the stochastic model’s predictions.

REFERENCES

[1] M. M. Waldrop, “The chips are down for Moore’s law”, *Nature*, vol. 530, no. 7589, pp. 144-147.

[2] F. Masuoka, M. Momodomi, Y. Iwata, R. Shirota, “New ultra high density EPROM and flash EEPROM with NAND structure cell”, *Electron Devices Meeting*, vol. 33, pp. 552-555, 1987.

[3] J. S. Meena, S. M. Sze, U. Chand, and T. Tseng, “Overview of emerging nonvolatile memory technologies”, *Nanoscale Research Letters*, no. 9, p. 526.

[4] “More-than-Moore”, *International Technology Roadmap for Semiconductors*. [Online]. Available: http://www.itrs2.net/itrs-models-and-papers.html

[5] “CBRAM Technology”, *Adesto Technologies*. [Online], Available: http://www.adestotech.com/about-us/technologyip/

[6] N. F. Mott, R. W. Gurney, “Electronic processes in ionic crystals”, *J. Phys. Chem.* vol. 42, pp. 1142, 1941.

[7] M. N. Huda, A. K. Ray, “Electronic structures and magic numbers of small silver clusters: A many-body perturbation-theoretic study”, *Phys. Rev. A*, vol. 67, 013201, 2003.

[8] M. Ribes, E. Bychkov, A. Pradel, “Ion transport in chalcogenide glasses: Dynamics and structural studies”, *J. Optoelectron Adv. Mater.*, vol. 3, no. 3, pp. 665-674, 2001.

[9] D. B. Strukov, R. S. Williams, “Exponential ionic drift: fast switching and low volatility of thin-film memristors”, *Appl Phys A*, vol. 94, pp. 515-519, 2009.

[10] A. T. Fromhold Jr., E. L. Cook, “Diffusion currents in large electric fields for discrete lattices”, *J. Appl. Phys.*, vol. 38, no. 4, pp. 1546, 1967.

[11] M. J. Dignam, “Ion transport in solids under conditions which include large electric fields”, *J. Phys. Chem. Solids*, vol. 29, pp. 249-260, 1968.

[12] A. Chen, J. Hutchby, V. Zhirnov, G. Bourianoff, *Emerging Nanoelectronic Devices*. Wiley, 2015, pp. 140.

[13] A. Zakery, S. R. Elliot, *Optical Nonlinearities in Chalcogenide Glasses and Their Applications*. Springer, 2007, pp. 127.

[14] N. J. Giordano, H. Nakanishi, *Computational Physics*. Pearson, 2006, pp. 456.

[15] “The Verlet algorithm”, Furio Ercolessi. [Online]. Available: http://www.fisica.uniud.it/~ercolessi/md/md/node21.html

[16] W. Ye, C. Shen, J. Tian, C. Wang, C. Hui, and H. Gao, “Controllable growth of silver nanostructures by a simple replacement reaction and their SERS studies”, *Solid State Sci.*, vol. 11, no. 6, pp. 1088-1093, 2009.

APPENDIX A

STOCHASTIC MODEL MATLAB CODE

```
%--------------------------------------------------------------------
% Function:
% Main (dendrite_radial_voltage_ramp)
%--------------------------------------------------------------------
% Arguments:
% argc = 0
%
% Returns:
% None
%--------------------------------------------------------------------
% Dependencies:
% SpawnParticle
% DrawPlot
%--------------------------------------------------------------------
% Description:
% Main script for particle transport sim. Change parameters under
section
% "Sim Parameters" to modify simulation behavior.
%--------------------------------------------------------------------
clear all;

% sim start time
time_start = datestr(now);
fprintf('Sim started: %s\n', time_start)

ii = 1; % generic sentinel var i
jj = 1; % generic sentinel var j

%% Sim Parameters
%
%
particles_to_spawn = 200; % number of particles to spawn during
sim (set '0' for no replenishment)
particle_spawn_time = inf; % time units between particle spawns
(set 'inf' for only replenishment)
init_cathode_size = 2; % radius of particle circle to place at
the cathode at init (set '0' for no size)
init_particle_count = 400; % number of particles in grid at init
(set '0' for no particles)
grid_radius = 50; % size of grid space
t = 30000; % max sim time

%% Physical Parameters
%
%

% parameters (user-defined)
start_V = 0.4; % start of voltage ramp (V) (if V_ramp_time = 0,
this is the applied V)
end_V = 4; % end of voltage ramp (V)
```

```
V_ramp_time = 10000; % # time units to ramp voltage (set '0' for
no ramp)
lattice_const = 3e-10; % barrier width (m)
Wa0 = 0.5; % barrier height (eV)
T = 300; % temperature (K)
v = 1; % hopping frequency

% constants
kb = 8.617e-5; % boltzmann constant (eV/K)
e0 = 8.854e-12; % vacuum permittivity (F/m)
ze = 1.602e-19; % cation charge (eV)

% derived constants
ke = 1/(4*pi*e0); % Coulomb constant (m/F)

% parameters (derived)
vt = kb*T; % thermal voltage (eV)
Wa0_eff = Wa0 - vt; % thermally adjusted barrier height (eV)

if (V_ramp_time == 0) % applied V at cathode (V)
      V = start_V;
else
      V(ii) = start_V;
      for ii = 2:V_ramp_time
            V(ii) = V(ii-1) + (1/V_ramp_time) * (end_V - start_V);
      end
      V(V_ramp_time+1) = end_V;
end

for ii = (size(V,2)+1):t
      V(ii) = V(ii-1);
end

%% Structural Definition
%
%
center = 0; % center of grid

%% Init Grid Occupancy
%
%
existing_particles = 0; % number of particles in the grid
spawned_particles = 0; % number of particles spawned
all_rho =
zeros(particles_to_spawn+10*init_cathode_size^2+init_particle_cou
nt, t); % rho of all particles at all times
all_theta =
zeros(particles_to_spawn+10*init_cathode_size^2+init_particle_cou
nt, t); % theta of all particles at all times
```

```
bound =
zeros(particles_to_spawn+10*init_cathode_size^2+init_particle_cou
nt,1); % particle location and bound state
num_bound = 0; % number of particles bound

% create particles in a circle around origin
all_rho_exist = 0;
all_theta_exist = 0;
if (init_cathode_size > 0)
      % place first particle at cathode center
      existing_particles = existing_particles + 1;
      this_rho = 0;
      this_theta = 0;
      all_rho(existing_particles,1) = this_rho;
      all_theta(existing_particles,1) = this_theta;
      all_rho_exist(existing_particles) = this_rho;
      all_theta_exist(existing_particles) = this_theta;
      bound(existing_particles) = 1;
      num_bound = num_bound + 1;
      % place remaining particles radially outward
      this_rho = 1;
      while (this_rho <= init_cathode_size)
            for this_theta = 0:.01:2*pi
                  if ( sum(sqrt(all_rho_exist(:).^2+this_rho^2-
            2.*all_rho_exist(:).*this_rho.*cos(this_theta-
            all_theta_exist(:))) <= .99) < 1 )
                        existing_particles = existing_particles +
                  1;
                        all_rho(existing_particles,1) = this_rho;
                        all_theta(existing_particles,1) =
                  this_theta;
                        all_rho_exist(existing_particles) =
                  this_rho;
                        all_theta_exist(existing_particles) =
                  this_theta;
                        bound(existing_particles) = 1;
                        num_bound = num_bound + 1;
                  end
            end
            this_rho = this_rho + 1;
      end
end

if (init_particle_count > 0)
      spawn_error = 0;
      spawn_rho = 'random';
      spawn_theta = 'random';
      for ii = 1:init_particle_count
            [all_theta(existing_particles+1,1),
      all_rho(existing_particles+1,1), spawn_error] =
      SpawnParticle(spawn_theta, spawn_rho, grid_radius,
      all_theta(:,1), all_rho(:,1));
           if (spawn_error)
                error('t=0 ERROR: Could not spawn particle. [init
           particle spawn routine]')
           end
           existing_particles = existing_particles + 1;
      end
end

num_particles = existing_particles + particles_to_spawn; % total
number of particles

% trim position matrices to remove excess space
all_rho = all_rho(1:num_particles,:);
all_theta = all_theta(1:num_particles,:);
bound = bound(1:num_particles);

% create single time step vectors
next_rho = all_rho(1:num_particles)'; % rho of all particles at
current time step
next_theta = all_theta(1:num_particles)'; % theta of all
particles are current time step

%% Particle Info
%
%
move_theta = (0:0.4:2*pi)'; % theta for random walk
diff_vec_x = zeros(num_particles,1); % diffusion vector x
component
diff_vec_y = zeros(num_particles,1); % diffusion vector y
component
diff_pmf_base(1:size(move_theta,1),1) = 1/size(move_theta,1); %
probability weighting factors for diffusion direction
diff_pmf_barriermod = 0; % weighting factor for how much the
diffusion barrier is altered by the potential source
index_rotate = floor(size(move_theta,1)/2); % derived parameter
for # of indices in pmf that span pi/2 rad
diff_pmf_size = size(diff_pmf_base,1); % size of pmf array (calc
once for optimization)
drift_dir_index = zeros(num_particles,1); % index in move_theta
corresponding to drift direction
drift_dir_pi_index = zeros(num_particles,1); % index in
move_theta corresponding to drift direction + pi
time_until_hop = zeros(num_particles,1) + v; % timer for hopping
attempt frequency

%% First Particle Spawn
%
%
```

```
spawn_error = 0; % flag set if there is a problem spawning
particle
force_particle_spawn = 0; % flag to force a particle to spawn
spawn_theta = 'random'; % theta of particle spawn location
spawn_rho = grid_radius; % rho of particle spawn location
if (particles_to_spawn > 0)
      [all_theta(existing_particles+1,1),
all_rho(existing_particles+1,1), spawn_error] =
SpawnParticle(spawn_theta, spawn_rho, grid_radius,
all_theta(:,1), all_rho(:,1));
      if (spawn_error)
            error('t=0 ERROR: Could not spawn particle. [first
      spawn routine]');
      end
      existing_particles = existing_particles + 1;
      spawned_particles = spawned_particles + 1;
end

spawn_theta = zeros(num_particles,1);
next_theta(existing_particles) = all_theta(existing_particles,1);
next_rho(existing_particles) = all_rho(existing_particles,1);

%% Particle Transport
%
%
particle_spawn_timer = particle_spawn_time;

for n = 1:t
      % init position for the new timestep
      all_theta(:,n) = next_theta(:);
      all_rho(:,n) = next_rho(:);

      % time saving optimization
      % if all particles are bound, stop time loop
      if (num_bound == num_particles)
            fprintf('All particles bound at %d\n', n)
            t = n;
            break;
      end

      for ii = 1:existing_particles
            % check if the particle is not bound
            if (bound(ii) == 0)
                  % current location in cartesian
                  [this_loc_x,this_loc_y] =
            pol2cart(next_theta(ii),next_rho(ii));

                  % all particles
                  [all_loc_x,all_loc_y] =
            pol2cart(all_theta(:,n),all_rho(:,n));
                  all_loc_x = all_loc_x(1:existing_particles);
                  all_loc_y = all_loc_y(1:existing_particles);
                  all_theta_exist =
            all_theta(1:existing_particles,n);
                  all_rho_exist = all_rho(1:existing_particles,n);

                  % find all bound particles
                  bound_index = find(bound==1);
                  bound_loc_x = all_loc_x(bound_index);
                  bound_loc_y = all_loc_y(bound_index);

                  % if there are any bound particles, check for new
            binds
                  if (bound_index > 0)
                        % if particle is touching another bound
                  particle, it gets
                        % bound
                        if (
                  sum(sqrt(next_rho(bound_index).^2+next_rho(ii)^2-
                  2.*next_rho(bound_index).*next_rho(ii).*cos(next_
                  theta(ii)-next_theta(bound_index))) <= 1.1) > 0 )
                              bound(ii) = 1;
                              num_bound = num_bound + 1;
                              force_particle_spawn =
                        force_particle_spawn + 1;
                              spawn_theta(force_particle_spawn) =
                              next_theta(ii);
                        end
                  end

                  % if current particle is not bound, calculate
            movement
                  if (bound(ii) == 0)
                        if (time_until_hop(ii) == 0)
                              % calculate next move
                              this_loc_x = this_loc_x +
                              diff_vec_x(ii);
                              this_loc_y = this_loc_y +
                        diff_vec_y(ii);
                              [this_theta, this_rho] =
                        cart2pol(this_loc_x,this_loc_y);

                              % if particle moved out of valid area,
                        bring it back in
                              if (this_rho > grid_radius)
                                    this_rho = grid_radius;
                                    [this_loc_x,this_loc_y] =
                              pol2cart(this_theta,this_rho);
                              end
                  % if the particle is trying to land on
            another
                  % particle, walk it back
                  all_rho_exist(ii) = this_rho;
                  all_theta_exist(ii) = this_theta;
                  runaway = 0;
                  if (
            sum(sqrt(all_rho_exist.^2+this_rho^2-
            2.*all_rho_exist.*this_rho.*cos(this_theta-
            all_theta_exist)) < 1) > 1 )
                       this_loc_x = this_loc_x - (0.1 *
                  diff_vec_x(ii));
                       this_loc_y = this_loc_y - (0.1 *
                  diff_vec_y(ii));
                       [this_theta, this_rho] =
                  cart2pol(this_loc_x,this_loc_y);
                       all_rho_exist(ii) = this_rho;
                       all_theta_exist(ii) =
                  this_theta;
                       while (
                  sum(sqrt(all_rho_exist.^2+this_rho^2-
                  2.*all_rho_exist.*this_rho.*cos(this_t
                  heta-all_theta_exist)) < 1) > 1 )
                            this_loc_x = this_loc_x -
                       (0.1 * diff_vec_x(ii));
                            this_loc_y = this_loc_y -
                       (0.1 * diff_vec_y(ii));
                            [this_theta, this_rho] =
                            cart2pol(this_loc_x,this_lo
                            c_y);
                            all_rho_exist(ii) =
                       this_rho;
                            all_theta_exist(ii) =
                       this_theta;
                            runaway = runaway + 1;
                            if (runaway == (1/.1))
                                 fprintf('Warning:
                            Large particle location
                            adjust\n');
                            end
                       end
                  end
                  next_theta(ii) = this_theta;
                  next_rho(ii) = this_rho;

                  % if particle reaches the cathode,
            it's automatically bound
                  if (abs(this_rho) < center + .5)
                       bound(ii) = 1;
                       num_bound = num_bound + 1;
```

```
        this_rho = 0;
        next_theta(ii) = this_theta;
        next_rho(ii) = this_rho;
        force_particle_spawn = 1;
        spawn_theta = this_theta;
else
        % calculate particle motion
parameters

        % DRIFT

        if (size(bound_index,1) > 0)
                this_loc_x(1:size(bound_ind
        ex),1) = this_loc_x;
                this_loc_y(1:size(bound_ind
        ex),1) = this_loc_y;

                % drift direction, from
        particle to each bound location
                drift_dir_x = bound_loc_x -
        this_loc_x;
                drift_dir_y = bound_loc_y -
        this_loc_y;
                drift_dir =
        wrapTo2Pi(atan2(drift_dir_y,
        drift_dir_x));

                % slope from bound to ion
                m = (this_loc_y -
        bound_loc_y) ./ (this_loc_x -
        bound_loc_x);

                % coords of anode
                edge_x = sign(this_loc_x)
        .* sqrt(grid_radius^2 .* m.^2 +
        grid_radius^2) ./ (m.^2 + 1); %
        take the distance from the
        cathode center (0,0) to the
        edge_x, solve for edge_x from
        (edge_y = m * edge_x + b) where
        m is between the bound loc and
        the ion.
                edge_y = m .* edge_x;

                % field direction from
        bound loc to anode
                field_dir =
        wrapTo2Pi(drift_dir + pi);
```

```
		% distance from bound loc
	to anode
		dist_from_anode =
	sqrt(bsxfun(@minus,edge_x,bound_
	loc_x).^2+bsxfun(@minus,edge_y,b
	ound_loc_y).^2);
	else
		% drift direction, always
	towards the center
		drift_dir_x = center -
	this_loc_x;
		drift_dir_y = center -
	this_loc_y;
		drift_dir =
	wrapTo2Pi(atan2(drift_dir_y,
	drift_dir_x));

		% distance from potential
	source
		dist_from_anode =
	grid_radius;
	end

	E = V(n) ./ (lattice_const *
dist_from_anode); % E = keQ/r^2,
Q=Vr/ke

	% DIFFUSION

	diff_pmf = diff_pmf_base;
	for jj = 1:size(drift_dir,1)
		% calculate barrier
	modification weight
		diff_pmf_barriermod = 1 * (
	(lattice_const * E(jj) / 2) /
	(Wa0_eff) ) / size(drift_dir,1);
	% (aE/2)/(Wa0-kbT) [eV] => %
	barrier mod

		[~, drift_dir_index(jj)] =
	min(abs(move_theta -
	drift_dir(jj))); % 2nd result of
	min() is the index

		drift_dir_pi_index(jj) =
	drift_dir_index(jj) +
	index_rotate;
		if (drift_dir_pi_index(jj)
	> diff_pmf_size)
```

```
      drift_dir_pi_index(jj) = drift_dir_pi_index(jj) -
diff_pmf_size;
                                       end

                                       % enhance diff_pmf in
                                 direction of drift and reduce
                                       % diff_pmf in the opposite
                                 direction
                                       diff_pmf(drift_dir_index(jj
                                 )) =
                                 diff_pmf(drift_dir_index(jj)) +
                                 diff_pmf_base(drift_dir_index(jj
                                 )) * diff_pmf_barriermod;
                                       diff_pmf(drift_dir_pi_index
                                 (jj)) =
                                 diff_pmf(drift_dir_pi_index(jj))
                                 -
                                 diff_pmf_base(drift_dir_pi_index
                                 (jj)) * diff_pmf_barriermod;
                                       if
                                 (diff_pmf(drift_dir_pi_index(jj)
                                 ) < 0)

      diff_pmf(drift_dir_pi_index(jj)) = 0;
                                       end
                                 end

                                 % choose a weighted random
                           direction for diffusion
                                 diff_dir =
                           randsample(move_theta,1,true,diff_pmf)
                           ;
                                 diff_vec_x(ii) = cos(diff_dir);
                                 diff_vec_y(ii) = sin(diff_dir);

                                 % reset time until next hop
                                 time_until_hop(ii) = v;
                           end
                     else
                           time_until_hop(ii) =
                     time_until_hop(ii) - 1;
                     end
                end
           end
           % update position for the new timestep
           all_theta(ii,n) = next_theta(ii);
           all_rho(ii,n) = next_rho(ii);
      end
```

```
     % spawn new particles after updating existing particles

     % only spawn new particles when the timer expires
     if (particle_spawn_timer == 0 || force_particle_spawn > 0)
          % check to see if we've spawned the maximum number of
          % particles already
          for ii = 1:force_particle_spawn
               if (spawned_particles < particles_to_spawn)
                    [all_theta(existing_particles+1,n),
               all_rho(existing_particles+1,n), spawn_error] =
               SpawnParticle(spawn_theta(ii), spawn_rho,
               grid_radius, all_theta(:,n), all_rho(:,n));
                    if (spawn_error)
                         break;
                    end
                    existing_particles = existing_particles +
               1;
                    spawned_particles = spawned_particles + 1;
                    % existing_particles got incremented
                    next_theta(existing_particles) =
               all_theta(existing_particles,n);
                    next_rho(existing_particles) =
               all_rho(existing_particles,n);
               end
          end
          particle_spawn_timer = particle_spawn_time;
          force_particle_spawn = 0;
     end
     particle_spawn_timer = particle_spawn_timer - 1;

     % Display sim time in command window
     fprintf('Elapsed Time: %d s\n', n)
end

if (spawn_error)
     error('t=%d ERROR: Could not spawn particle. [particle
transport routine]',n)
end
% sim finished time
time_finish = datestr(now);
fprintf('Sim finished: %s\n', time_finish)
%% Result Plot
%
% set first arg to 1 for final plot only, 2 for particle movie

DrawPlot(1, t, grid_radius, all_theta(:,:), all_rho(:,:));

%-----------------------------------------------------------
% Function:
% SpawnParticle
```

```
%-----------------------------------------------------------
% Arguments:
% argc = 5
% argv(1) = spawn_theta (scalar)
% argv(2) = spawn_rho (scalar)
% argv(3) = grid_radius (scalar)
% argv(4) = all_theta (1d vector)
% argv(5) = all_rho (1d vector)
%
% Returns:
% return(1) = spawn_theta (scalar)
% return(2) = spawn_rho (scalar)
% return(3) = error (scalar)
%-----------------------------------------------------------
% Dependencies:
% None
%-----------------------------------------------------------
% Description:
% Spawn particles at a position inside the active region.
%-----------------------------------------------------------
function [ spawn_theta, spawn_rho, error ] = SpawnParticle(
spawn_theta, spawn_rho, grid_radius, all_theta, all_rho )

valid_start_pos = 0;
timeout = 0;

spawn_mode_theta = spawn_theta;
spawn_mode_rho = spawn_rho;

theta = 0:.01:2*pi;
theta_tol = .1745; % 10 degrees

while (valid_start_pos == 0 && timeout < 1000)
     if (spawn_mode_theta == 'random')
          spawn_theta =
     wrapToPi(theta(randi([1,size(theta,2)])));
     else
          spawn_theta = spawn_theta + (-theta_tol +
     2*theta_tol*rand);
     end

     if (spawn_mode_rho == 'random')
          spawn_rho = 1+grid_radius*rand;
     end

     if ( sum(sqrt(all_rho.^2+spawn_rho^2-
2.*all_rho.*spawn_rho.*cos(spawn_theta-all_theta)) <= 1) == 0 )
          valid_start_pos = 1;
     else
          timeout = timeout +  1;
```

```
      end
end

if (timeout < 1000)
      error = 0;
else
      error = 1;
end

end

%-----------------------------------------------------------------
% Function:
% DrawPlot
%-----------------------------------------------------------------
% Arguments:
% argc = 4
% argv(1) = plot_type
% argv(2) = t
% argv(3) = thetas
% argv(4) = rhos
%
% Returns:
% None
%-----------------------------------------------------------------
% Dependencies:
% None
%-----------------------------------------------------------------
% Description:
% Plots results and draws visual candy. If plot_type = 1, show
only end
% result. If plot_type = 2, show particle movie.
%-----------------------------------------------------------------
function [ ] = DrawPlot( plot_type, t, radius, thetas, rhos )

% Particle Plot
if (plot_type == 1)
      linet = 0:.01:2*pi;
      figure
      line = polar(linet,radius*ones(size(linet)));
      hold on
      polar(thetas(:,t),rhos(:,t),'.')
else
      numframes = t;
      fig1 = figure(1);
      winsize = get(fig1,'Position');
      winsize(1:2) = [0 0];
      A = moviein(numframes,fig1,winsize);
      set(fig1,'NextPlot','replacechildren')
```

```
        for j = 1:numframes
                %hold on
                polar(thetas(:,j),rhos(:,j),'.')
                A(:,j) = getframe(fig1,winsize);
        end

        if (plot_type == 3)
        % repeat the plot animation
        movie(fig1,A,30,60,winsize)
        end
end
```