Senior Design Reports (CBE)

Department of Chemical & Biomolecular Engineering

4-2011

Model-Based Control with Stochastic Simulators: Building Process Design and Control Software for Advanced Materials Processing Technology

Efrem Braun
University of Pennsylvania

Marija Mircevska
University of Pennsylvania

Manuel Molina Villalba
University of Pennsylvania

Braun, Efrem; Mircevska, Marija; and Villalba, Manuel Molina, "Model-Based Control with Stochastic Simulators: Building Process Design and Control Software for Advanced Materials Processing Technology" (2011). *Senior Design Reports (CBE)*. 25.
http://repository.upenn.edu/cbe_sdr/25

Model-Based Control with Stochastic Simulators: Building Process Design and Control Software for Advanced Materials Processing Technology

Abstract

An analysis was made on the financial feasibility of a start-up company that will sell software developed for the off-line optimization and on-line control of a thin film deposition process. This analysis found some niche applications for a potential startup company that sells thin film deposition modeling and control software solutions. Due to the potential versatility of the software that was developed, other potential markets may exist. This investigation found that the startup company can be competitive over a five year time horizon with a 20% IRR. Molecular modeling software that employs the kinetic Monte Carlo method was used for the simulation of thin film growth. Due to the capability of this model to retain both surface and internal atomic structure of the thin film, this model can simulate thin film properties such as roughness and porosity. Development work was done on producing a suitable objective function to represent a set of application-imposed thin film micro-structure property requirements. This objective function was used in the generation of an optimal transient profile. A model predictive control framework was designed to control film growth based on the objective function and the optimal transient evolution of the film. The model predictive control algorithm was analyzed and shown to perform the desired control.

Model-Based Control with Stochastic Simulators: Building Process Design and Control Software for Advanced Materials Processing Technology

Chemical and Biomolecular Engineering 459

Professor Leonard A. Fabiano

April 15, 2011

Efrem Braun

Marija Mircevska

Manuel Molina Villalba

Department of Chemical and Biomolecular Engineering

University of Pennsylvania

Project Adviser: Dr. Talid R. Sinno

Project Recommended By: Dr. Talid R. Sinno, University of Pennsylvania

April 15, 2010

Department of Chemical and Biomolecular Engineering
School of Engineering and Applied Sciences
University of Pennsylvania
220 S. 34th Street
Philadelphia, PA 19104

Dear Drs. Talid Sinno and Warren Seider, and Professor Leonard Fabiano,

Enclosed in this book is the final copy of our Senior Design Report on Model-Based Control and Stochastic Simulators: Building Process Design and Control Software for Advanced Materials Processing Technology. A software program was designed to model deposition processes with kinetic Monte Carlo simulations. Model Predictive Control is utilized to optimize and control thin film growth by manipulating a single growth parameter. The economic feasibility of a start-up software company was analyzed. It was found that the product fills a niche in the marketplace that is currently vacant. The necessary selling price of the product to achieve a 20% IRR was found to range from $4,000 to $11,000, which is within the range that industry has shown itself to be willing to pay for such a product.

Sincerely,

Efrem Braun　　　　Marija Mircevska　　　　Manuel Molina Villalba

Table of Contents

Abstract

An analysis was made on the financial feasibility of a start-up company that will sell software developed for the off-line optimization and on-line control of a thin film deposition process. This analysis found some niche applications for a potential startup company that sells thin film deposition modeling and control software solutions. Due to the potential versatility of the software that was developed, other potential markets may exist. This investigation found that the startup company can be competitive over a five year time horizon with a 20% IRR. Molecular modeling software that employs the kinetic Monte Carlo method was used for the simulation of thin film growth. Due to the capability of this model to retain both surface and internal atomic structure of the thin film, this model can simulate thin film properties such as roughness and porosity. Development work was done on producing a suitable objective function to represent a set of application-imposed thin film micro-structure property requirements. This objective function was used in the generation of an optimal transient profile. A model predictive control framework was designed to control film growth based on the objective function and the optimal transient evolution of the film. The model predictive control algorithm was analyzed and shown to perform the desired control.

1. Introduction

1.1. Thin Film Deposition Process and Applications

Thin film deposition is a process that is used to deposit solid material on a substrate surface. A diagram of a typical deposition reactor is presented in Figure 1. Chemical vapor deposition (CVD) and physical vapor deposition (PVD) are commonly used variants of thin film deposition. Chemical vapor deposition is a process in which gaseous precursor material flows into a chamber that contains a heated substrate surface to be coated (Sudarshan & Park, 2001). The precursor material reacts or decomposes on the surface, resulting in a deposition. The reaction is usually accompanied by the release of chemical byproducts, which are exhausted out of the chamber along with any non-reacted precursor gas. Enhanced CVD methods are available which involve plasma, ions, photons, hot filaments, lasers, etc. which usually increase deposition rates or lower required temperatures. Physical vapor deposition is a process in which a vaporized form of the material to be deposited flows over the substrate surface and condenses, resulting in a deposition (Mattox, 1998). No chemical reaction occurs in this process. Figure 2 compares PVD to CVD. A PVD model is used in the experiments in this report.

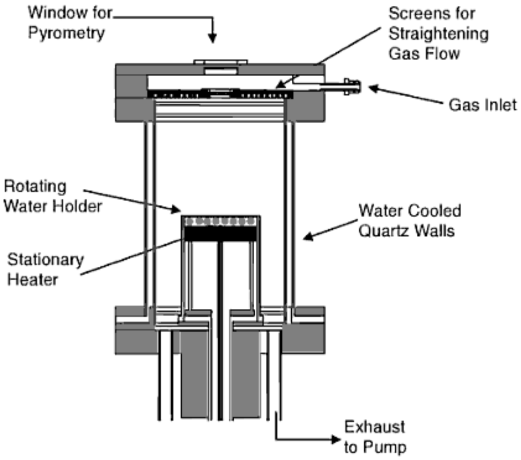


Figure 1: Deposition reactor. A substrate wafer is stationed on the rotating disk. The gas flux is perpendicular to the substrate surface.

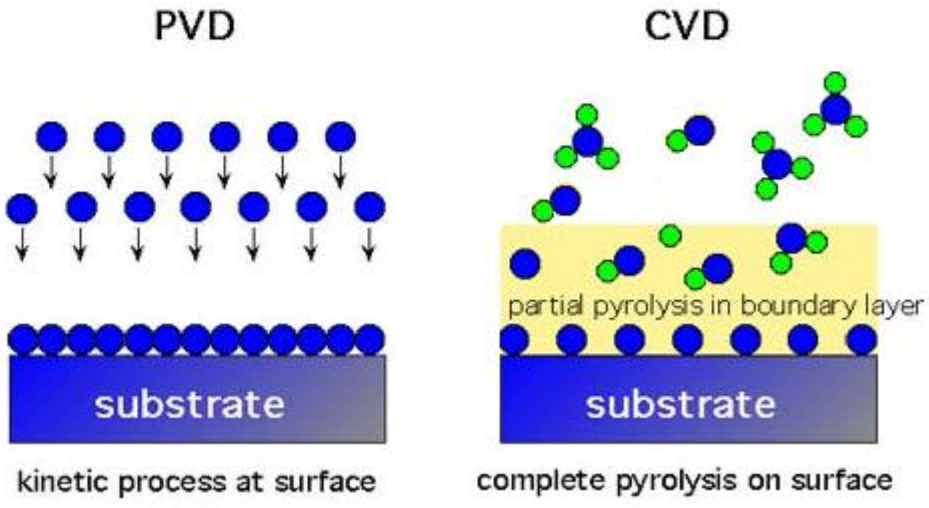

PVD
CVD
partial pyrolysis in boundary layer
substrate
substrate
kinetic process at surface
complete pyrolysis on surface

Figure 2: Physical and Chemical Vapor Deposition.

Thin film technology has a variety of applications, in microelectronic devices, optics, micro-electro-mechanical systems (MEMS), and biomedical products. With this process versatility, there is a vast range of optimal thin film property requirements that are associated with specific applications. Most of these film property requirements tend to be imposed on the microstructure of the thin film. For instance, in the semiconductor industry, thin films are required to have minimal surface roughness and minimal internal defects, which directly relate to electrical and mechanical properties of microelectronic devices (Hu et al., 2009). In photovoltaic applications, the surface roughness relates to the optical performance of solar cells (Xinyu et al., 2010). While surface roughness of thin films is related to interfacial properties at film layer junctions, internal defects, represented as a porosity measurement, affect mechanical and electrical properties. Thus, advanced control systems must be developed to obtain the desired microstructural properties.

1.2. Project Objectives

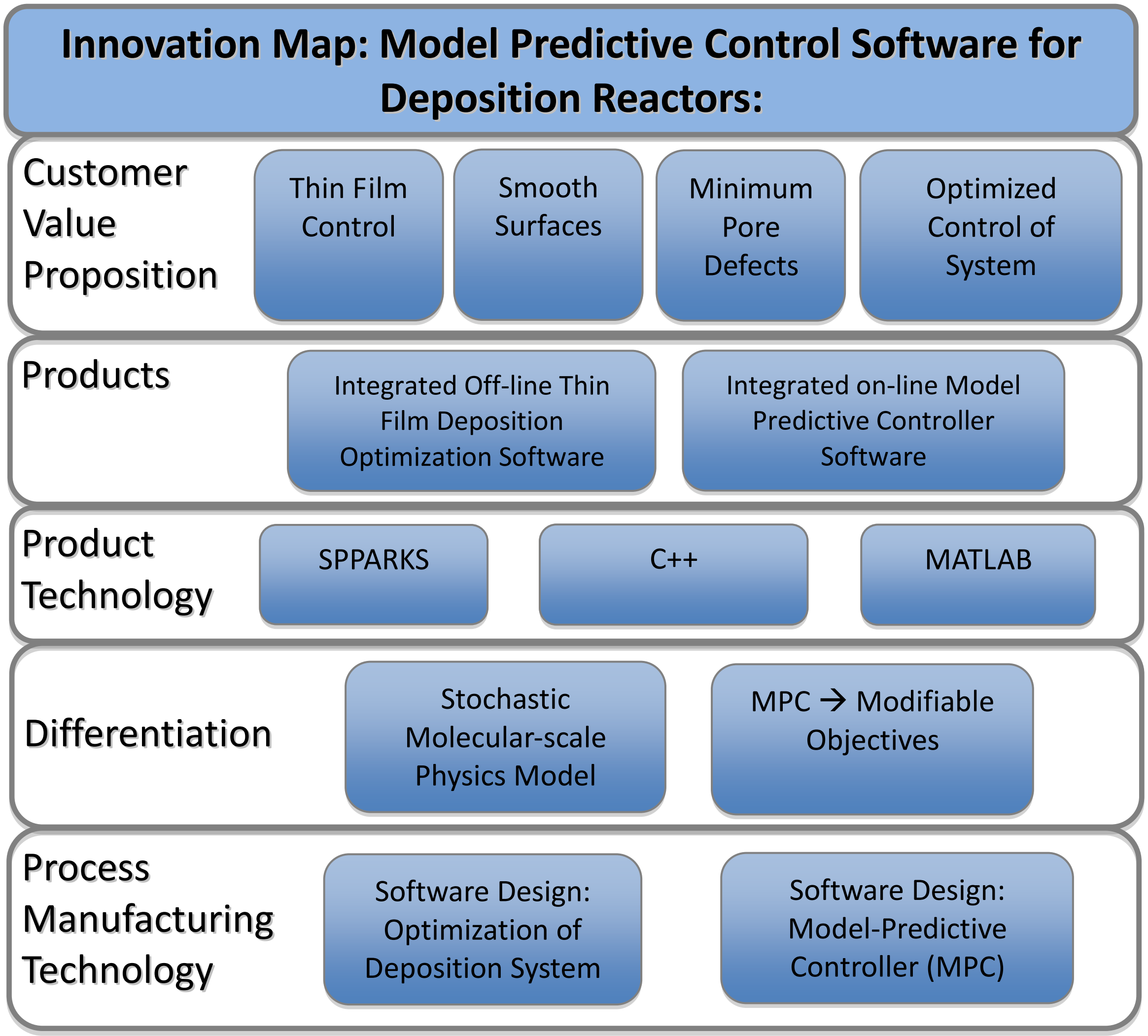
With the advent of more advanced micro devices, increasingly more stringent requirements translate into the need for more rigorous control over the surface and internal morphologies of thin films. With an interest in semiconductor applications, roughness and porosity must both be minimized for thin films on the order of 1-10 nm film thickness. A model predictive control algorithm with this purpose is developed in this report.

The project charter in Table 1 briefly describes the goals and scope of the project. Three primary goals were established, the first of which was to characterize a standardized Kinetic Monte Carlo Model (KMC) that is part of the Stochastic Parallel Particle Kinetic Simulator (SPPARKS) suite, used in the microscopic modeling of thin film deposition processes in this report. The second goal was to build a model predictive control (MPC) framework to dynamically control the thin film deposition process. Lastly, the third goal was to provide a feasibility plan for a start-up PVD solutions company that will develop its own robust client-need based KMC model and sell MPC software to clients that employ vapor deposition reactors.

Project Name	Stochastic Simulation and Model Predictive Control of Thin Film Vapor Deposition Process
Project Champions	Talid Sinno
Project Leaders	Efrem Braun, Marija Mircevska, and Manuel Molina Villalba
Specific Goals	• To characterize a developed kinetic monte carlo model • To develop software to optimize and control a thin film vapor single particle deposition processes
Project Scope	In Scope • Simulate a generic single particle deposition process • Observe effects of temperature, deposition rate, lattice size and binding energies on roughness and porosity of the film microstructure • Optimize film deposition reaction • Develop an off-line Optimization Method to determine an optimal state trajectory for roughness and porosity • Develop an on-line Model Predictive Controller to bring a system to desired state based on optimal profile Out of Scope • Modeling a specific single particle substrate deposition • Modeling multi-particle substrate deposition • Assessment on the validity of SPPARKS on the representation of the real chemical process
Deliverables	Software package built around an existing model to optimize and control thin film single particle growth
Timeline	• 4 weeks to characterize SPPARKS • 4 weeks develop Optimization and Model Predictive Control Software • 4 weeks to develop a case study

Table 1: Project Charter

The innovation map for a film deposition optimization and control product is shown in Figure 3 below.



Innovation Map: Model Predictive Control Software for Deposition Reactors:

Customer Value Proposition	Thin Film Control	Smooth Surfaces	Minimum Pore Defects	Optimized Control of System
Products	Integrated Off-line Thin Film Deposition Optimization Software	Integrated on-line Model Predictive Controller Software		
Product Technology	SPPARKS	C++	MATLAB	
Differentiation	Stochastic Molecular-scale Physics Model	MPC → Modifiable Objectives		
Process Manufacturing Technology	Software Design: Optimization of Deposition System	Software Design: Model-Predictive Controller (MPC)		

Figure 3. Innovation Map

1.3. Process Modeling

It is difficult to obtain real-time measurements of microstructural properties of thin films during the deposition process. In the design of a model predictive controller, a rigorous process model will be used to simulate a real PVD process in order to obtain these real-time measurements. A model predictive control algorithm will use a less rigorous version of the model to predict the film microstructure after a specified period of time, for different combinations of input parameters. The controller will calculate film properties based on film microstructure information provided by the model, and select the input variables which give the best result for that time period. The selected input variables will be provided as input to the rigorous deposition model, as well as to the actual physical process which is running in parallel with the rigorous simulation. Figure 4 below shows a schematic of the overall process.

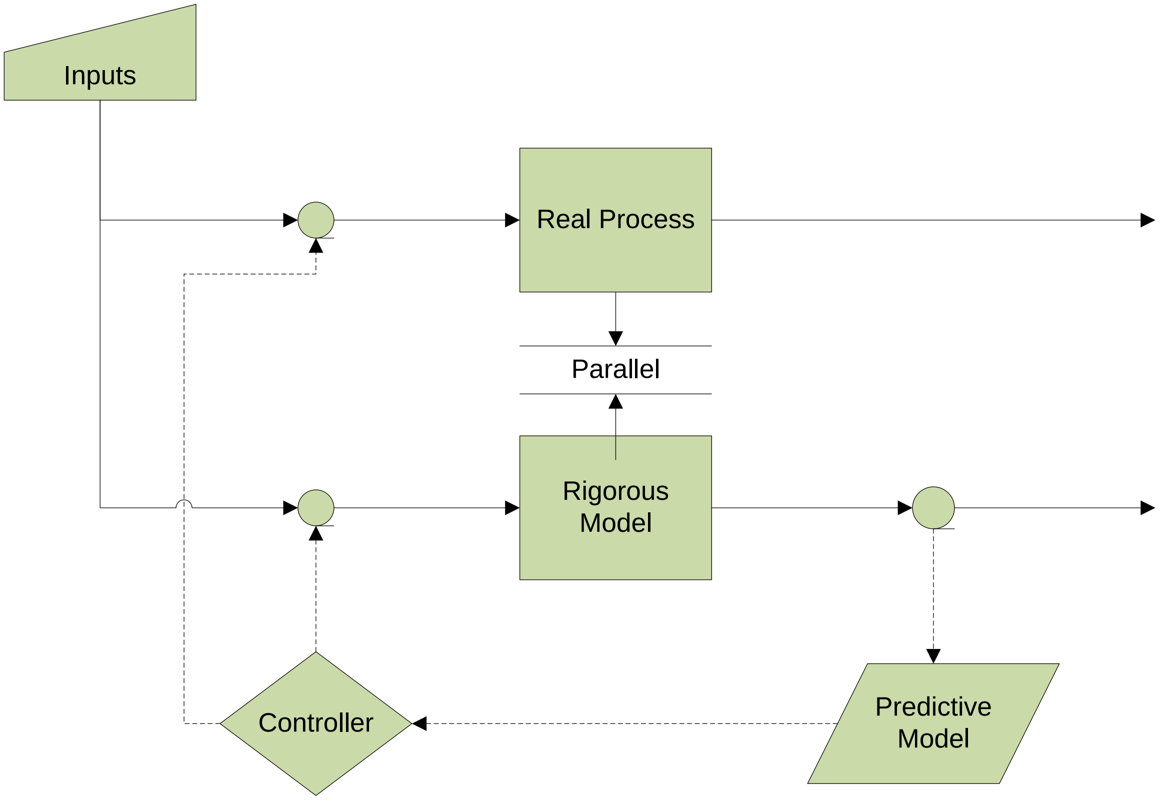


Figure 4: Process Flow Diagram of the Control Framework

We will use a simplified model of a deposition process, which does not take into account desorption. The goal of this project is to demonstrate the feasibility of a model predictive control unit that will control properties on an atomic scale. The controller can easily be modified to include more accurate process models.

Numerous simulation techniques exist for modeling the processes considered. The two major categories into which these techniques can be organized are deterministic and stochastic processes. A deterministic process is one in which the outcome is predetermined by the inputs (i.e. running the model another time with the same set of initial conditions will always result in the same outcome), while a stochastic process relies on randomness.

Generally, one develops deterministic simulations by deriving partial differential equations (PDEs), which can be numerically solved to give the outcomes. These PDEs can be on different scales. On one end is the continuum approximation, which assumes that materials are a continuous mass rather than discrete particles and builds a macromolecular-scale model based on dynamic conservation equations. On the other end lie techniques such as molecular dynamics, which use equations derived from statistical mechanics to predict atomic motion.

These deterministic methods cannot be used to model physical vapor deposition (PVD). The continuum approximation is invalid, as the microstructure of the film is important in determining the film's properties. However, this approximation is useful in modeling the heat and mass transport processes in the gas phase, which takes the depositing atoms to the surface where they adsorb. Molecular dynamics is infeasible due to the many calculations that this model requires; typical timesteps are on the order of one femtosecond, while PVD of thin films takes minutes to hours.

Kinetic Monte Carlo (KMC) is a general type of stochastic process which can be applied at any scale and in multiple areas. It is often used when a deterministic method would be incapable of obtaining a solution. KMC can be used when processes occur with a known rate. Thin film deposition lends itself to modeling by KMC because a finite number of events can occur at any given time and the rates of these events are well established; additionally, KMC can be used to model the evolution of the surface microstructure.

2. Kinetic Monte Carlo

2.1 Events

A Monte Carlo event is characterized by the type of the event and the site at which the event is executed. For thin film PVD, there are three main types of events: deposition, diffusion (or migration), and desorption. Only the first two, depicted in Figure 5, will be treated here because SPPARKS (the KMC software used) does not model desorption.

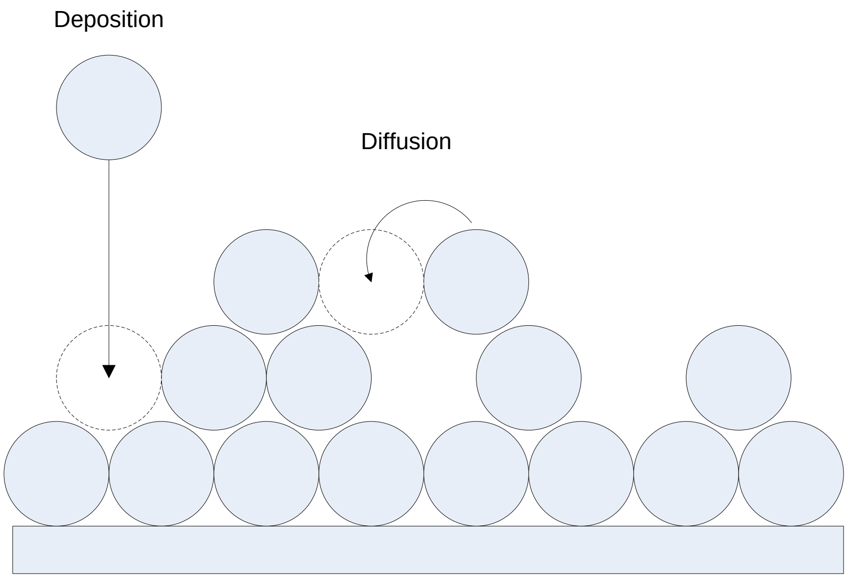


Figure 5: Types of KMC events

The deposition rate is considered to be site-independent and proportional to the flux of the incoming gas; thus, its rate can be stated as a constant in units of atomic depositions per second per site.

$$\hat{r}_{dep} = A_d \left[\frac{event}{site*second}\right] \qquad \textbf{Equation 1}$$

A diffusion event occurs when an atom is able to overcome an energy barrier arising from binding energies. Initially, it will be assumed that atoms can only diffuse to their nearest neighbors (defined in Figure 6 below). Diffusive hops to an atom’s second nearest neighbor, termed Schwoebel Hops, are discussed in a later section.

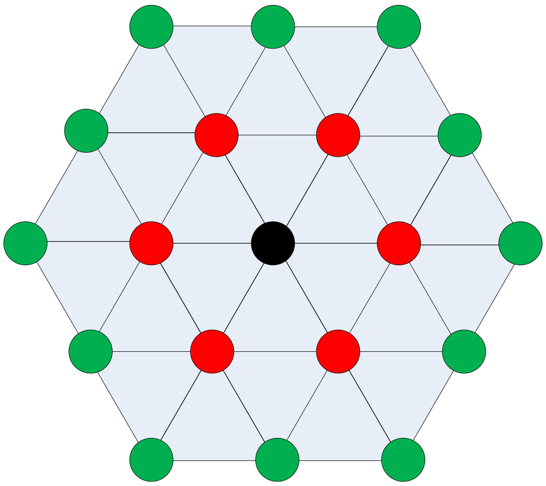


Figure 6: Tri-lattice Diagram. Black lines connect nearest neighbors. Black-colored particle in the center has six nearest neighbors (red-colored particles) and twelve second-nearest neighbors (green-colored particles).

Due to the similarity between a diffusion event and a chemical reaction, the rate of a diffusion event is modeled by the Arrhenius Equation. The probability of an individual atom overcoming the energy barrier to migrate to a neighboring location is in the form of a Boltzmann distribution.

$$\hat{r}_{dif} = A_m exp\left(\frac{-\Delta E - Q}{k_B T}\right)\left[\frac{event}{site * unit\ time}\right] \qquad \textbf{Equation 2}$$

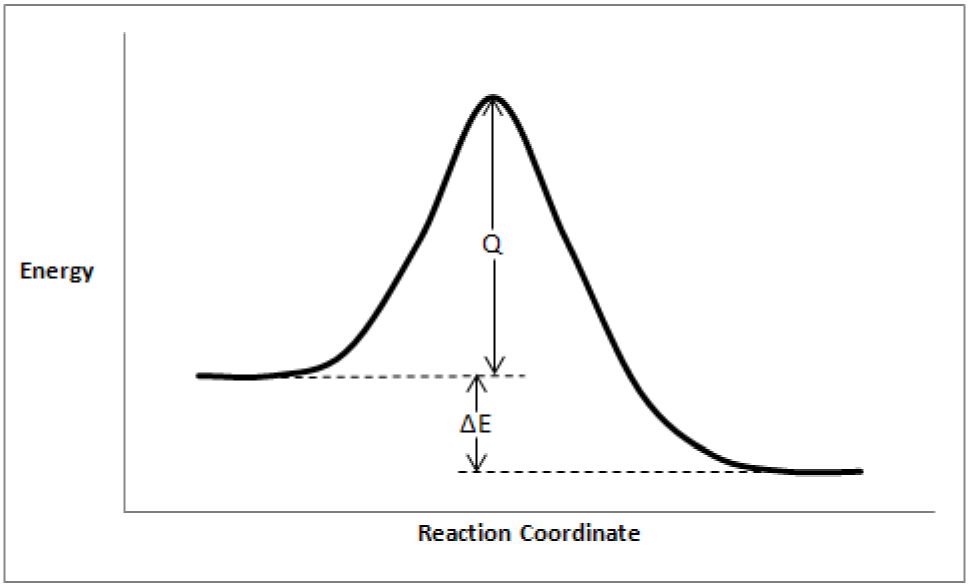


Figure 7: Energy Level Diagram for a Diffusion Event with a Negative Change in Energy. Note the similarity to the energy level diagram of a chemical reaction.

In Equation 2, Q is the barrier energy to diffusion which must be overcome for all diffusion events. ΔE is the change in energy of the system that would occur as a result of a diffusion event. The energy of the system is determined by summing the energy of each atom in the system over the entire system. The energy of each atom is specified by its coordination number, which is the number of neighboring atoms to which it is bonded. The greater the coordination number of an atom, the more bonds the atom has, and the greater its binding energy will be. If ΔE is negative (the diffusion is favorable), ΔE is excluded from Equation 2. This can be seen from Figure 7 by tracing the reaction coordinate in the positive x-direction; the atom needs to have enough energy to overcome the barrier Q regardless of the magnitude of the decrease in energy. A_m is a pre-exponential factor, k_B is the Boltzmann constant, and T is the temperature of the system. Clearly, the rate of diffusion is temperature dependent, increasing with greater temperature.

Different types, or classes, of diffusion events are possible. A type of diffusion event is defined by the initial and final coordination numbers of the diffusing atom. Equation 2 is applied for each type of diffusion event. Therefore, several of these rates are associated with each atom.

2.2. Algorithm

A schematic overview of the KMC algorithm used in this project is presented in Figure 8 (page 16). The first step in the KMC algorithm is choosing a general event—deposition or diffusion— to be executed. The total rate of deposition is:

$$\boldsymbol{W_{dep} = N * \hat{r}_{dep}} \qquad \textbf{Equation 3}$$

where *N* is the number of surface sites on the lattice. If performing a 2D simulation on a 1D substrate that is *M* sites long, *N* is set to *M*. If performing a 3D simulation on a 2D substrate that is *M* by *M* sites, *N* is set to M^{2}.

The total rate of diffusion is:

$$\boldsymbol{W_{dif} = \sum_{i=0}^{\#\, of\, classes\, of\, diffusion} M_i * \hat{r}_{dif}(i)} \qquad \textbf{Equation 4}$$

where M_i is the number of diffusion events in the system that belong to the *i*-th class.

A random number following a uniform distribution in the unit interval, ξ, is generated. If $0 < \xi < \left(\frac{W_{dep}}{W_{dep}+W_{dif}}\right)$, a deposition event is selected. If $\left(\frac{W_{dep}}{W_{dep}+W_{dif}}\right) < \xi < 1$, a diffusion event is selected.

This process is then repeated within the general event. For example, if diffusion is selected, weightings of each class of event are calculated, and a random number is used to select the class. The process repeats until the exact event to be executed is chosen.

After selecting and executing a process, the program computes the time, *Δt* by which to advance the simulation. Again, a random number in the unit interval, ξ, is selected.

$$\Delta t = \frac{-ln(\xi)}{W_{deposition}+W_{migration}} \qquad \textbf{Equation 5}$$

The combination of the equation for advancing time and the equations for event selection ensure that the rate of every individual event is approximately followed with stochastic deviations. This algorithm is repeated until the time meets or exceeds the time set by the user. This process ensures that the events are treated as Poisson processes.

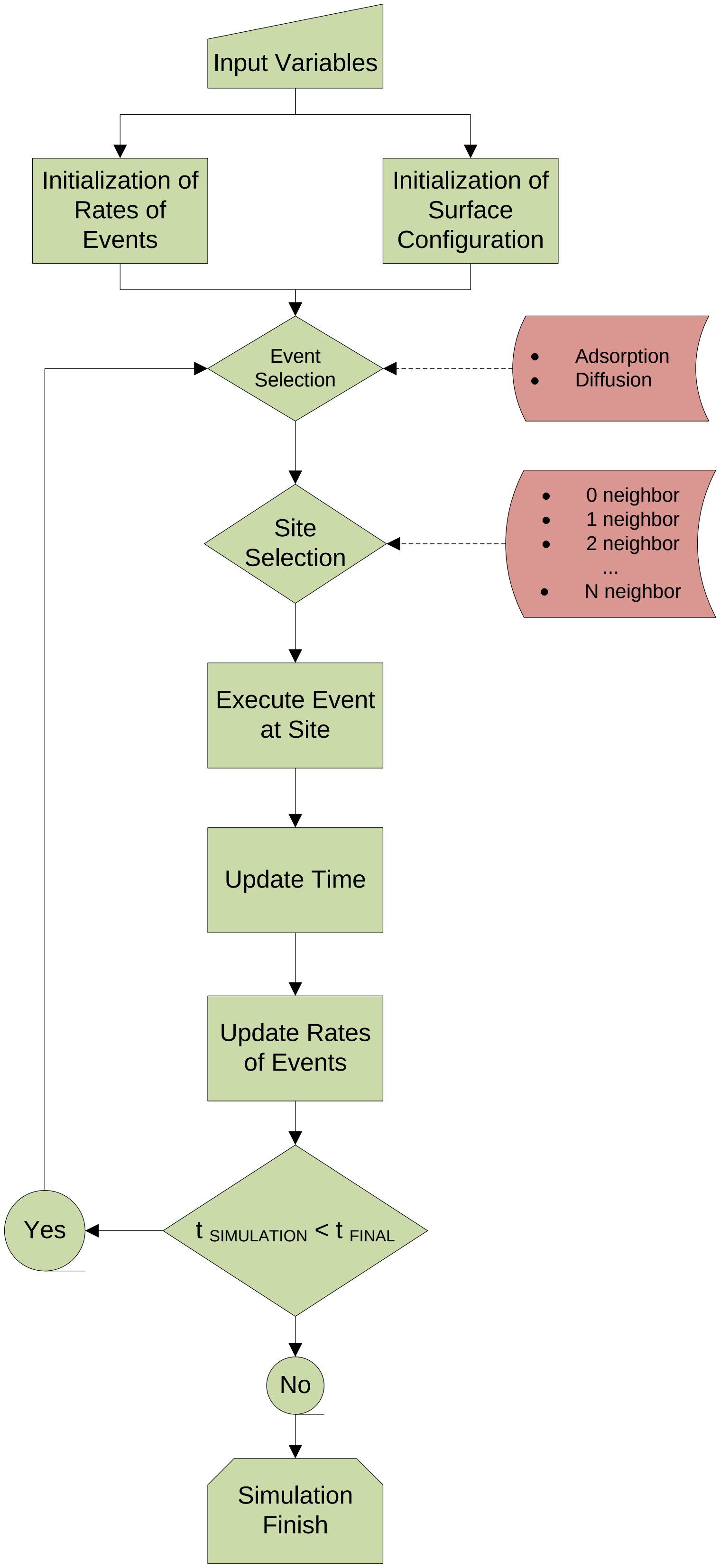


Figure 8: Schematic Representation of the KMC algorithm used in this project.

2.3. Stochastic Parallel PARticle Kinetic Simulator (SPPARKS)

SPPARKS is a standardized open-source Monte Carlo code that can be used to model various different processes. SPPARKS was designed to be easy to extend; for this reason, it can be used in two ways. First, it can be made as an executable program into which one sends an input script which specifies the run's settings and parameters. Second, it can be put into library format, which enables the user to build code around the SPPARKS engine and call it as necessary.

For every process it models, SPPARKS implements an *application*, which defines events, probabilities, and acceptance/rejection criteria. Based on spatial considerations, applications can be classified in 3 categories: *on-lattice*, which define static event sites with fixed neighbor connectivity; *off-lattice*, which define mobile event sites such as particles; and *general*, which have no spatial component.

This project uses the on-lattice application *diffusion* which performs diffusive hops on a lattice whose sites are either occupied or unoccupied (vacant). It can model diffusion on 2D or 3D lattices. Sites are assigned discrete values, called spins, which can be in one of two states, designating a site as occupied or vacant. Neighboring sites can exchange their spins, simulating a diffusion event. Although many of SPPARKS applications can be run on parallel processors, the application *diffusion* is only capable of using a single processor, which can significantly increase processing time.

3. Development of Quantitative Parameters for Control

3.1. Base Case Material

In order to explore different aspects of the control system, a base case material must be defined. This sample material was modeled on a 2-dimensional triagonal lattice (Figure 6 on page 12) using a binding model typical of semiconducting materials such as gallium arsenide (GaAs) or silicon. Values for the bond energy, diffusion barrier, and pre-exponential frequency factor, and ranges for realistic temperature and deposition rate are presented in Table 2. Binding energy was computed as a linear function of the number of bonds a particle forms. Only nearest neighbor hops were allowed in this model. The effect of adding Schwoebel hops is investigated later in section 7.1. Simulation run time was adjusted to grow a total of 30 monolayers.

Common Parameters		GaAs	Base Case Material
Single Bond Energy [eV]	E_{bond}	0.27	0.27
Diffusion Barrier [eV]	Q	1.82	1.58
Diffusion Frequency Factor	k_{des}	$5.8*10^{13}$	10^{13}
Deposition Rate [monolayer/sec]	r_{dep}		1-4
Temperature [K]	T		500-900

Table 2: GaAs and Base Case Material KMC Algorithm Parameters

3.2. Roughness

Roughness is measured as the root mean square of the surface profile. The surface profile of a system is defined by its surface particles, which are the particles that occupy the highest filled site of any vertical column. Figure 9 below shows a typical surface profile on a 2 dimensional *triagonal* lattice. The root mean square surface roughness is calculated as follows. Equation 6 is applicable for a 2-dimensional lattice, and Equation 7 is applicable for a 3-dimensional lattice.

$$\boldsymbol{\rho} = \sqrt{\frac{\sum_{i=0}^{N}\left(h_i - \bar{h}\right)^2}{N}} \qquad \textbf{Equation 6}$$

$$\boldsymbol{\rho} = \sqrt{\frac{\sum_{i=0}^{N}\sum_{j=0}^{N}\left(h_{ij} - \bar{h}\right)^2}{N \times N}} \qquad \textbf{Equation 7}$$

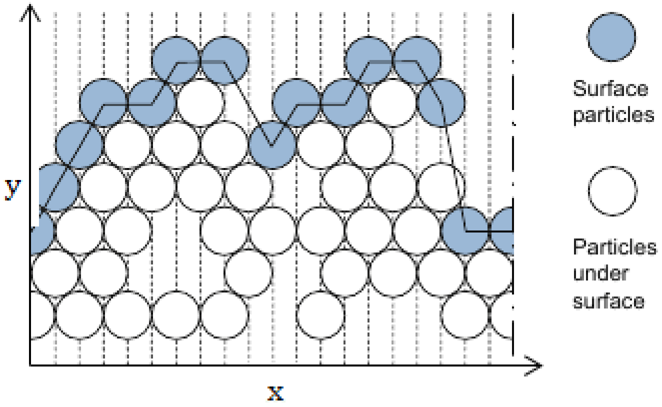


Figure 9: 2D surface profile. Gray colored particles are considered surface particles. White colored particles are all filled sites below the surface. Empty sites are not explicitly drawn.

Equations 6 and 7 measure height variations on a single particle scale. For a silicon atom, this is on the order of 100 pm. Height differences on this scale are unlikely to significantly affect film properties. A high-pass filter was developed using an alternative formula that defines contiguous vertical columns of fixed length that contain *n* number of sites in the lateral direction in 2 dimensions, and fixed surface area that contain *n x n* number of sites in 3 dimensions. The number *n* is called the "segment size." The average height of every possible column constructed in this way (local average) is computed and the roughness is defined as the standard deviation of the local average from the global average. Roughness computed in this way filters out variations on a scale smaller than the segment size. In 2 dimensions, the filtered roughness formula is as follows:

$$\boldsymbol{\rho_n} = \sqrt{\frac{\sum_{k=0}^{N-n}[(\overline{h_\imath}-\bar{h})^2]}{N-n-1}} \qquad \textbf{Equation 8}$$

where ρ_n is roughness on a scale *n*, *N* is the lattice size, *n* is the segment site, $\bar{h}$ is the global average, and $\overline{h_\imath}$ is the local average computed as follows:

$$\overline{\boldsymbol{h}_\imath} = \sum_{i=k}^{n+k+1} \frac{h_i}{n} \qquad \textbf{Equation 9}$$

where h_i is the height of the surface site at *x*-coordinate *i*. In 3 dimensions, the corresponding equations are the following:

$$\boldsymbol{\rho_n} = \sqrt{\frac{\sum_{l=0}^{N-n}\sum_{k=0}^{N-n}[(\overline{h_{\imath\jmath}}-\bar{h})^2]}{(N-n+1)^2}} \qquad \textbf{Equation 10}$$

$$\overline{\boldsymbol{h}_{\imath\jmath}} = \sum_{i=k}^{n+k-1}\sum_{j=l}^{n+l-1}\frac{h_{ij}}{n^2} \qquad \textbf{Equation 11}$$

where h_{ij} is the height of the surface site at *(x,y)*-coordinate *(i,j)*.

3.3. Porosity

Porosity (σ) is defined as the fraction of vacant sites within the film to total number of sites that are part of the film. A vacant site is part of the film if it is below a surface site. Porosity is calculated as follows:

$$\boldsymbol{\sigma = \frac{M}{N_{total}}}$$ **Equation 12**

where M is the number of vacant sites in the film, and N_{total} is the total number of sites in the film.

3.4. Objective Function

3.4.1. Definition

For the purpose of designing a control mechanism, roughness and porosity can be combined into a single parameter, constructed such that its minimum corresponds to the most desirable combination of roughness and porosity values. This single parameter is termed an objective function; the benefit of defining it is that it allows the user to easily change the importance of any given parameter simply by modifying the objective function to be minimized, without necessitating making any changes in the control algorithm. Other parameters, such as rate of growth, can easily be added to the objective function as well. An objective function defined for the base case material has the following form:

$$\boldsymbol{OF = A \times \frac{\rho}{\rho_{max}} + B \times \frac{\sigma}{\sigma_{max}}}$$ **Equation 13**

where ρ and σ are roughness and porosity, defined in Equations 8 (page 21) and 12 (page 22) respectively; ρ_{max} and σ_{max} are the maximum roughness and porosity values in a dataset; A and B are user determined weighting factors based on the relative priority of each parameter.

3.4.2. Effects of Weighting Factors

Different weighting factors can be placed on roughness and porosity in order to ensure that the objective function is capable of controlling for different types of films. In the final user-friendly version of the controller, it is expected that the user will be able to select a high, medium, or low weighting factor for each of the measured film properties. The software would then generate an optimal profile through the open-loop optimizer, and the controller would act to keep the real process on this profile. This will be further explored in Section 5.

3.5. Dimensionless Parameter to Quantify Film Properties

3.5.1. Origin and Definition

Film microstructure is directly dependent on the two surface microprocesses described earlier: diffusion and deposition. Diffusion events are a function of temperature and system configuration, and they act to minimize the energy of the system by kinetically favoring diffusive hops that lead to more extensively bonded particles. Therefore, diffusion acts to minimize surface area, reducing pores and smoothing the surface. Deposition events, on the other hand, are independent of system configuration; candidate sites for deposition are selected at random, with no regard for the effect on the system energy. Film microstructure, therefore, is determined by the relative frequencies of diffusion to deposition events.

A dimensionless parameter, γ, was defined to combine the effects of diffusion and deposition into a single parameter that dictates film properties. γ is defined as the ratio of the rate of diffusion to the rate of deposition on a per site basis. Since diffusion is dependent on system configuration, several rates of diffusion can be defined depending on the effects that a diffusive event has on the system configuration. For diffusion events which result in a negative energy change (more bonds are formed than are broken), the only obstacle to diffusion is the diffusive energy barrier. The rate of diffusion events per site is then given by:

$$\boldsymbol{r_0 = k \times \exp\left(\frac{-Q}{k_b T}\right)} \qquad \textbf{Equation 14}$$

where *Q* is the diffusive energy barrier.

Conversely, for diffusion events which result in a positive system energy change, the obstacle to diffusion is a sum of the diffusive energy barrier and the total energy of the net number of broken bonds. For a lattice which specifies *N* number of neighbors, assuming that particles are not allowed to desorb (gain a configuration number of 0) and that a particle with a full number of neighbors cannot diffuse, the number of net broken bonds can vary from 0 to *N*-1. Therefore, *N*-1 different rates of diffusion can be defined, as follows:

$$\boldsymbol{r_n = k \times \exp\left(\frac{-Q-\Delta n E_{bond}}{k_b T}\right)} \qquad \textbf{Equation 15}$$

where Δ*n* can vary from 0 to *N*-1. When Δ*n* is zero, Equation 14 reduces to Equation 13.

For simplicity, and under the assumption that the fastest rate of diffusion, *r*$_{0,}$ is dominant, γ is defined in terms of *r*$_0$ only, as follows:

$$\boldsymbol{\gamma \cong \gamma_0 = \frac{\hat{r}_{dif}}{\hat{r}_{dep}} = \frac{k \times \exp\left(\frac{-Q}{k_b T}\right)}{r_{dep}/m}} \qquad \textbf{Equation 16}$$

where *m* is the number of sites in one monolayer.

For the binding model of the base case material, r_0 is 10^3 times greater than r_1 and 10^5 times greater than r_2 at 600K, and 30 times greater than r_1 and 10^3 times greater than r_2 at 900K. Therefore, the assumption that r_0 always dominates introduces a small error in the calculation of γ at high temperatures.

3.5.2. Validation of γ as the Only Manipulated Variable

To verify that γ is the only parameter that dictates film properties, four simulations were run at two different γ values. For a given value of γ, two different combinations of temperature and deposition rate (such that they result in the same value of γ, according to Equation 16) were tested. Simulation run time was adjusted such that the same number of monolayers was deposited in each simulation. Measured film properties, roughness and porosity, are shown in Figure 10 and 11 below. The roughness and porosity profiles for the same value of γ at different temperature and deposition rate combinations coincide, demonstrating that film properties indeed depend only on γ, and not on temperature or deposition rate individually.

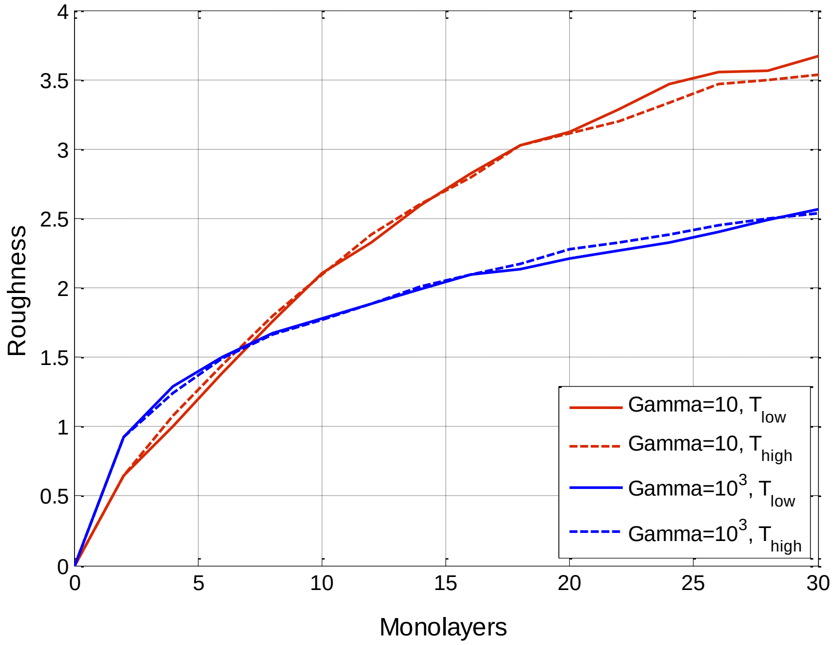


Figure 10: Comparison of Roughness between Runs of Similar γ, at Different Deposition Rates and Temperature Combinations.

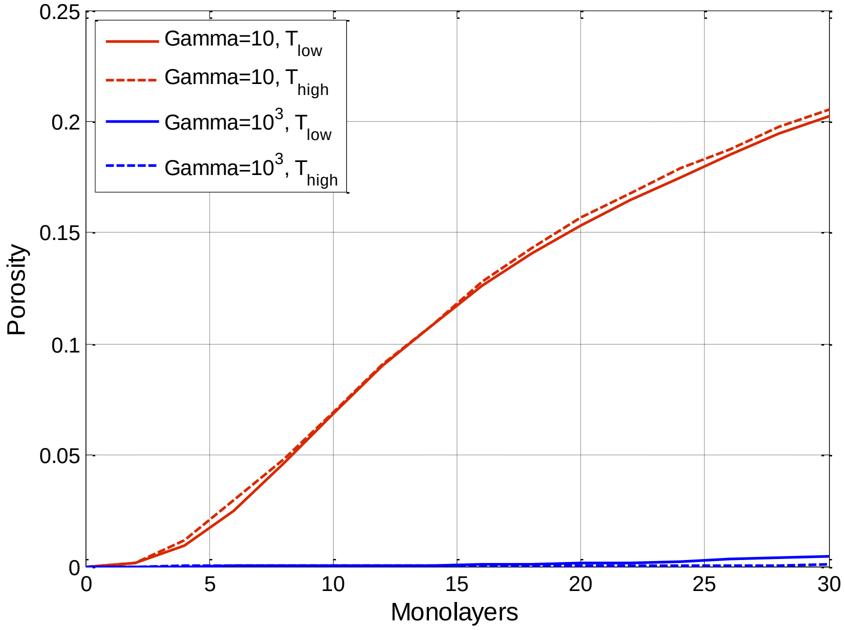


Figure 11: Comparison of Porosity between Runs of Similar γ, at Different Deposition Rate and Temperature Combinations

4. Development of a Base Case Material

4.1 System Size Selection

It is important that the controller has the ability to simulate events quickly in order to not introduce too much lag time into the control process. The computational demands of KMC can be large if many calculations need to be performed. Several KMC solvers are available within SPPARKS which enable the user to minimize computational time; the linear solver was selected, which chooses an event by scanning the list of events in a linear fashion. Hence the time cost to pick an event scales as a function of the number of events. The number of events scales linearly with lattice size, where lattice size is defined as the number of lattice sites available for each monolayer of the simulation. It was found that the function that relates computational time to the lattice size is approximately quadratic. Table 3 below shows approximate computational times for runs of constant γ on different lattice sizes. Figure 12 shows the quadratic curve fitting. This experiment was performed with the base case material: two-dimensional growth on a one-dimensional substrate.

Lattice Size	Computational Time [seconds]
100	165
250	657
500	2,470
1000	6,650
2000	26,100

Table 3: Computational Time as a function of Lattice Size. All experiments were run as two-dimensional film growth on a one-dimensional lattice. γ was kept constant as 10^5, which corresponds to a temperature of 731 K.

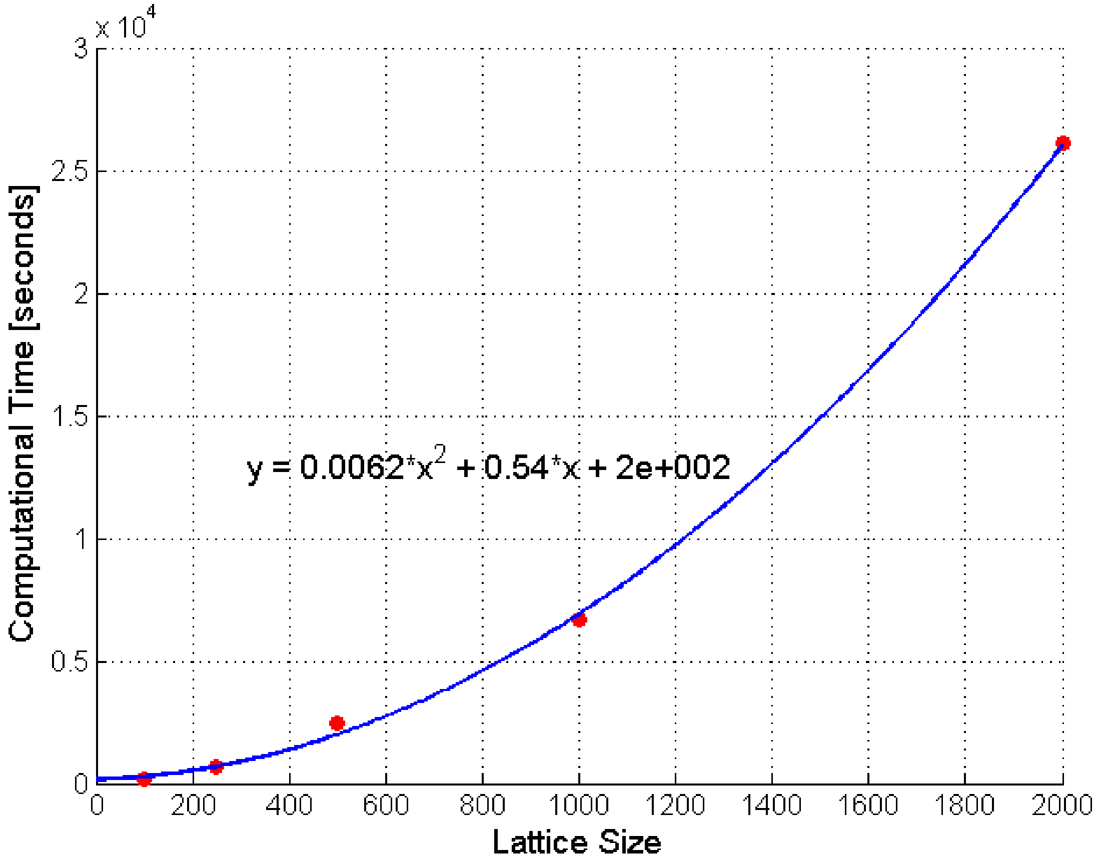
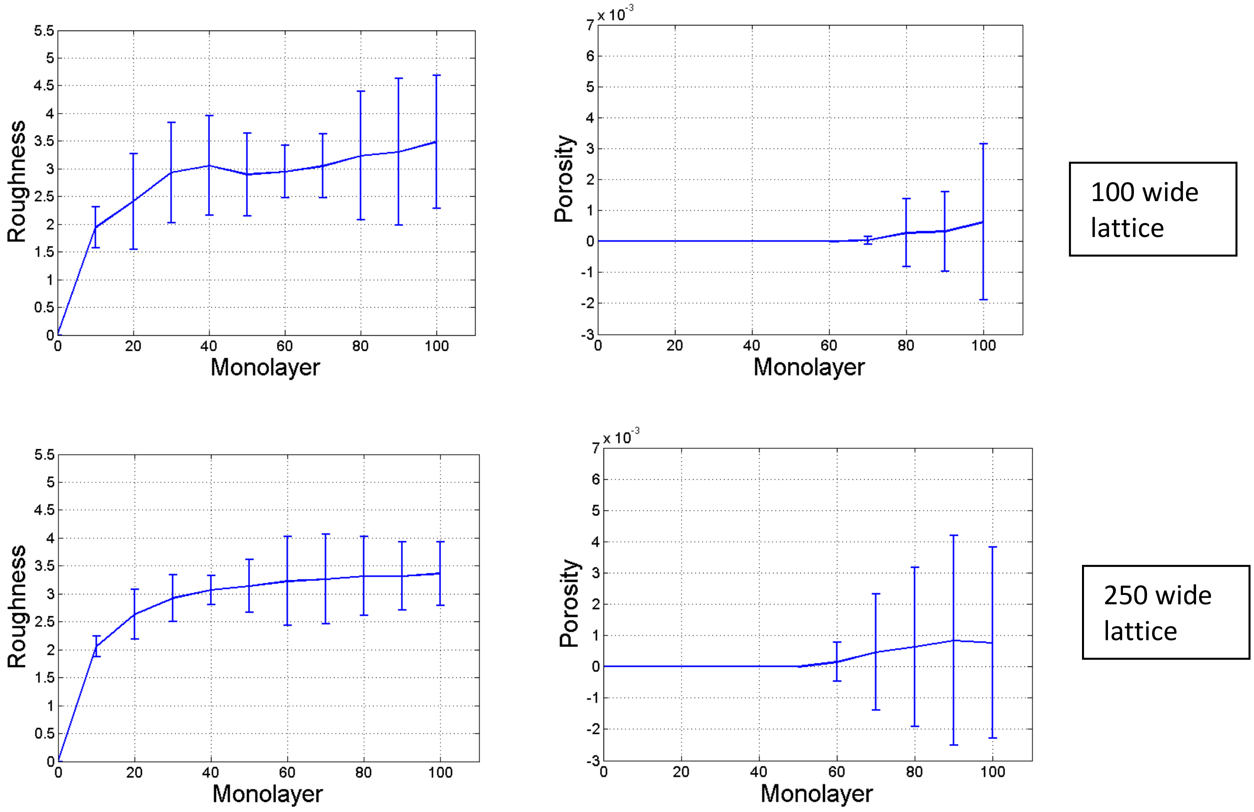


Figure 12: Plot of the data from Table 3. Computational time is a quadratic function of lattice size.

Though decreasing the lattice size results in a faster simulation, it also increases the noise inherent in a stochastic simulation. Although it is expected that no two simulations with all parameters exactly the same will give results that are exactly the same, it is crucial that the simulation converges to the same film properties that would be seen in the actual process. Thus, an important early step in the development of a controller is determining an optimal lattice size to use for the simulation, with the goal of using as small a lattice size as possible without distorting the results of the simulation. To determine this optimal lattice size, four simulations were run, each on lattice sizes ranging from 100 to 2000, keeping γ constant throughout. These simulations were run for 100 monolayers: longer than only the first 30 monolayers of growth in order to evaluate how film properties will change with increased time. Figure 13 below shows that the variance of roughness decreases with increasing lattice size as expected, with the exception of the 1000 wide lattice. The odd result on the 1000 wide lattice is probably a result of four runs not being sufficient to completely eliminate the noise from this experiment. The variance of porosity behaves oddly, which is due to the fact that these experiments were run at relatively high γ, so little porosity was noticeable early on in the experiment. Small changes in porosity when the pores began to develop have exaggerated effects. A lattice size of 1000 was chosen for the rigorous thin film growth model (the simulated real process) as a good balance between computational time and variability.



(figure continued on next page)

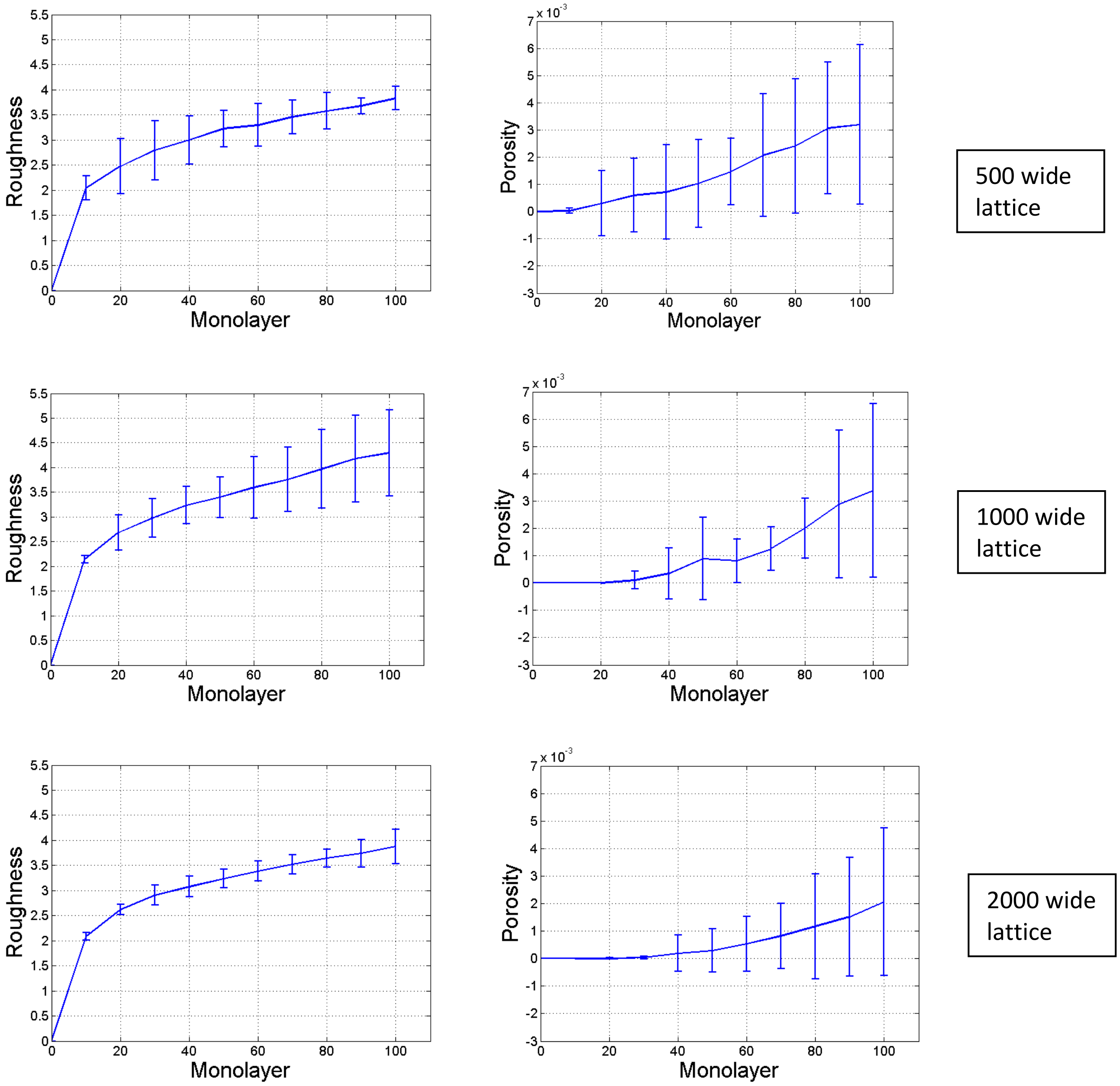


Figure 13: Roughness and Porosity vs. Monolayer for lattice sizes of 100, 250, 500, 1000, and 2000. All experiments were run on a tri lattice, with constant γ of 10^5 and temperature of 730 K. Error bars shown are two standard deviations above and below the mean for each timestep. Four runs were done with each lattice size.

Due to the fact that computational time does not scale linearly with lattice size, it would be more efficient to average the results of several small-lattice runs rather than attempt one large-lattice run. It would take as much time to do forty runs on a lattice size of 100 as it would take to do one run on a lattice size of 1000. Figure 14 below shows that over the first 30 monolayers, averaging four 100 wide lattice runs is sufficient to stay within the 95% confidence interval of the low-variance 2000 wide lattice runs (at 20 monolayers, it is slightly out of range). For future work in which more than 30 monolayers are grown, more averaging runs will be needed. The controller averaged five 100 wide lattice runs for extra safety at the cost of some computational time.

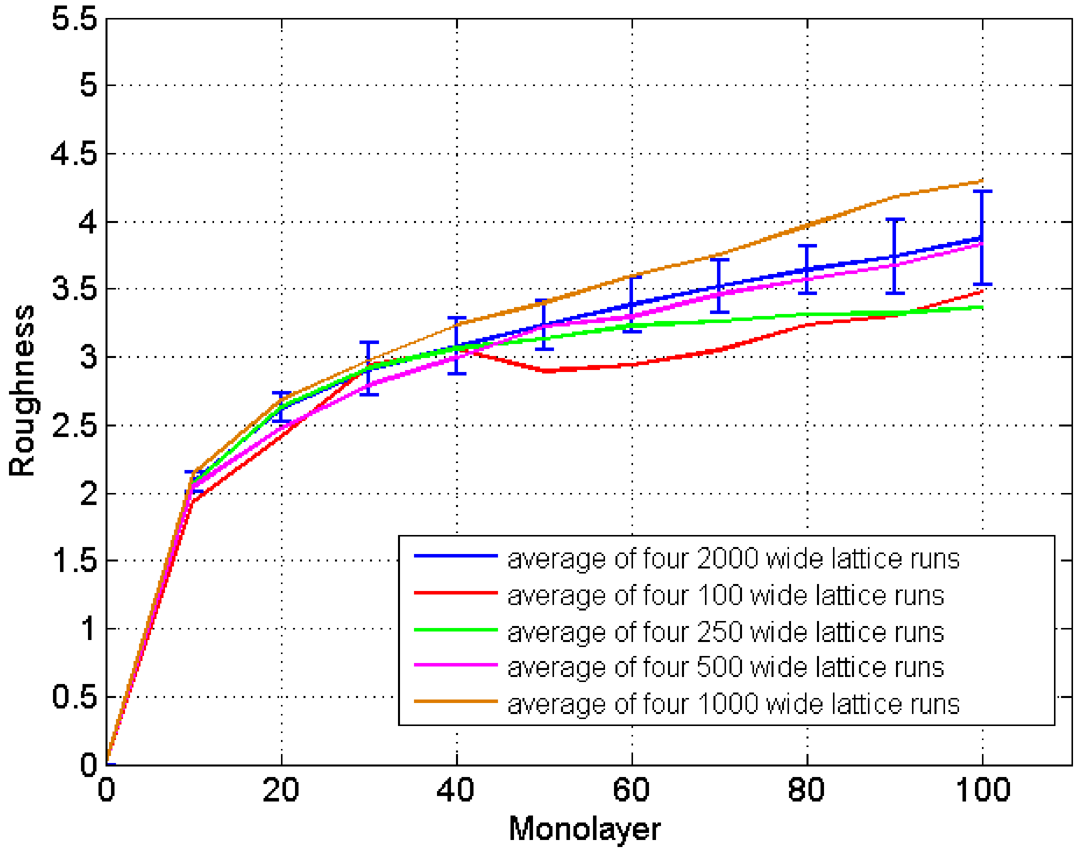


Figure 14: Roughness vs. Monolayer for lattice sizes of 100, 250, 500, 1000, and 2000. Within the first 30 monolayers, averaging four 100 wide lattice runs is sufficient as a substitute for one 2000 wide lattice run.

4.2 Optimal Steady-State Profile

4.2.1 Material Behavior

The base case material was characterized by examining its behavior after 30 monolayers of growth in terms of roughness and porosity at different values of γ. Figure 15 shows film images at five different γ values, and Figures 16 and 17 show corresponding time evolutions of porosity and roughness, respectively.

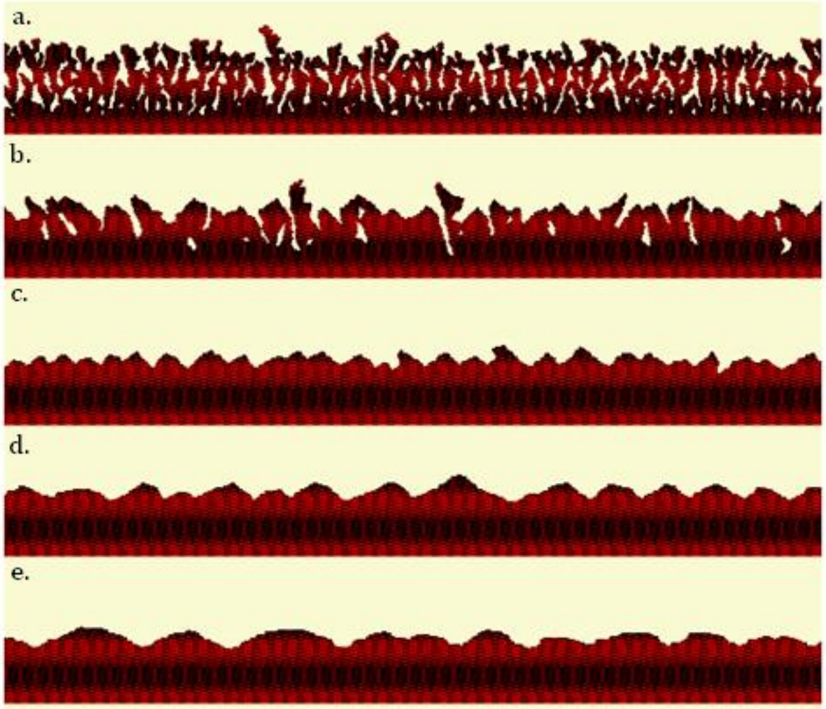


Figure 15: Film Images. a. γ=10; b. $\gamma=10^2$; c. $\gamma=10^3$; d. $\gamma=10^4$; e. $\gamma=10^5$. 30 monolayers deposited.

As expected, porosity decreases with increasing γ. Roughness, however, shows an interesting behavior. At low values of γ, it is high as expected, and decreasing with increasing γ. However, it reaches a minimum at $\gamma=10^3$, and proceeds to rise as γ increases further. Examining the film images, it can be seen that when γ is greater than 10^3, the surface profile exhibits a sinusoidal behavior, and the wavelength and amplitude of the roughness waves increases with γ.

As discussed in Section 3.2, surface roughness is computed such that it ignores roughness on a small scale, defined by the segment size specified. Since the amplitude of the wavelength increases with γ, so does the computed roughness. The base case material, therefore, has a minimum computed roughness at an intermediate value of γ.

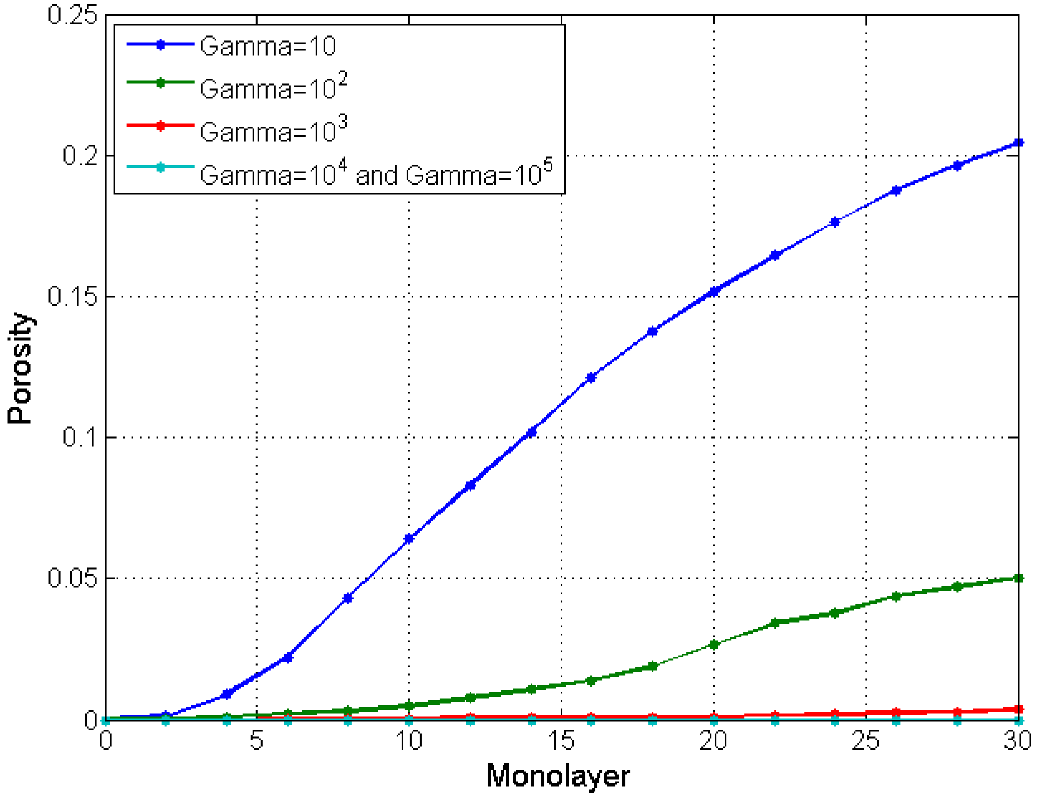


Figure 16: Time Evolution of Porosity. Porosity decreases with increasing γ.

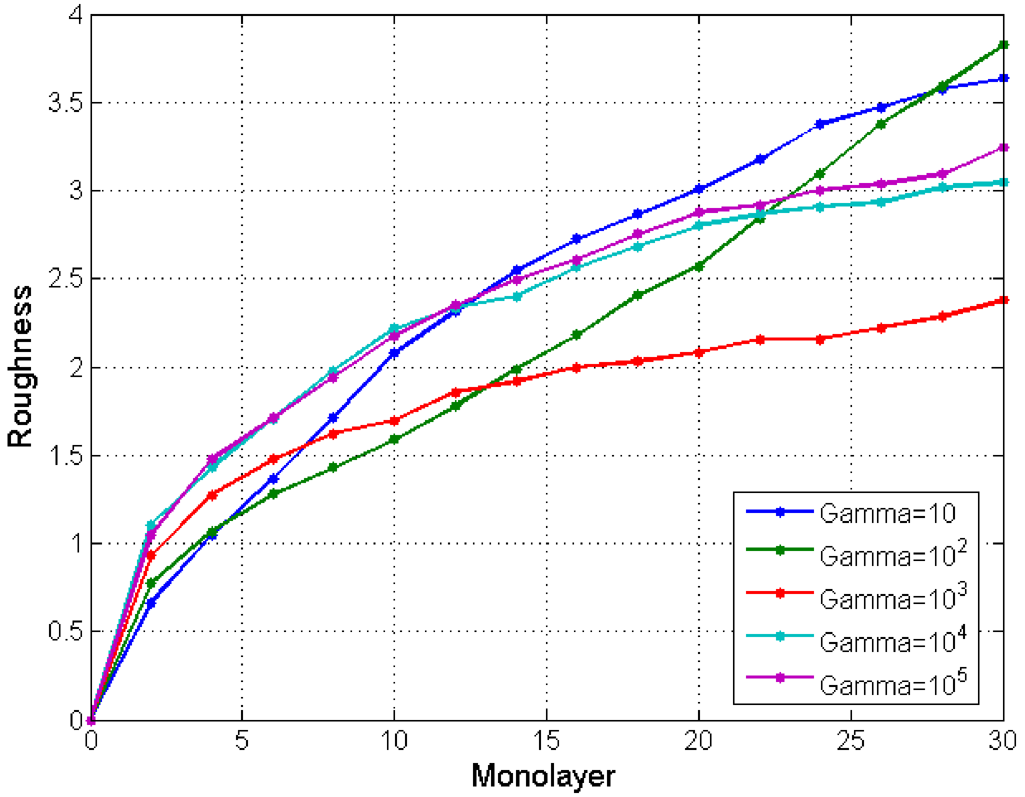


Figure 17: Time Evolution of Roughness. Roughness reaches a minimum at $\gamma=10^3$ and proceeds to increase as γ increases further

4.4.2. Effects of Weighting Factors on the Optimal Steady-State Profile

Experiments were conducted using two different weighting factor combinations to see how they would affect the optimal steady-state profile: one in which the weighting factors were 94% porosity and 6% roughness—Weighting 1 (high porosity, low roughness)—and one in which the weighting factors were 0.04% porosity and 99.96% roughness (low porosity, high roughness)—Weighting 2. Weighting 1 was used to indicate to the controller that porosity must be minimized (minimizing roughness need only be considered when the choice of two different γ values would result in the same level of porosity), while Weighting 2 signals that roughness is the more critical value to minimize. It is expected that the optimal profile under Weighting 1 would be rougher and less porous than under Weighting 2.

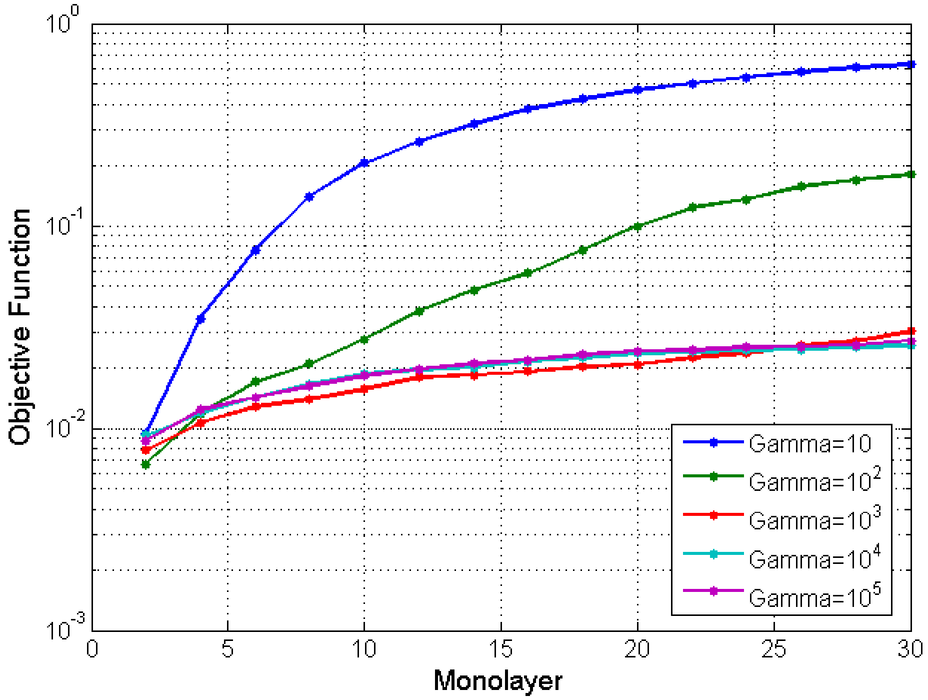


Figure 18: Time evolution of Objective Function with Weighting 1 (94% porosity, 6% roughness) at different γ values. The optimal film forms at $\gamma=10^4$.

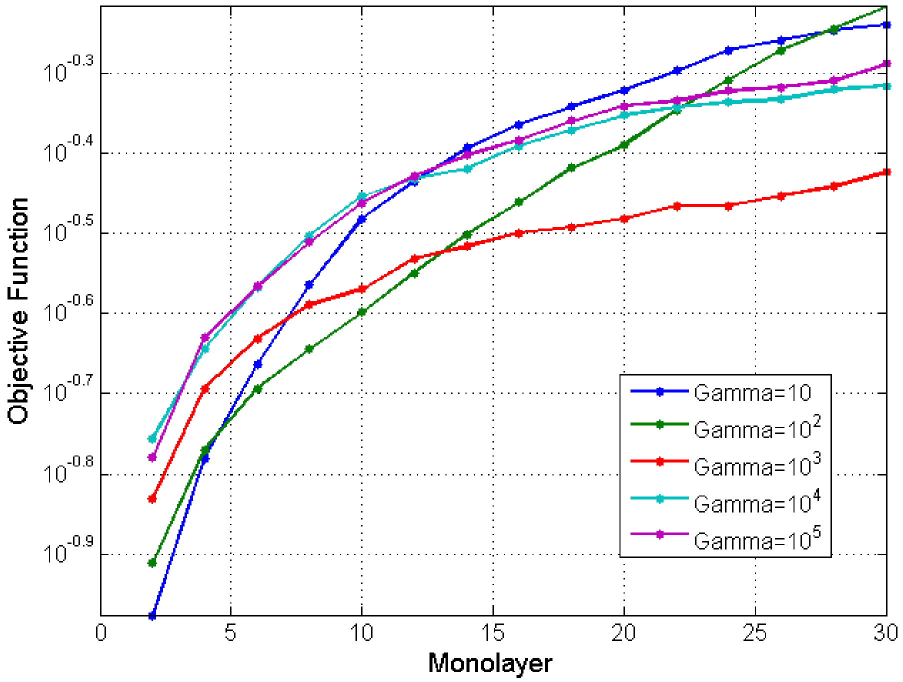


Figure 19: Time evolution of Objective Function with Weighting 2 (0.04% porosity, 99.96% roughness) at different γ values. The optimal film forms at γ=10^3.

As can be seen from Figure 18 and Figure 19 above, changing the weighting factors on the objective function changes which value of γ is the optimal to run at steady state. With Weighting 1, γ=10^4 is best, as this film is tied with γ=10^5 for the lowest porosity, and it has a lower roughness than γ=10^6. With Weighting 2, γ=10^3 is best, as this film has the lowest roughness. To see what these different films look like, see Figure 15 (page 32). Thus, these weighting factors can be used to control for the film properties of interest. This will be expanded upon in section 5.2.5 shortly.

5. Model Predictive Control: Open Loop Optimizer

5.1 Introduction to Model Predictive Control

Due to difficulties in measuring roughness, porosity, and other parameters of thin films in real time during the film-growth process, control systems based on modeling thin film growth have been shown much interest. Model Predictive Control (MPC), which was first explored by industrial engineers in 1978, is one such control system that has attracted attention. The block-flow diagram for MPC is shown in Figure 20 below. In an industrial setting, a rigorous simulation of a deposition process is run in parallel with an actual deposition process. However, for the purposes of this study, no actual deposition process is run. From this simulation, important film properties can be measured. A simulation designated as the "model" is considered to be a less rigorous representation of the real process and is used to predict the future morphology of the system. At a given starting point, film properties are taken from the rigorous simulation and fed into the model process. The model process simulates a time step, known as the predictive step, at the current temperature and compares the film properties to the properties in a reference trajectory, which is fed to the controller before the run. If the properties are within an allowable error of each other, the rigorous simulation and the real process moves forward a time step, known as an action step. If, however, the properties are not within the allowable error of each other, the model process simulates the predictive step at several different temperatures to see which temperature gives the lowest error between the simulated film properties and the optimal profile properties. This "optimal temperature" is input to the rigorous simulation and to the real process, which moves forward the action step.

Two points are worth noting. First, the objective function is used in place of multiple film parameters in order to simplify the problem from a Single Input Multiple Output (SIMO) problem to a Single Input Single Output (SISO) problem. Second, although this algorithm explicitly changes temperature while keeping the deposition rate constant, it is implicitly changing γ. As explained in section 3.5.2, this single variable is valid as the sole manipulated variable. This will be explored further shortly.

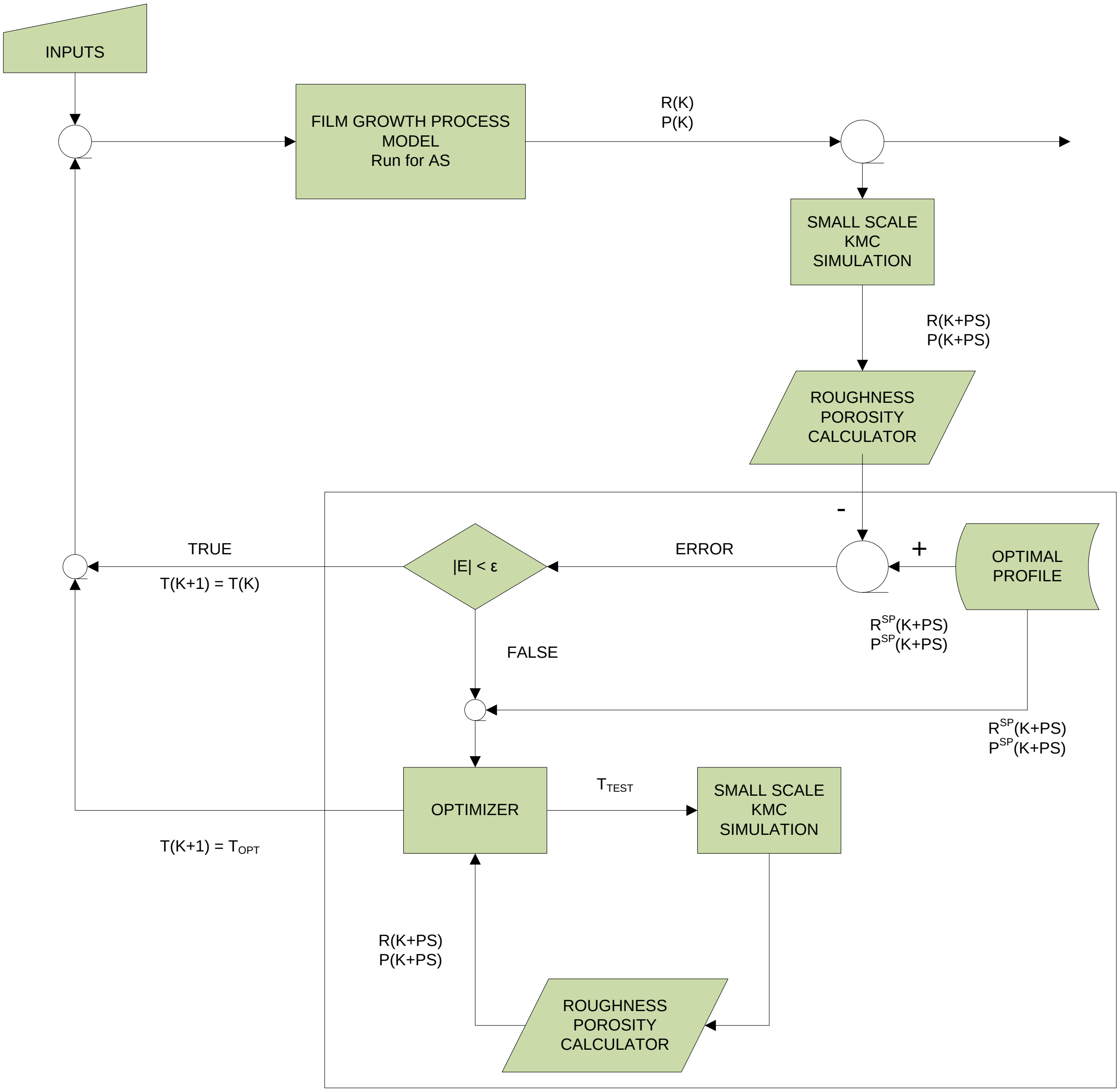


Figure 20: Model Predictive Controller Algorithm: *AS* represents the action step, *PS* is the predictive step, *K* is the current state of the real process, *sp* represent set point properties taken from the optimal profile, *T* is the process temperature.

The presence of the reference trajectory is intended to decrease the computational time necessary for the controller, thereby decreasing the controller's dead time. Using a reference trajectory, the controller need not solve the receding horizon optimization problem of minimizing the difference between the final surface properties and the desired values; the controller only needs to solve the fixed short-horizon optimization problem of minimizing the difference between the instantaneous surface properties and the reference values. Any feasible reference trajectory can be used, but if the controller's intent is to guide the film towards optimal properties, an optimal profile trajectory must first be found. The process of finding an optimal profile, which is computationally demanding, is usually done off-line.

5.2 Open-loop Optimizer

5.2.1. Algorithm

The algorithm for finding an optimal profile is an ideal open-loop deposition (it is assumed to be the result of an undisturbed process in which there is no set point comparison made to create a control action). Essentially, it is the MPC algorithm with an adjustment: no optimal profile is provided; instead, the process always enters the control loop, and the optimal temperature is the one which absolutely minimizes the objective function. This algorithm is presented in block-flow diagram form in Figure 21 below. The open-loop optimization algorithm begins with a flat substrate, with the optimal steady state temperature set as the initial temperature. It was postulated that this is a good first approximation of an optimal initial temperature.

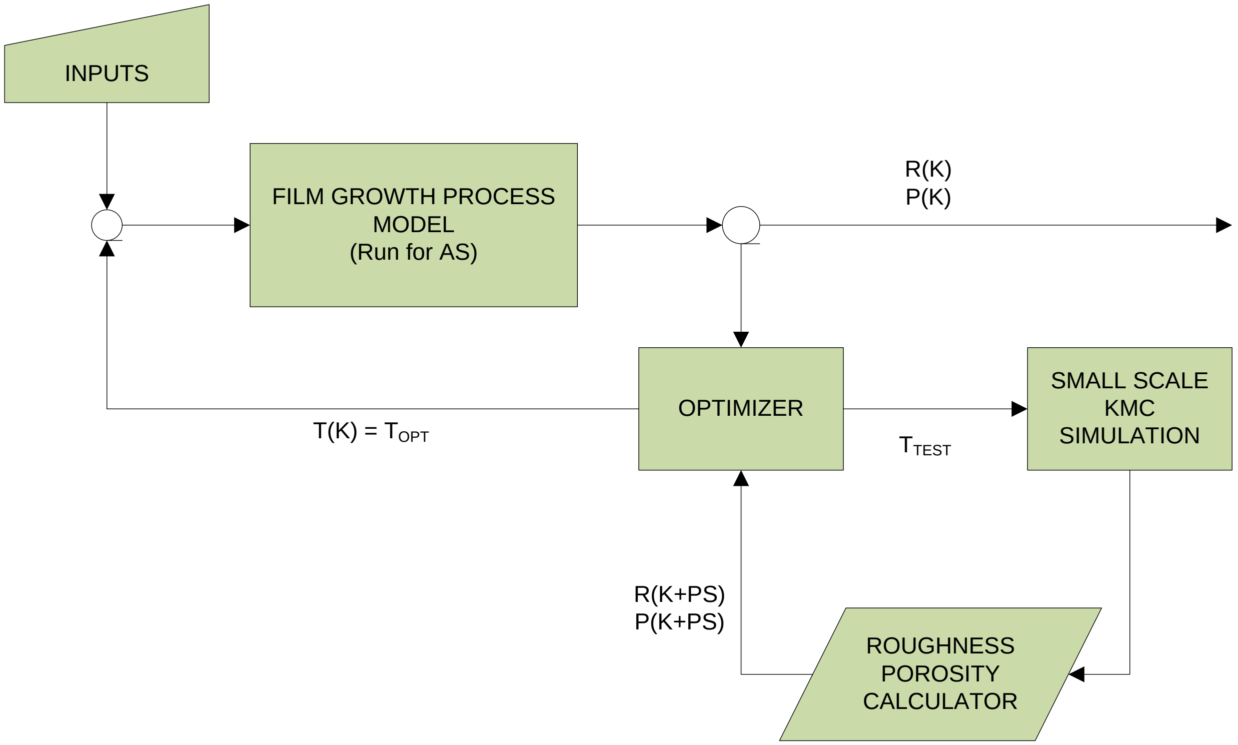

INPUTS
FILM GROWTH PROCESS MODEL (Run for AS)
R(K)
P(K)
OPTIMIZER
SMALL SCALE KMC SIMULATION
T(K) = T_{OPT}
T_{TEST}
R(K+PS)
P(K+PS)
ROUGHNESS POROSITY CALCULATOR

Figure 21: Open-loop Optimizer. K is the current state of the process, R is surface roughness, P is porosity, PS is the predictive step, AS is the action step, T is temperature.

Similarly to the algorithm for MPC, two simulations are executed over the course of the open-loop optimizer. The simulation intended to model the real process is considered to be a rigorous representation of the real deposition process, while the simulation designated as the "model" is considered to be a less rigorous model representation of the real process and is used to predict the future morphology of the system. The two simulations differ by the simulation system size, which is measured in terms of substrate sites on the lattice. The ratio of the size of the rigorous simulation to the size of the model simulation is specified by the user. As discussed in section 4.1 on page 27, many smaller simulations can be executed in the time taken to execute one larger simulation; thus, a small system is used for the model simulation and several runs are averaged. The deposition rate for the model simulation is scaled to give the same monolayer growth rate as the real process simulation according to Equation 17 below.

$$\frac{N_R}{N_M} = \frac{d_R}{d_M}$$ **Equation 17**

where *N* is the number of sites in a monolayer, *d* is the deposition rate onto the substrate, *R* denotes the rigorous simulation properties, and *M* denotes the model simulation properties.

A method for approximating the morphology of the model simulation from the rigorous simulation was formulated; this approximation takes place during the initialization of the model simulations before the execution of the predictive step. Figure 22 displays the procedure for the approximation of the atomic arrangement of the model simulation, which is achieved by cropping a fraction of the atomic arrangement from the real process simulation. The morphology is always cropped from the (0,0) or (0,0,0) coordinate of the rigorous process's atomic arrangement to the desired size. After the cropping procedure, the model simulations are executed for the predictive step at the scaled deposition rate given by Equation 17 above.

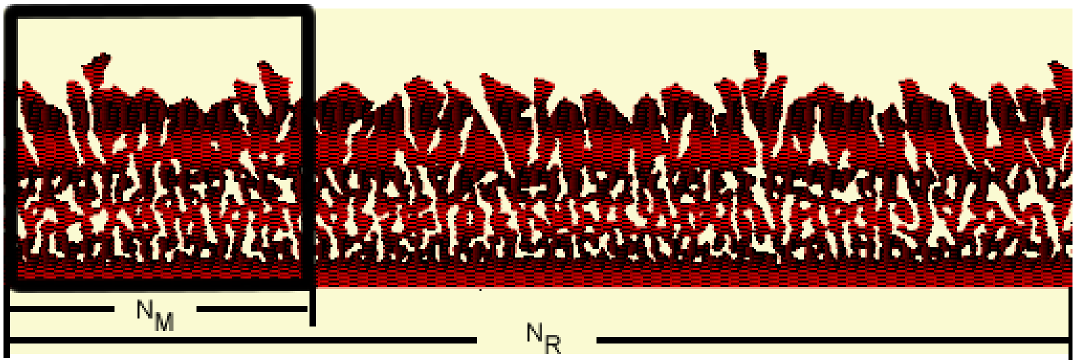


Figure 22:Approximation of model simulation atomic arrangement from the real process. R represents the atomic arrangement of the rigorous simulation resulting from the previous Action Step. M represents the approximation of the atomic arrangement that is used to initiate the model simulation in the predictive step.

For each optimization cycle, model simulations are executed for a predictive step at various sample temperatures within a specified temperature range. This temperature range represents the maximum temperature deviation from the previous temperature that the reactor can physically change. The number of temperatures tested within this range is specified by the user; they are chosen at equal intervals within the range allowed. Multiple runs are done at each temperature and averaged to reduce noise. The number of runs that get averaged is also specified by the user.

5.2.2. User-Specified Parameters

For the following experiments, except where otherwise specified, the user-specified parameters were set to the values presented in Table 4. The deposition rates and temperature ramp rates that were used in the open-loop optimizations are typical to thin film deposition processes. The time to deposit a full monolayer of atoms equates to 1 second, resulting in a 1 atom/(second*sites) deposition rate. This is a typical deposition rate of many deposition processes. The temperature was allowed to change a maximum of 20˚C per monolayer deposited for all open-loop optimizer experiments. This results in a maximum temperature ramp rate of 20˚C/s. Different reactors have different maximum temperature rates, running from 5˚C/s to rates up to 2000˚C/s; thus, the specified maximum temperature ramp rate was considered to be acceptable. The amount of model simulations to average and the size of the model simulations were decided based on lattice size considerations, which are discussed in section 4.1. The number of temperatures within the range tested is a significant parameter with regards to the time require for the optimization. Seven trials allow an average open-loop optimizer to run for 15 hours to deposit 30 monolayers. The choice for the weightings on roughness and porosity in the objective function are discussed in the objective function section. These weightings emphasize the importance of the reduction of porosity over surface roughness. All parameters are shown in Table 4 below. The choices for the length of time of the action step and predictive step are explored in Section 5.2.3 below.

Parameter	Specification
Predictive Step	2 monolayer
Action Step	1 monolayers
Rigorous Size : Model Size	1000:100
No. Model Simulations to Average	5
Temperature Range [˚C]	20
No. Temperature Trials	7
Roughness Weighting	0.06
Porosity Weighting	0.94

Table 4: Open-Loop Optimizer Parameters

5.2.3. Experiments on the Predictive Step and the Action Step

The length of time of the predictive step and the action step are also user-specified. The predictive step must be long enough for the film to be able to experience changes in morphology due to the change in temperature before the predictive step simulation. However, it must not be so long so as to prevent the controller from “realizing” that it has the capacity to change the temperature within smaller increments of time. It is hypothesized that the action step should be as short as possible within no limit other than computational time; feasibly, the action step could be a single deposition or diffusion event. After this step, the controller would re-enter the loop and calculate a new optimal temperature running forward another predictive step. However, at some point the decrease in action step should have little value and can be very computationally expensive.

Experiments were designed to obtain a proper combination of predictive step and action step execution that yields the best optimal profile that represents the most optimal evolution of thin film morphology. To characterize these step parameters, Table 5 summarizes experiments of predictive step and action step combinations that were done on the base case material using the open-loop optimizer. Five prediction and action step combinations were used in the experiments. These combinations were designed to examine the effects that predictive step had on the optimization holding the action step constant and action step holding the predictive step constant. To reduce noise, the open-loop optimizer was run on each experimental combination three times.

Predictive Step	Action Step
1 Monolayer	1 Monolayer
2 Monolayers	1 Monolayer
3 Monolayers	1 Monolayer
3 Monolayers	2 Monolayers
3 Monolayers	3 Monolayers

Table 5: Open-loop Optimizer Experiments

In the step combination experiments, the open-loop optimizer produced different optimal transient profiles for the various prediction step (PS) and action step (AS) combinations, which demonstrates that these parameters play an important role in the generation of the optimal profile. Figure 23 shows the resulting optimal profiles of objective function for each step combination experiment. Figure 24 summarizes the control actions on γ. Figure 26 and Figure 27 are important with regards to deciding on a PS and AS combination that is to be used in later experiments.

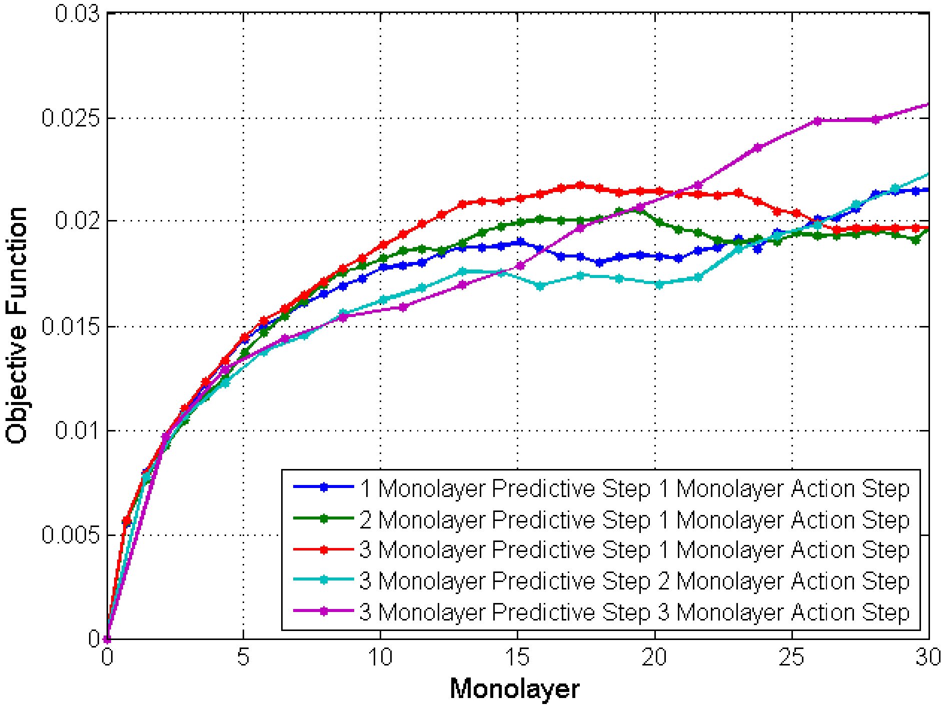


Figure 23: Optimal Transient Profile Comparison of Various Predictive and Action step Experiments. Each optimal profile is constucted from 3 open loop optimizer runs on the same step combination experiment.

In Figure 23, the optimizer demonstrates similar optimal levels of evolution for all of the combination steps during the first 5 monolayers (ML) of deposition, as indicated by a 5% standard deviation from the average of all the step combination objective functions at the 5 ML as seen in Figure 25 (page 47). This is an important result, since it indicates that for very thin film applications, on the order of 1 nanometer, more advanced film deposition control may be needed. From Figure 24, the 3 PS and 3 AS combination has the largest increase in γ during this early stage in the process. However, the objective function does not change relative to other step combinations. This indicates that at early stages of the film deposition process, the film growth will have negligible benefit from changes in γ.

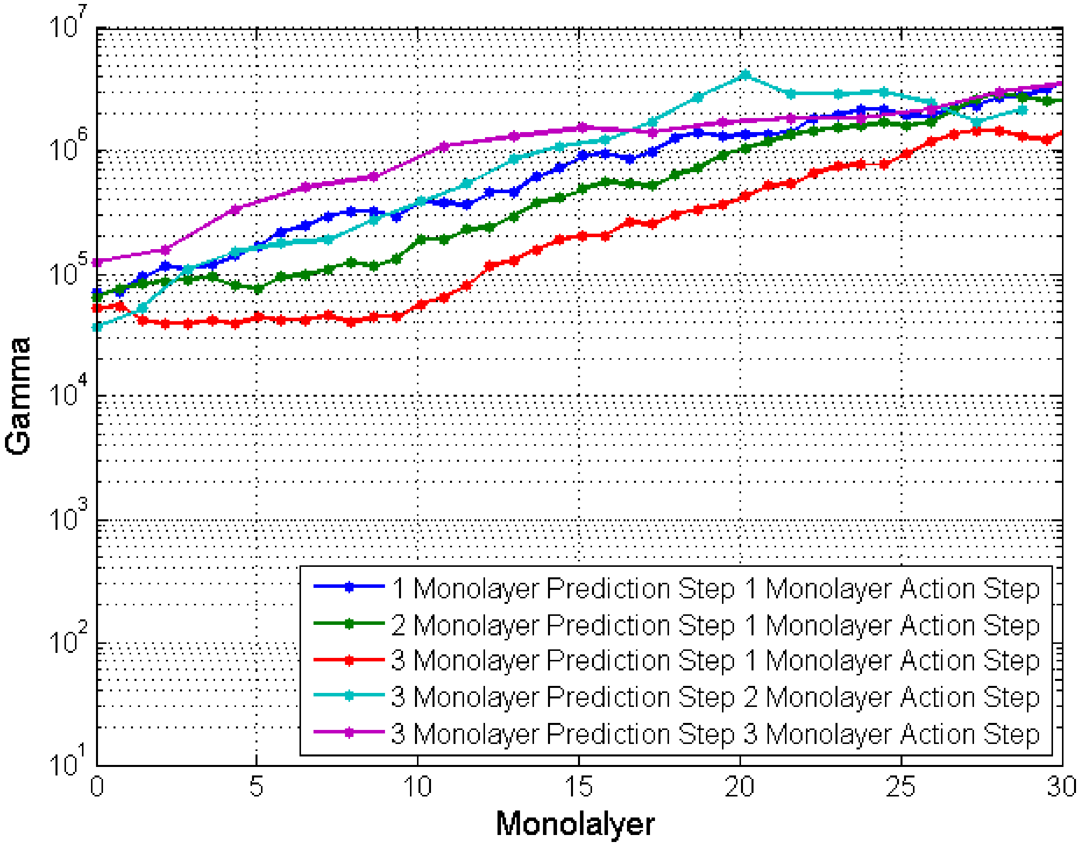


Figure 24: Transient γ Profile Comparison of Various Predictive and Action step Experiments. Each γ profile is constructed from 3 open loop optimizer runs on the same step combination experiment

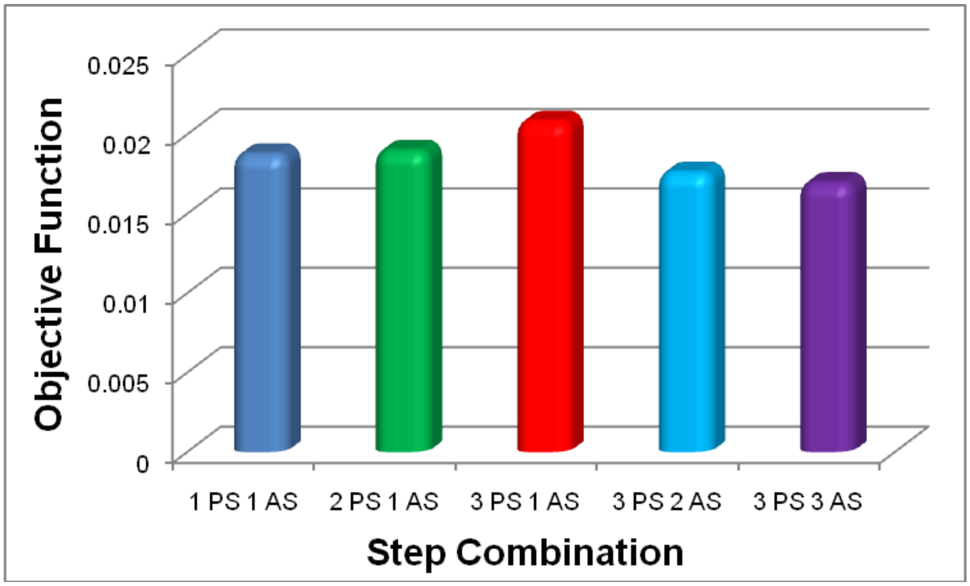


Figure 25: Objective Function of Step Combination Experiments After 5 ML

By observing the experiments in which the PS was kept constant while the AS varied, it becomes apparent that lowering the AS always has beneficial effects on the optimal profile, as expected. Isolating the experiments in which the AS was kept constant while the PS varied shows that varying the PS has unknown effects, also as expected. The best optimal profile was found with 2 PS and 1 AS. This profile was used as the reference trajectory for the MPC runs, which will be explored in Chapter 6. Images of the films after 30 monolayers of growth are shown below in Figure 27. The roughness profiles and porosity profiles are shown in Figures 28 and 29. It appears that a final steady-state value for roughness has been approached for the 2 PS and 1 AS combination, but porosity is still increasing.

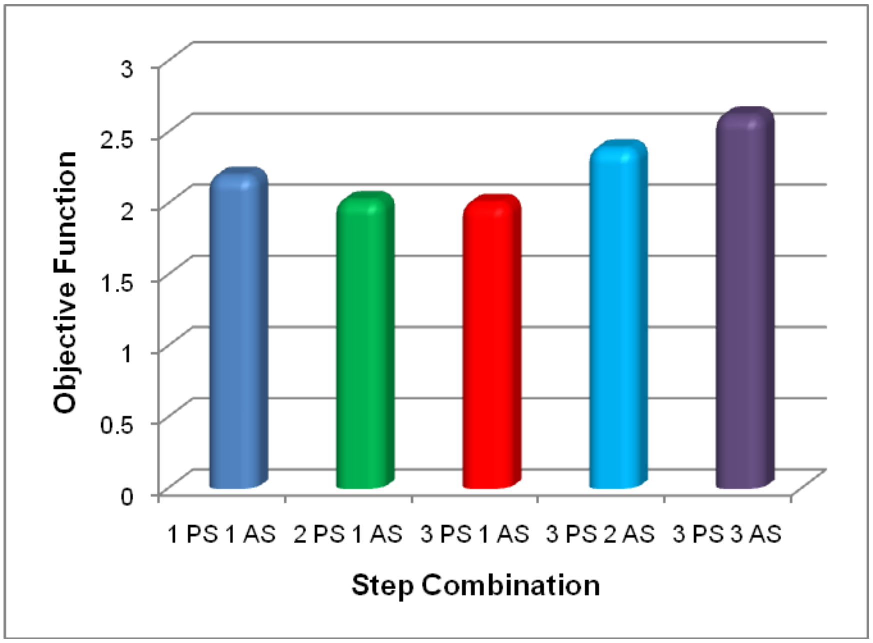


Figure 26: Final Objective Function Values Comparison. Various step combinations after 30 monolayers of simulated growth

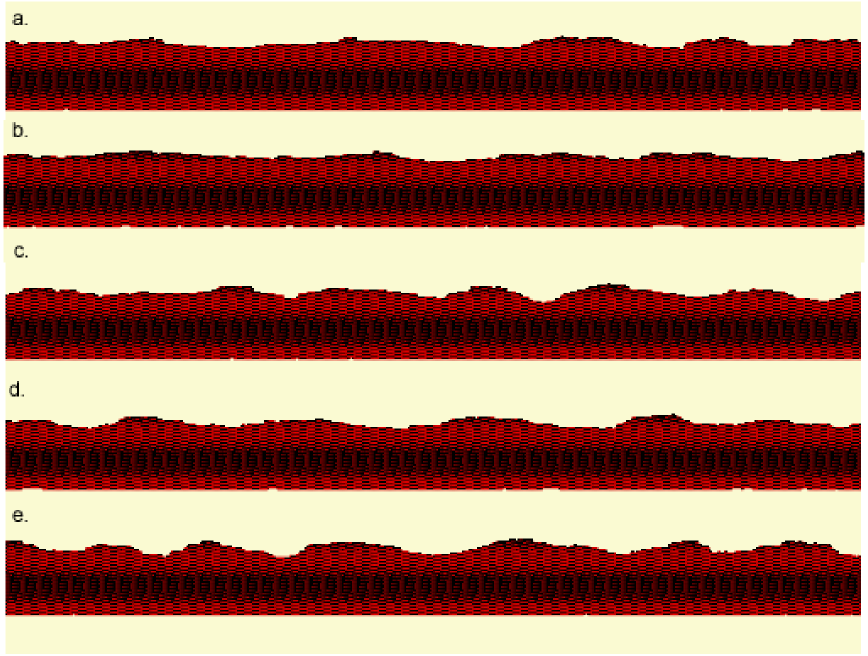


Figure 27: Step Combination Experiments Open-Loop Optimizer. Comparison of films resulting from a) 1ML PS 1ML AS, b)2ML PS 1ML AS, c) 3ML PS 1ML AS, d) 3ML PS 2ML AS, e) 3ML PS

3ML AS. The 2ML PS 1ML AS results in the smoothest film. Due to film morphology considerations the 2ML PS 1ML AS combination was used for subsequent optimizations in this report.

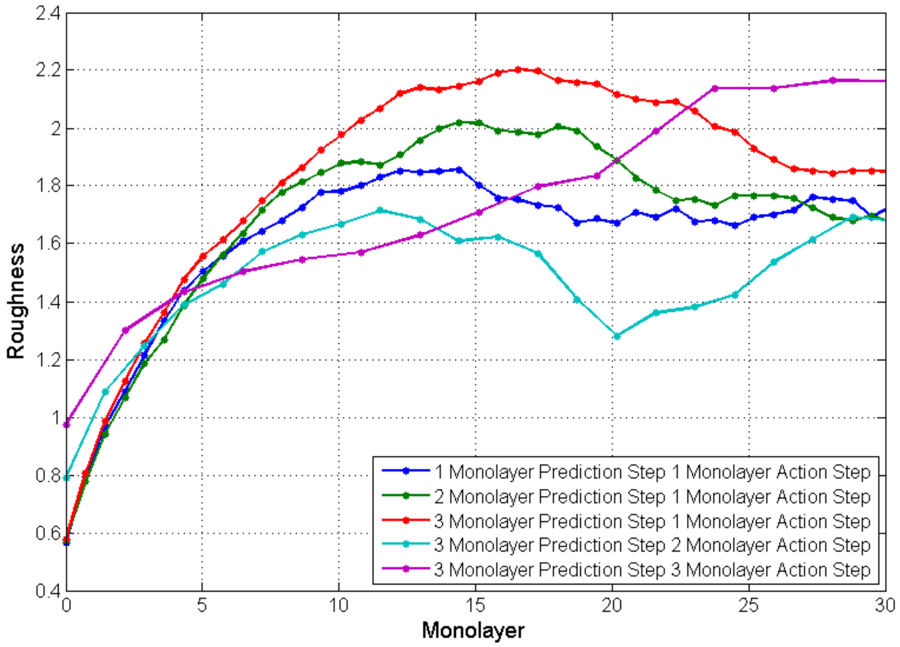


Figure 28: Transient Roughness Profiles at Different γ values.

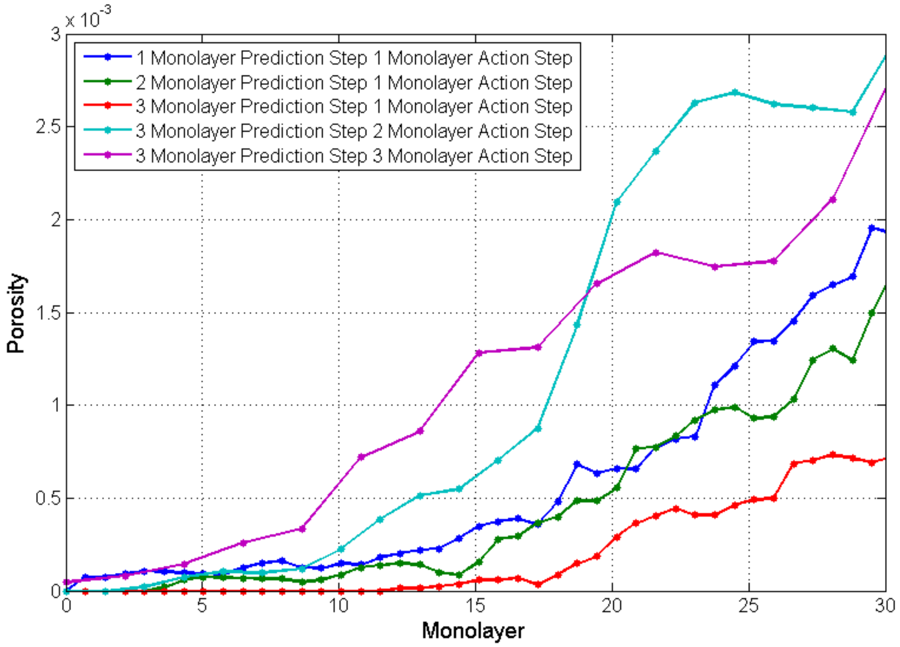


Figure 29: Transient Porosity Profiles at Different γ values.

5.2.4. Validation of Single Variable Input

As discussed previously, film properties vary as a function of γ. Therefore, the two available manipulated variables, temperature and deposition rate, can be combined into a single input variable, γ. This effectively transforms the Multiple Input Single Output control problem into a much simpler Single Input Single Output problem. However, an important limitation pertaining to the controller action must be noted. The control algorithm changes γ implicitly by modifying temperature explicitly. Since temperatures are sampled at equal intervals within a fixed range above and below the system temperature and γ is an exponential function of temperature, a fixed range of temperatures results in a variable range of γ values depending on the system temperature (Figure 30). When temperature is the only variable used to manipulate γ, at higher system temperatures the controller will not be able to change γ as rapidly as it would at lower temperatures, resulting in a more sluggish response. Therefore, the optimum transient profiles of two processes at different fixed deposition rates (therefore requiring different temperatures to keep γ constant) will be different, merely because the evolution of γ with time will be different. Thus, the optimal profile that the open-loop optimizer program gives is not entirely applicable for all deposition rates that the reactor could have. Within a range, it is acceptable. This is not a fault of γ as the manipulated variable, but is simply due to the fact that only temperature and deposition rate can be specified by the user for the SPPARKS input script.

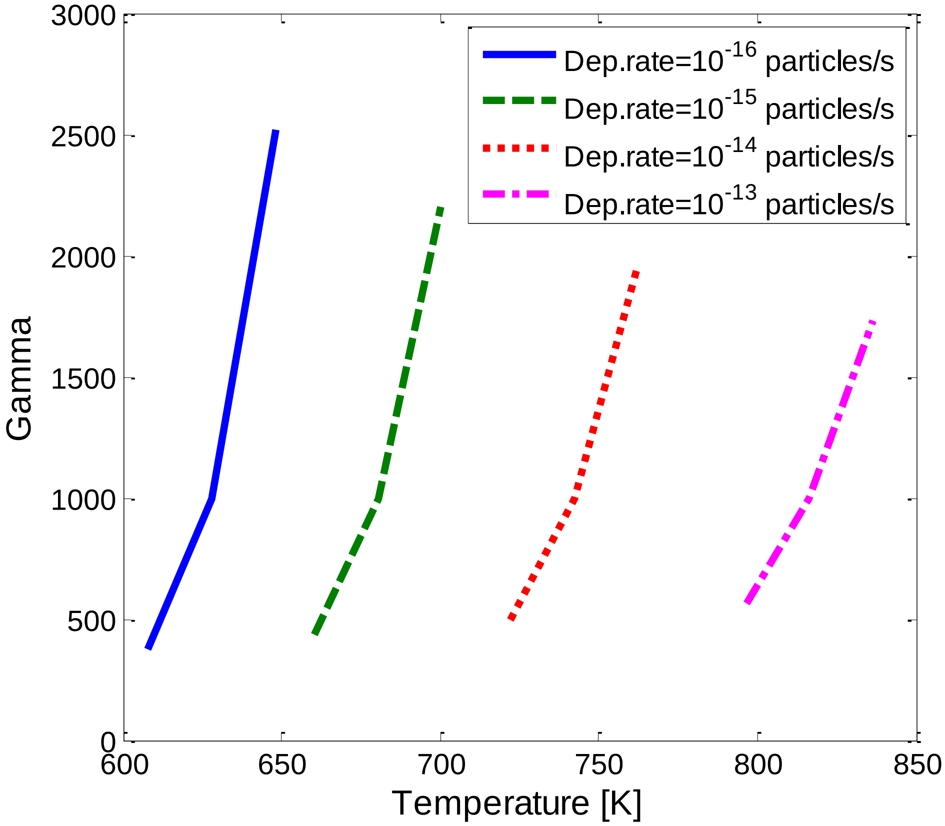


Figure 30: Change of γ Over a Constant Temperature Range. For each deposition rate, γ is plotted as a function of temperature over a range of 40K (T±20K), where T was chosen such that γ at T and the given deposition rate is equal to 1000. It is clear than for a constant change in temperature (40K), there is different change in γ, depending on the value of T.

5.2.5. Effect of Weighting Factors on the Optimal Transient Profile

As explained earlier in Section 4.4.2, the controller will allow the user to manipulate weightings on roughness and porosity in order to control for different types of film that the user is interested in. It was shown that applying two different weightings—Weighting 1 with 94% porosity and 6% roughness, and Weighting 2 with 0.04% porosity and 99.96% roughness—gave different results for the best steady-state γ. It is important to show that inputting these different weightings into the Open Loop Optimizer program will give the user an optimal profile to follow which leads to the desired film. The program was run three times each for the two weightings. A roughness profile is shown in Figure 31 below. As expected, over most of the run Weighting 2 gave a film with lower roughness. Unexpectedly, after rising quickly the roughness of the films with Weighting 1 had a sharp downturn towards the end of the run, ending with slightly lower roughness than films with Weighting 2. This may indicate that the algorithm used to determine an optimal profile may need further development; it was created on the assumption that obtaining the minimum objective function could best be obtained by minimizing the objective function at all time steps prior. If this assumption is incorrect, the algorithm may be giving a profile that puts the film into a local minimum of the objective function which may not enable the film to get to the minimum at the end of the growth.

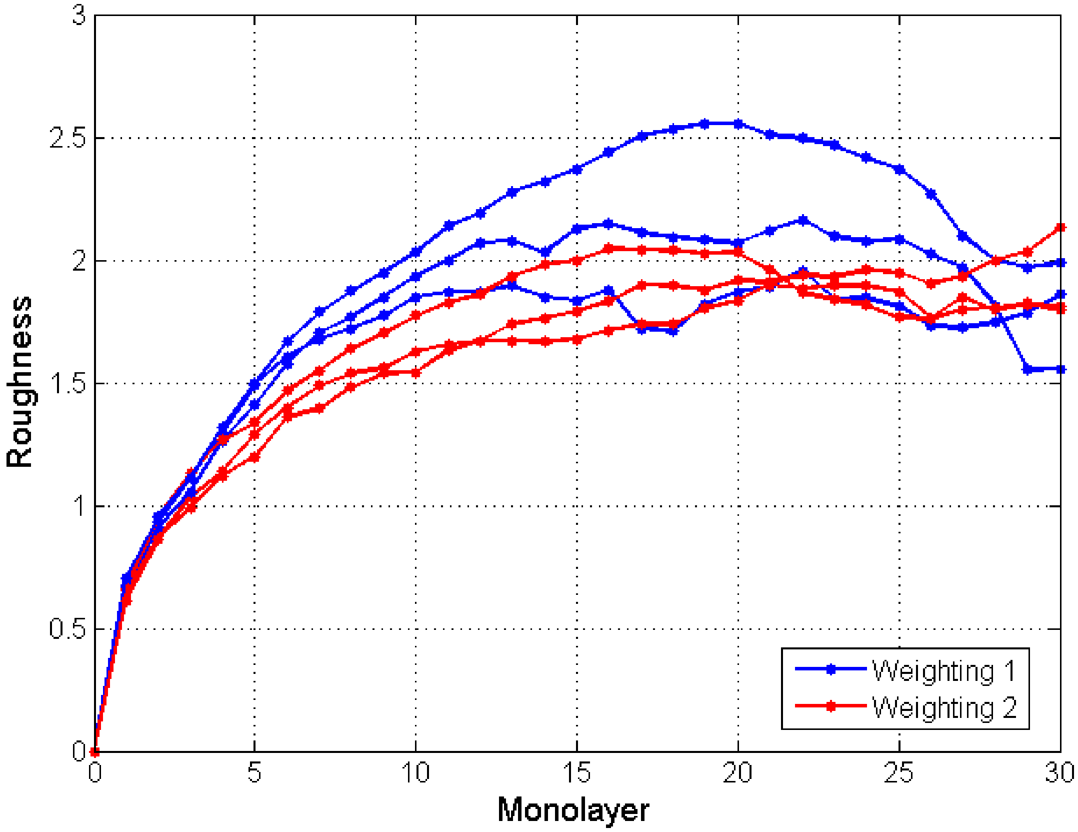


Figure 31: Optimum Transient Roughness Profiles at Different Weighting Factors.

A porosity profile of the two weightings is shown in Figure 32 below, and the γ profile in Figure 33. It appears that Weighting 2 did a better job of controlling porosity than Weighting 1, even though the intention was for Weighting 1 to control porosity more. However, observing the scale, it seems apparent that all profiles are within the noise of the system. It may be the case that for Material 1, increasing γ over time controls for both roughness and porosity, but when γ increases too much, the system gets into an excited state in which single-site pores form and quickly diffuse to the bottom of the film (as can be seen in Figure 34 below). The open loop optimizer may have increased γ for Weighting 1 in order to prevent the formation of large pores, inadvertently creating smaller pores. The open loop optimizer was started at a higher γ for Weighting 1 in accordance with the protocol to start the program at the optimal steady-state γ.

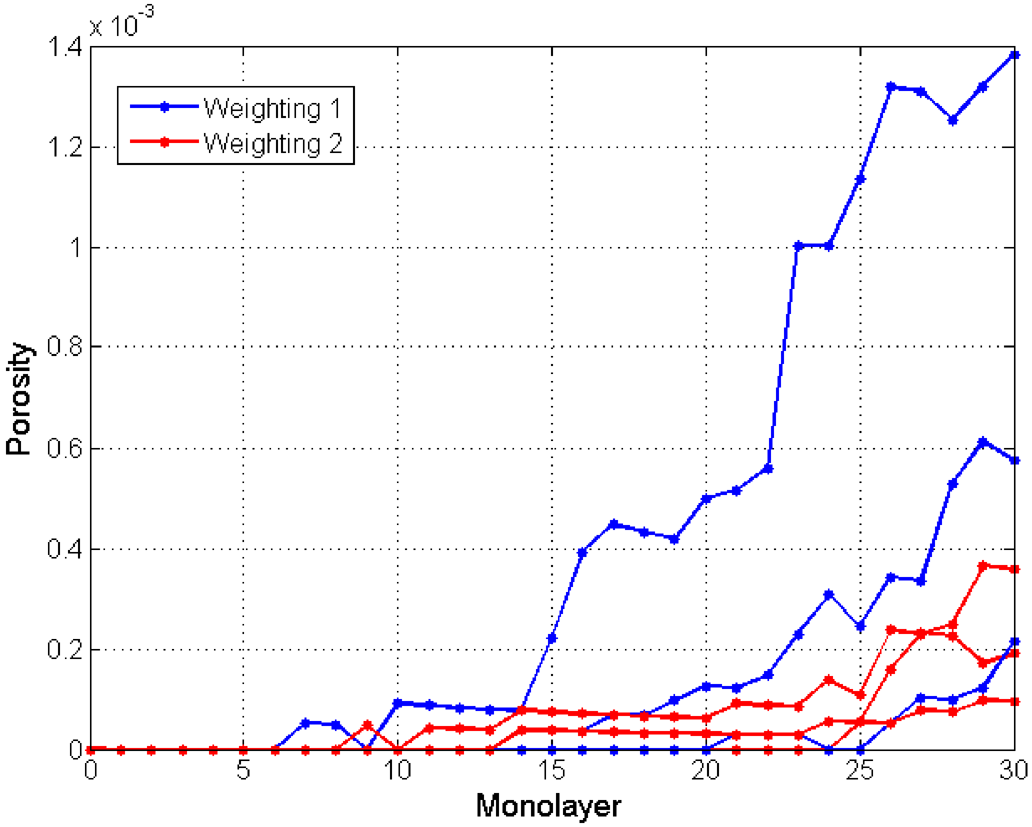


Figure 32: Optimum Transient Porosity Profile at Different Weighting Factors

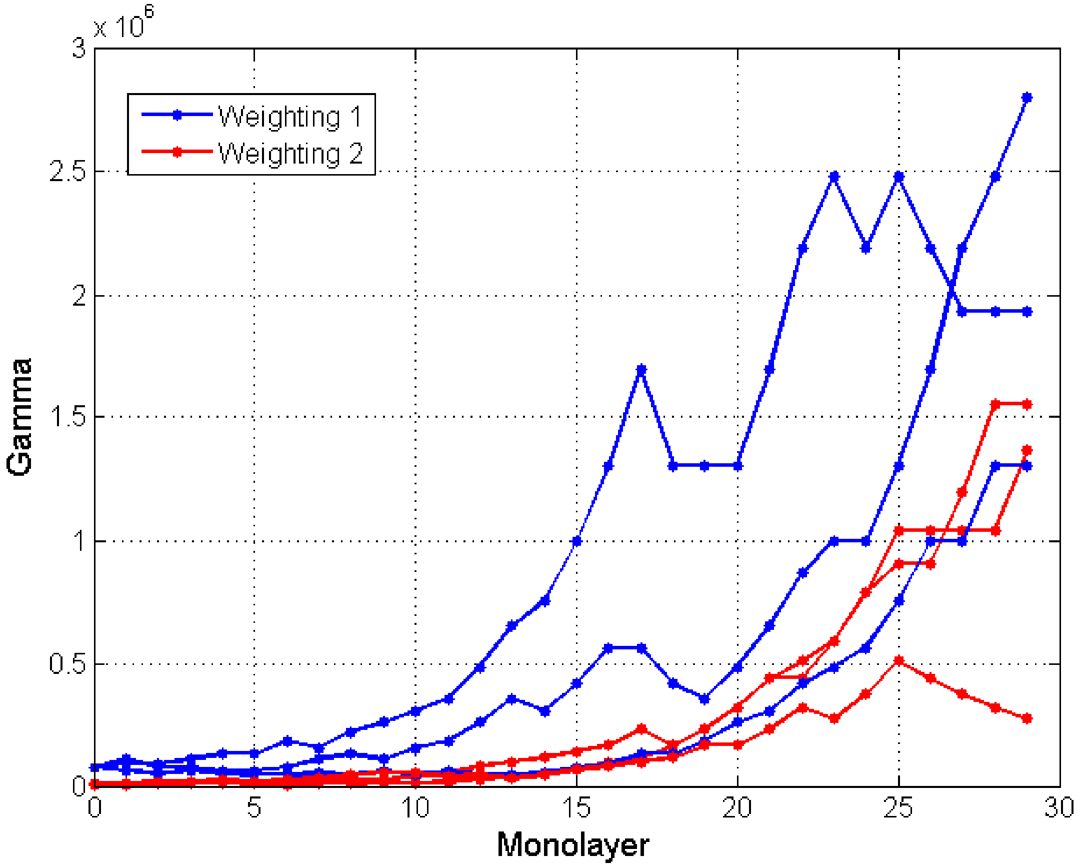


Figure 33: Optimum Transient γ Profile at Different Weighting Factor

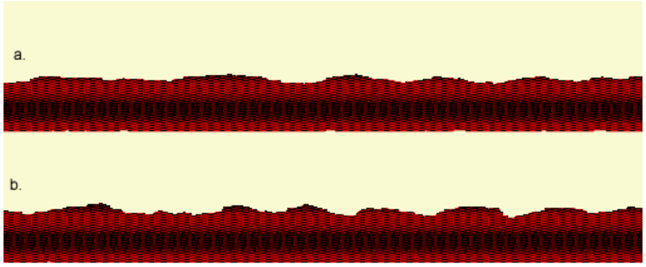


Figure 34: a. weighting 1; b. weighting 2.

6. Model Predictive Control: Closed Loop Control

6.1. Model Predictive Control Algorithm

Once the optimal profile has been established for a thin film deposition process, a model predictive control (MPC) strategy can be developed to control the complex microscopic interactions. The algorithm for this process is shown in Figure 20 on page 38. It was explained briefly in Section 5.1. It is known as closed-loop control due to the introduction of a set point (the reference trajectory) and a possible disturbance. The optimal profile serves as a reference trajectory that the deposition process is controlled to follow, resulting in a film with the properties specified in the open-loop optimization. This control strategy makes use of the same types of simulations that are used in the open-loop optimizer: a rigorous simulation represents the actual process in order to get real-time measurements, and less rigorous models predict results of a control action. This control strategy also makes use of the established terminology for the action step and the predictive step.

The largest difference between the MPC algorithm and the open-loop optimizer algorithm discussed in the previous chapter is that the MPC algorithm does not always go into the control loop. The model simulation simulates a predictive step at the initial temperature of the reactor. The objective function weightings that were used to develop the optimal profile are used to calculate the objective function of this prediction. This objective function is compared to the optimal profile of the process to produce an error measurement, expressed by Equation 18.

$$Error = \frac{|OF_p - OF_{sp}|}{OF_{sp}} \qquad \textbf{Equation 18}$$

where *OF* is the objective function, *p* denotes a process variable, and *sp* denotes a set point variable.

If the calculated error is within acceptable tolerances, the rigorous process will continue to simulate the next action step without adjusting temperature. If the error is greater than the tolerance, different temperatures are sampled for a predictive step. The optimizer samples temperatures in the same manner as the open-loop optimizer and with the parameters presented in Table 4 on page 43. The optimizer’s objective is to minimize the squared residual of the objective functions of the model simulation prediction and the optimal profile set point, Equation 19.

$$Objective_{optimizer} = min\left[\left(OF_P - OF_{sp}\right)^2\right] \qquad \textbf{Equation 19}$$

The temperature that meets this objective is used as the control action for the next action step. For the purposes of this design the controller is assumed to be a perfect controller, since the rigorous process simulation is paused while the control action is computed. The control action cycle is repeated until all the set points of the profile are exhausted.

Parameter	Specification
Predictive Step	2 monolayer
Action Step	1 monolayers
Rigorous Size : Model size	1000:200
Model Simulations to Average	5
Error Tolerance	0.001
Temperature Range [˚C]	20
No. Temperature Trials	7
Roughness Weighting	0.06
Porosity Weighting	0.94

Table 6: Model Predictive Controller Parameters.

6.2. Roughness Disturbance Experiment

In the photovoltaic and semiconductor industries, initial surface disturbances cannot be entirely eliminated. A model predictive control system must be able to compensate for these defects and yield the desired film morphology within reasonable response times. It is assumed that the best way to accomplish this is to get the disturbed film’s objective function trajectory to join the optimal process trajectory, which was obtained starting from an undisturbed film. The model predictive controller was tested on a surface with an initial roughness disturbance, shown in Figure 35. The morphology resembles peaks and troughs, with peaks having a height of 15 monolayers and the troughs 7 monolayers. The peaks have a periodicity of 200 lattice sites on a 2000 site lattice. The reference trajectory that was used for these experiments was the optimal transient profile that was developed in the open-loop optimizer section for a surface with no disturbance.

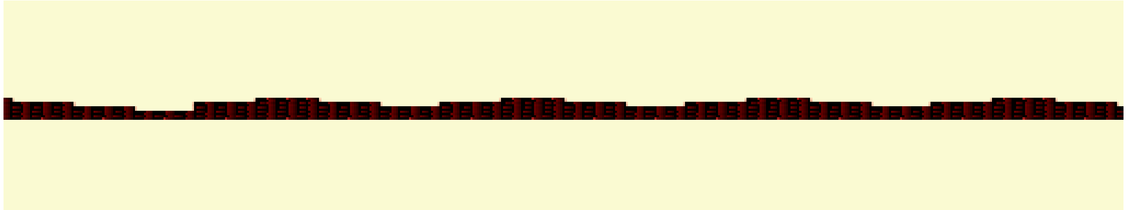


Figure 35: Initial Roughness Disturbance to a Model Predictive Controlled process. Note that it's zoomed so not all 10 peaks are displayed.

Figure 36 shows the objective function profiles resulting from the model predictive control process as compared to the optimal transient profile. For comparison, the optimal steady-state temperature was also run on the roughness disturbed surface. These results indicate that the controller is taking action to correct the morphology of the film. Comparing the model predictive control profile with the optimal steady state profile, there is a significant adjustment of the controlled process towards following the reference trajectory of the optimal transient process. Figure 37 shows the adjustments made to the thin film deposition process γ by the controller. It is evident that the γ adjustments of the controller lag the γ adjustments that the optimized process requires. However, the same adjustment trend exists.

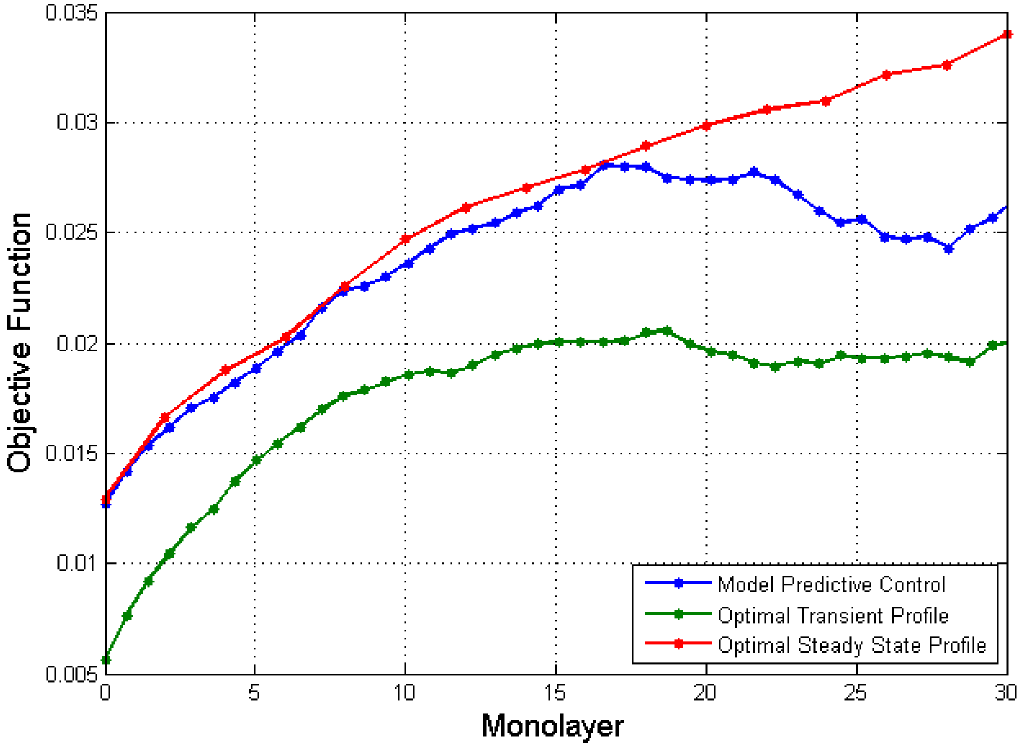


Figure 36: Objective Function Profiles of the Roughness Disturbance Experiment. The MPC did a better job than the optimal steady-state temperature at achieving an optimal film.

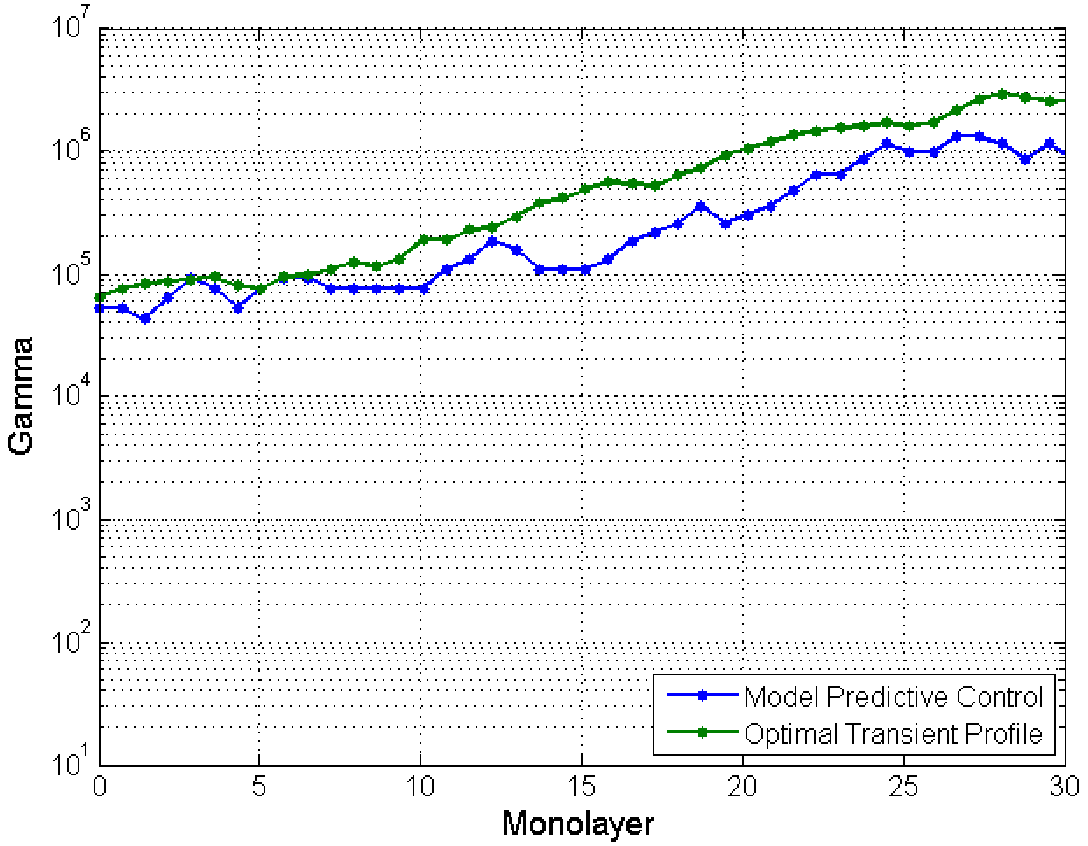


Figure 37: Gamma Profiles of the Roughness Disturbance Experiment.

The most important result from the controller is the resulting thin film morphologies. Since the controller is using a numerical representation of the desired properties, it is important to confirm that when the process's objective function trajectory is adjusted to follow a reference optimal profile of the process that it results in the desired film. With this effort in mind, Figure 38 displays a comparison of the thin film morphologies resulting from the open-loop optimizer, the model predictive control process, and the optimal steady state process. Although the morphology of the model predictive control on an initial roughness disturbance is not completely similar to the desired morphology, the morphology of the controlled film has been improved over the steady state film. With more time, the model predictive controller might even exactly match the optimal film.

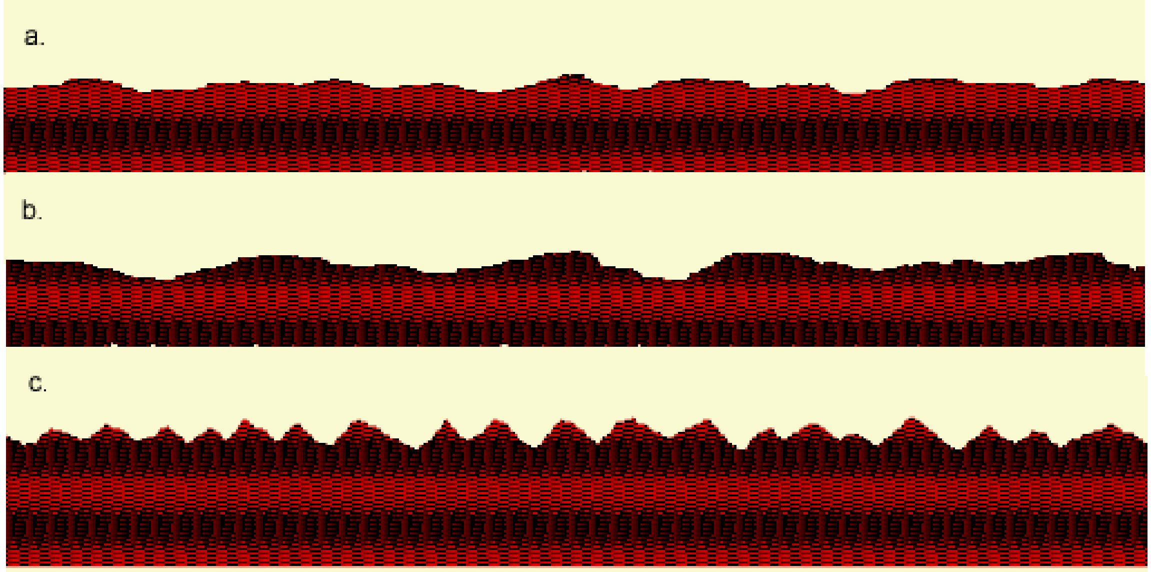


Figure 38: Model Predictive Control Comparison of Resulting Film Morphologies. a. The desired or reference film resulting from the Open-loop Optimizer, b. the film resulting from model predictive control of a rough initial disturbance on the film, c. film resulting from the Optimal Steady State Temperature. From a comparison to the optimal steady state film with no control, the model predictive control has demonstrated corrective action to produce the desired film. It is hypothesized that for a longer process the resulting morphologies of the reference and the controlled film will be very similar.

The proposed model predictive control strategy has been shown to perform necessary adjustments to control the morphology of the film. However, it is important to note that a more rigorous study must be undertaken to better characterize the model predictive controller. Some possible experiments may consist of performing similar step combination experiments as presented for the open-loop optimizer. Measurement noise must also be considered. It is noteworthy that sometimes the controller failed to make proper adjustments due to noisy predictions.

6.3. Step Change Experiment

The reference trajectory that the MPC follows does not necessarily have to be of the optimal profile. Any trajectory can be fed in to the MPC and followed. One experiment conducted, which took advantage of this fact, observed the effect of a step change in γ (temperature). An objective function profile was constructed by setting up a run on an initially smooth surface in which γ was suddenly increased after 30 monolayers of growth. Figures 40a and 40b demonstrate that the model predictive controller was successful in tracking γ and temperature. Figure 40b demonstrates that the maximum ramp rate performed by the controller was just 10 ^{o}C/s, although it had a maximum allowable temperature ramp rate of 20 ^{o}C/s. That the controller was not adjusting as quickly as it could to the step change suggests that some lag is present in the system. After 55 monolayers of growth, the objective function of the model predictive controller came to within 6% of the objective function of the reference profile. Then, the reference trajectory was exhausted, which stopped the controller simulation at a point during the overshoot of the controller. The profile did not extend to a point in the process at which the controller was allowed to settle at the new temperature. Thus, further experiments must be undertaken to fully characterize the response of this controller. These may include a step response experiment in temperature that allows the system to settle to obtain control variables such as gains and response times.

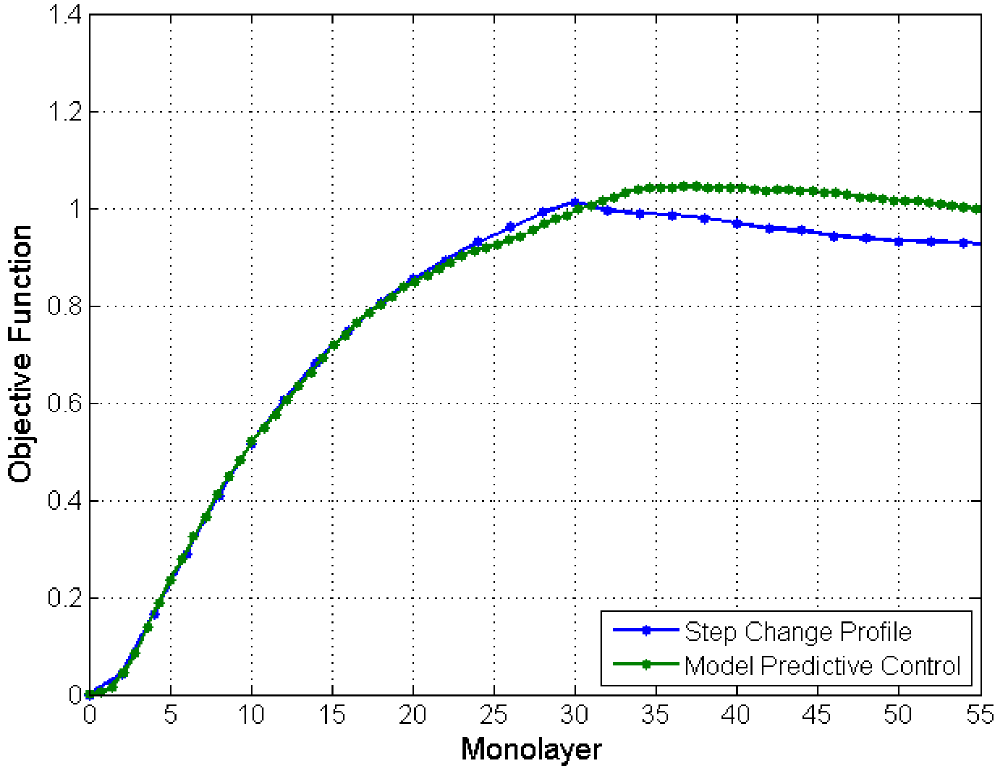


Figure 39: Objective Function Profiles for the Model Predictive Controller Temperature Step Change.

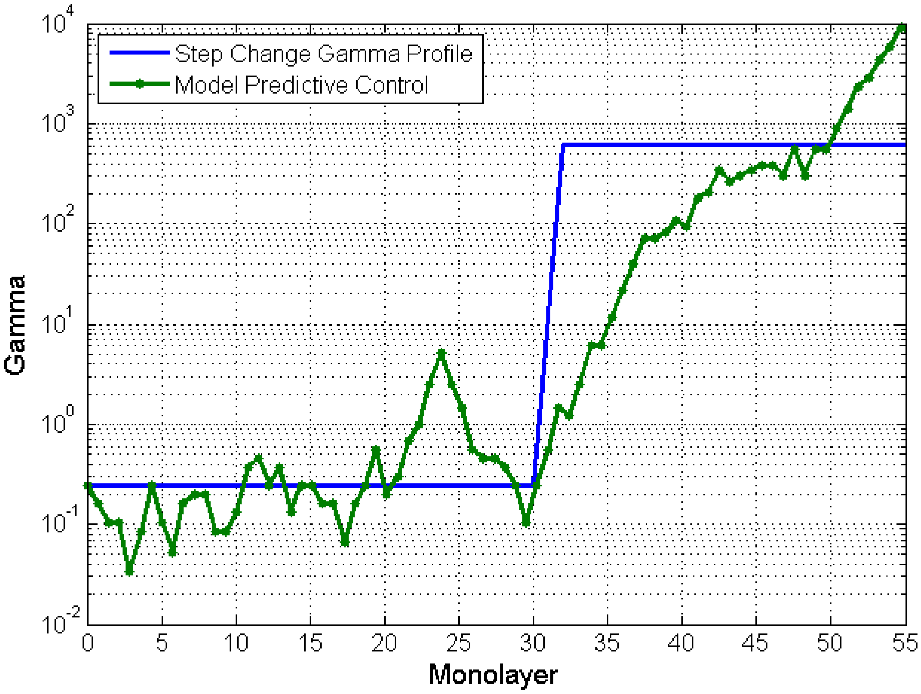


Figure 40a: Temperature Profiles for the Model Predictive Controller Temperature Step Change.

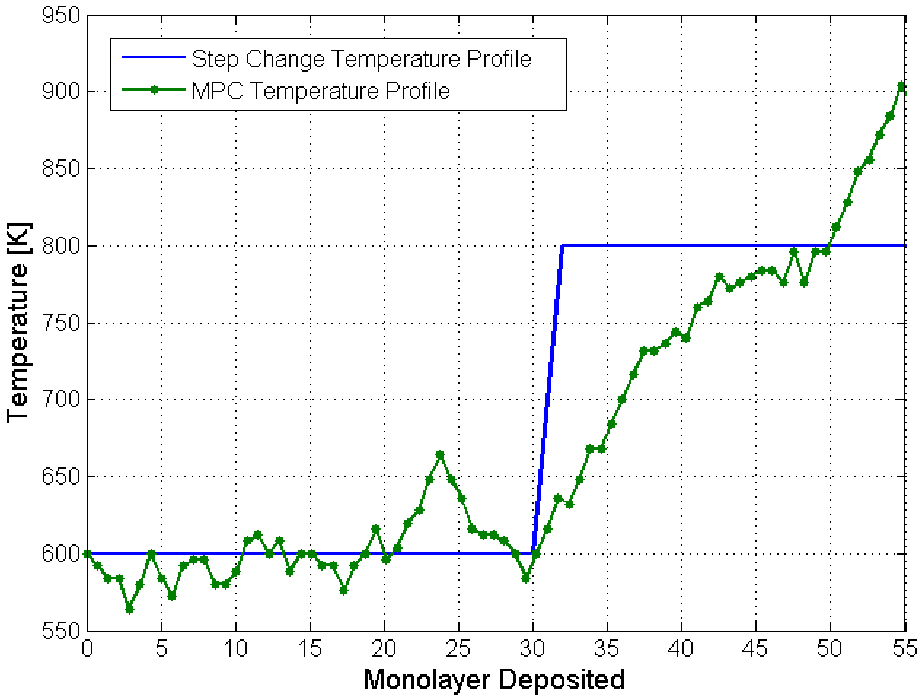


Figure 40b: Temperature Profiles for the Model Predictive Controller Temperature Step Change.

On a qualitative basis, the controller produced a similar morphology to that of the reference material. Figure 41.a shows that at the bottom of the reference film the structure is more porous. This corresponds to the lower γ experienced by the film for the first 30 monolayers deposited. For the surface of the film, a higher temperature corresponding to a higher γ induces relaxation of the morphology. The particles begin to fill in pores after 30 monolayers and smooth out rough edges. The film resulting from the model predictive control run in Figure 41.b has a similar morphology to that of the reference material.

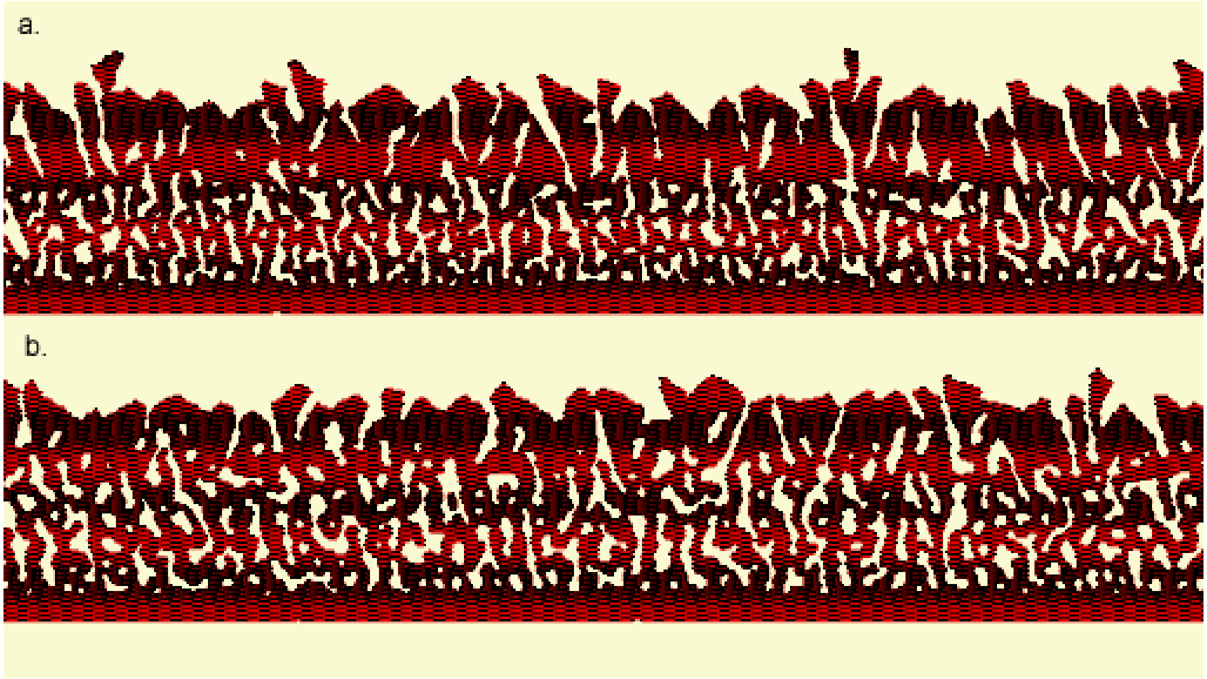


Figure 41: Resulting Morphologies of the Reference Material (*a*) and the Controlled Material (b). The purpose of the model predictive controller is to obtain a desired morphology represented by the reference material. The controller controls the process based on the specific requirements as represented by the objective function. As the controller tracks predicted changes and attempts to follow the optimum transient profile, the controller is subjecting the film to control actions that are sensitive to the materials process history. This produces the desired morphologies such as the reference material.

7. Other 2D Materials

7.1 Material 1a; Schwoebel Hops

7.1.1. Optimal Steady-State Profile

In addition to the base case material, other two-dimensional materials were explored. One of the other materials explored used the same binding energies as the base case, but it allowed Schwoebel hops. This is termed Material 1a as it is a slight adjustment on the base case material. As explained in the Kinetic Monte Carlo section, Schwoebel hops occur when an atom hops to its second nearest-neighbor. The diffusion barrier of these Schwoebel hops was specified to be 1.84 eV, in comparison to a 1.58 eV barrier for nearest-neighbor hops. This resulted in approximately one-fifth the number of Schwoebel hops as there were nearest-neighbor hops. Experiments were run at the same γs, temperatures, and deposition rates as were used to characterize the base case material.

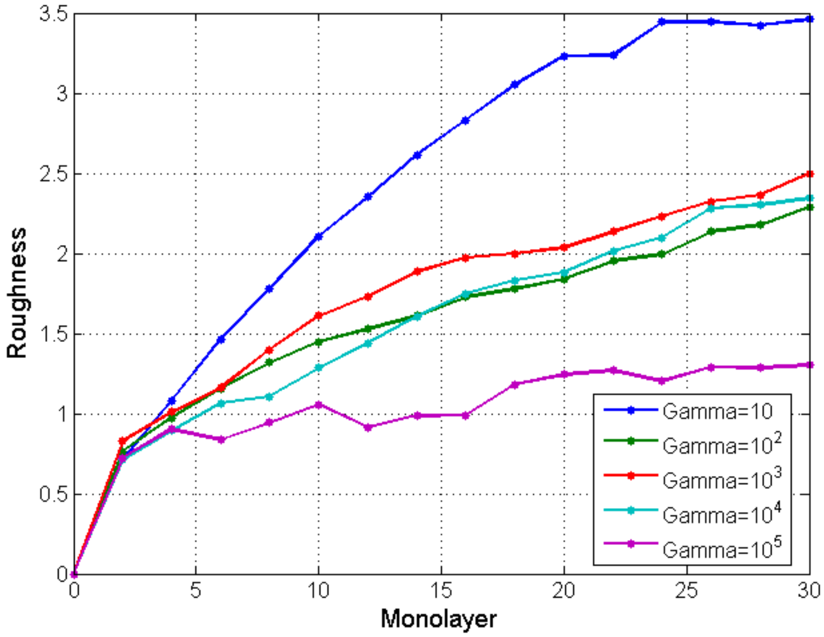


Figure 42: Roughness vs. Monolayer for Material 1a. Schwoebel hops allowed**.**

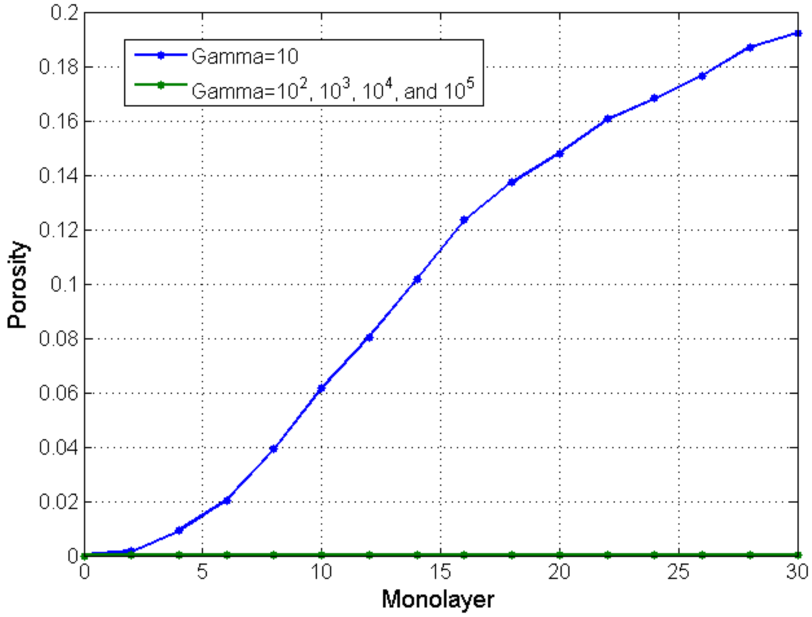


Figure 43: Porosity vs. Monolayer for Material 1a. Schwoebel hops allowed.

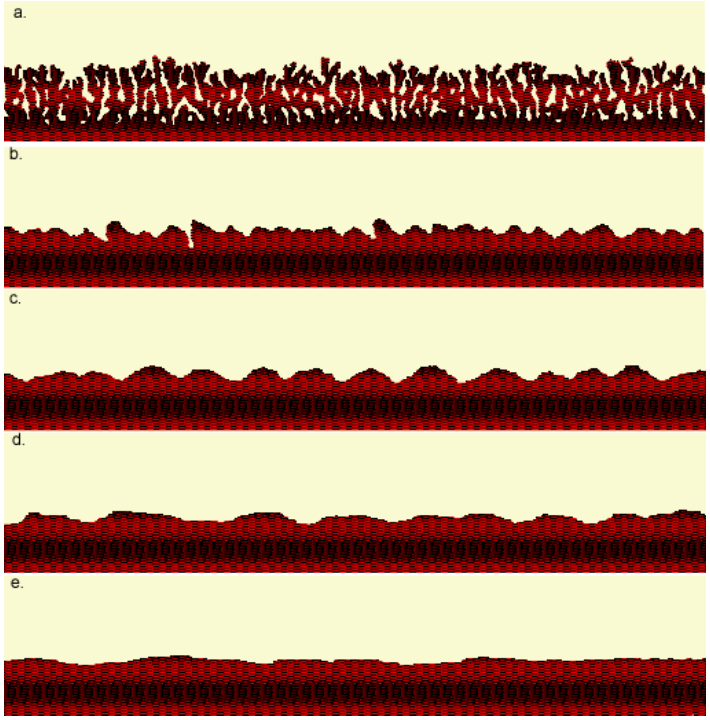


Figure 44: Film images of Material 1a. Schoebel hops allowed, 30 monolayers deposited. a: γ=10. b: γ=10^2. c: γ=10^3. d: γ=10^4. e: γ=10^5.

As can be seen from Figure 42, roughness decreases with increasing γ. Interestingly, exploring the same γ range as had been explored for the base case material (up to 10^4), it was found that roughness found a local minimum at a value of 10^2. Had experiments not been conducted setting γ to 10^5, Material 1a might have been found to have the same roughness response as the base case material. This leads one to question if the optimal steady-state temperature found for the base case is truly the optimal steady-state temperature. γ ranges were explored that were physically reasonable, so within that limit, the optimal steady-state temperature found for the base case material is correct, but increasing γ above that limit might have resulted in a much smoother film. More developmental work is necessary to explore this hypothesis.

As can be seen from Figure 43, Material 1a reaches a porosity of zero at a γ much lower than that in Material 1. The images in Figure 44 suggest that Material 1a is very similar to the base case; it reacts to γ in the same fashion while similar reactions can be seen at one order of magnitude lower γ than the reactions of the base case material. This again suggests that if the γ of the base case material had been increased beyond physical limitations, a smoother film would have developed.

Due to the fact that both roughness and porosity decrease with increasing γ, an open-loop optimizer program was unable to be run on Material 1a as there was no control problem. The open-loop optimizer program would simply increase temperature as much as it could to decrease both roughness and porosity. A third parameter needs to be set such that increasing γ would increase the value of this parameter, which would give a control problem. Such a parameter would be efficiency, which is only possible is desorption is allowed; since SPPARKS does not allow for desorption, such a parameter cannot be used. As the controller changes γ, it implicitly changes growth rate, but without Multiple Input Single Output it not possible to distinguish whether changes to γ are due to changes in temperature or growth rate.

7.1.2. Optimal Transient Profile

For Material 1a, the same objective function weightings as with the base case material (6% roughness, 94% porosity) were used. As shown in Figure 45 and 46 below, the optimal steady-state γ was found to be 10^5, the highest value tested. It is likely that a higher value of γ would have yielded a more optimal steady-state profile.

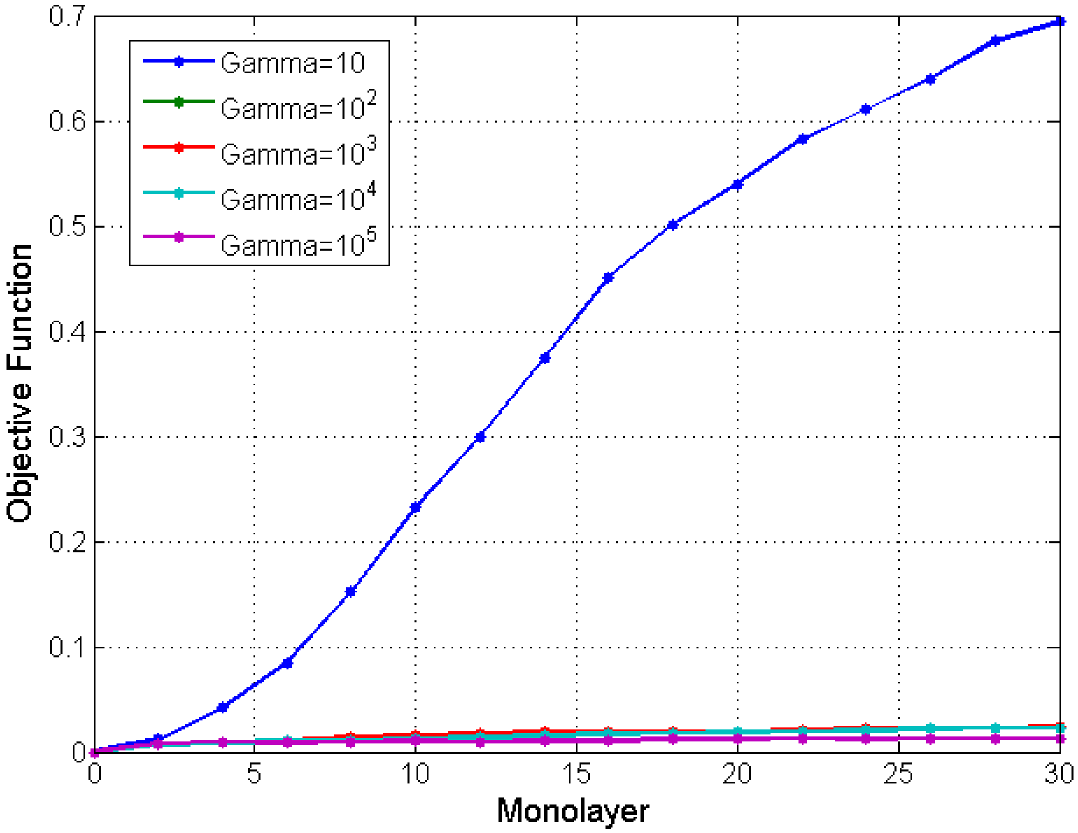


Figure 45: Time evolution of objective functions at different γ values for Material 1a. Weightings for the objective function were 6% roughness and 94% porosity. The optimal steady-state γ was found to be 10^5. A close-up with γ=10 excluded is found in Figure 46 below.

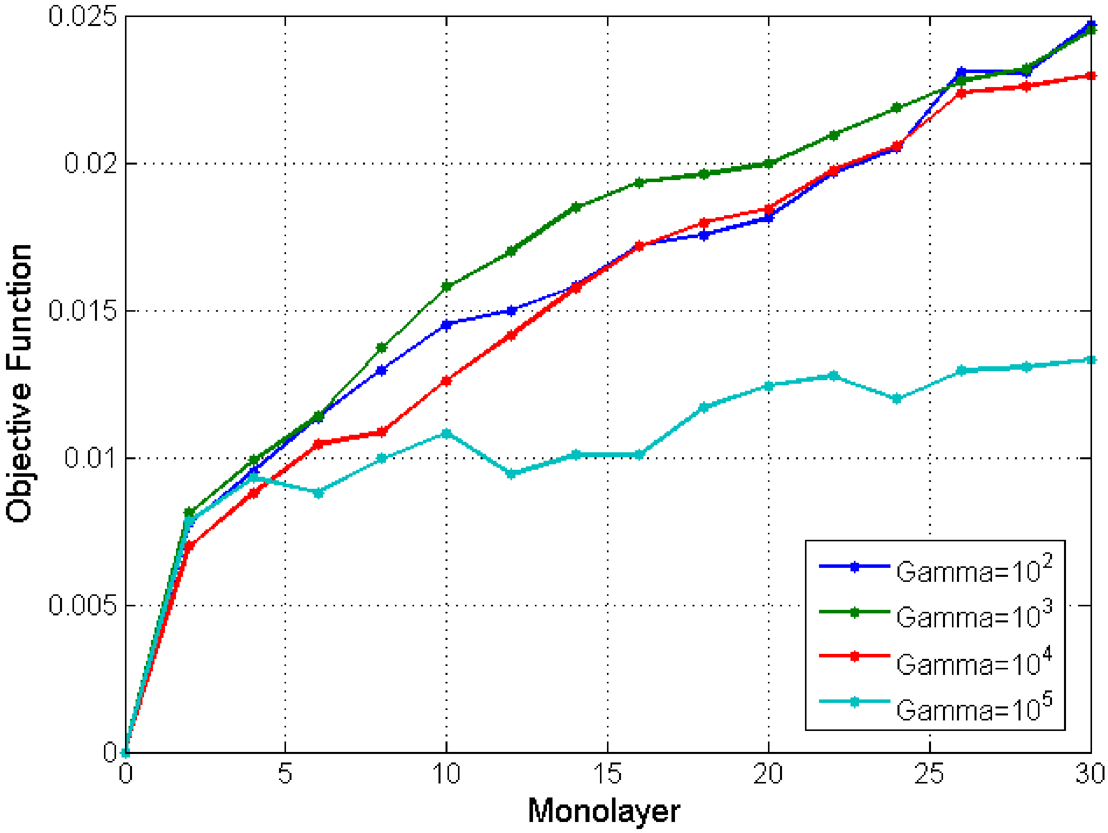


Figure 46: Time evolution of objective functions at different γ values for Material 1a. Weightings for the objective function were 6% roughness and 94% porosity. The optimal steady-state γ was found to be 10^5. This is figure 45 zoomed in to exclude γ=10.

As explained earlier, the open-loop optimizer program was started at the optimal γ. For Material 1a, temperature was allowed to change 40 K per second, which is 20 K per monolayer. The γ profile with respect to monolayers deposited is shown below in Figure 47. As can be seen, the program seemed to form a sinusoidal pattern with γ, oscillating around 5X10^5, a value slightly higher than the optimal steady-state γ. This is not unreasonable, as higher values of γ were not tested when finding the optimal steady-state value. As shown in Figure 48 below, the transient profile was approximately as good as the best steady-state profile. Figure 49 shows the image of the best steady-state temperature film and the image of the open-loop optimizer film, which look nearly identical.

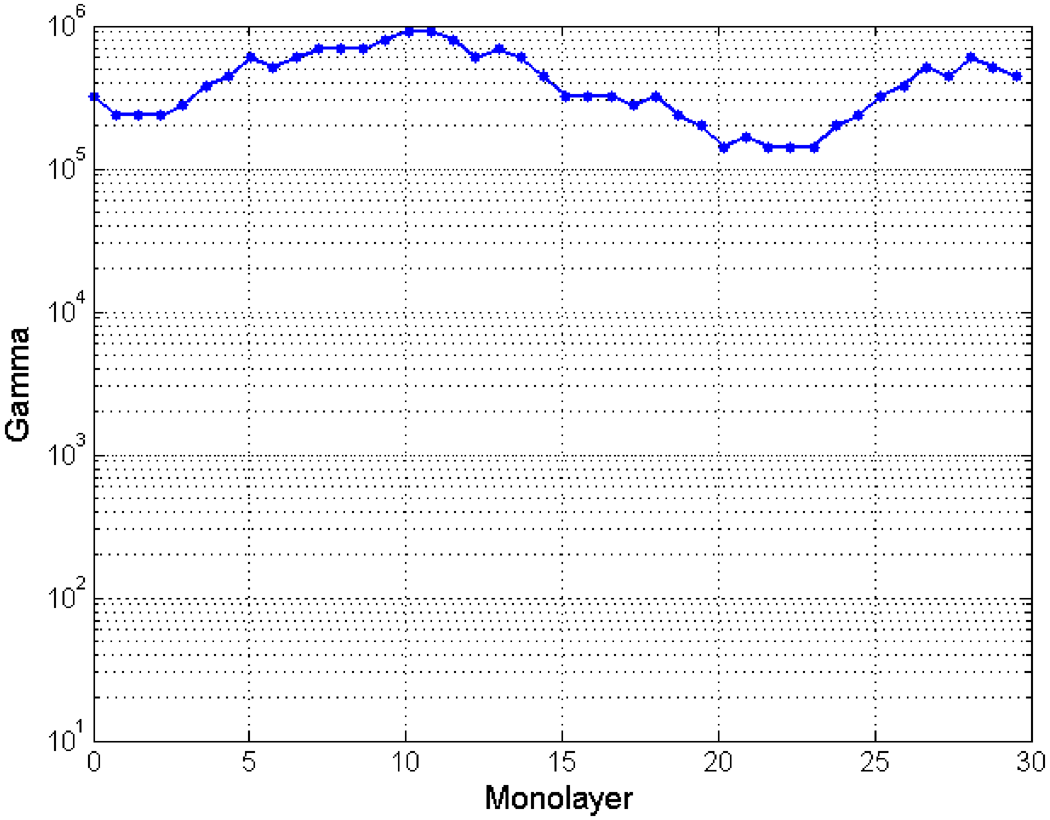


Figure 47: Optimum Transient Gamma Profile of Material 1a.

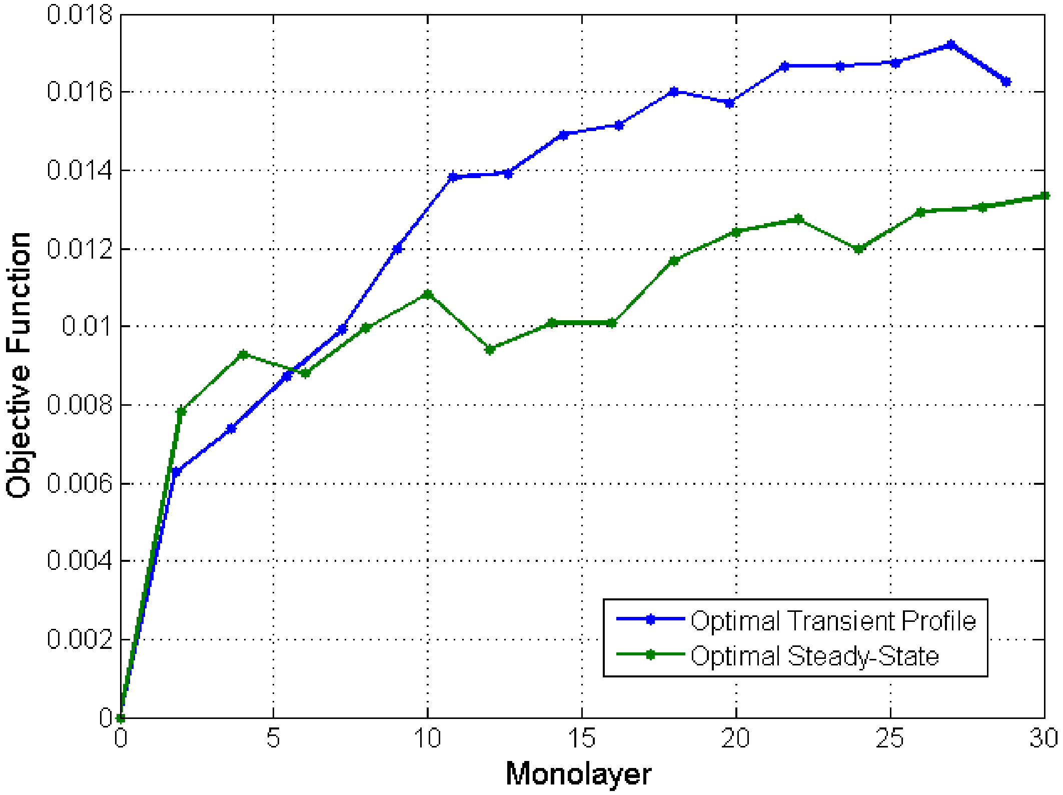


Figure 48: Objective Function Profile for Material 1a. The optimal transient profile does no better than the optimal steady-state profile.

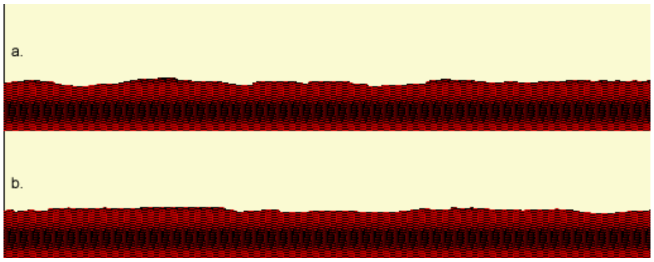


Figure 49: Film images of Material 1a, with Schwoebel hops, at 30 monolayers deposited. a. Optimal Steady-State; b. Optimal Transient Profile.

7.2 Material 2; Square Lattice

7.2.1 Optimal Steady-State Profile

Another material explored had a square lattice. This lattice is shown in Figure 50 below.

Every atom has eight nearest-neighbors and sixteen second nearest-neighbors. This material was given the same bonding energies as the base case material; Schwoebel hops were not permitted. This material is termed Material 2.

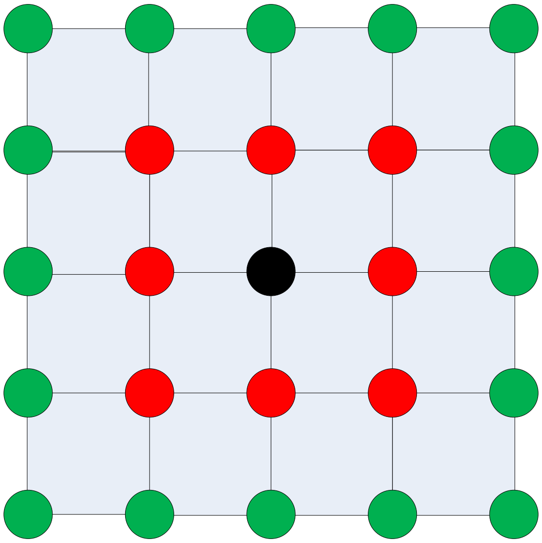


Figure 50: Square-lattice diagram. Shows nearest neighbors (red) and next-nearest neighbors (green)

This material behaved in a very different manner from the base case material and Material 1a. As shown in Figure 51 below, roughness was observed to be constant, regardless of γ. No sinusoidal waves formed on the material’s surface, and it was relatively flat.

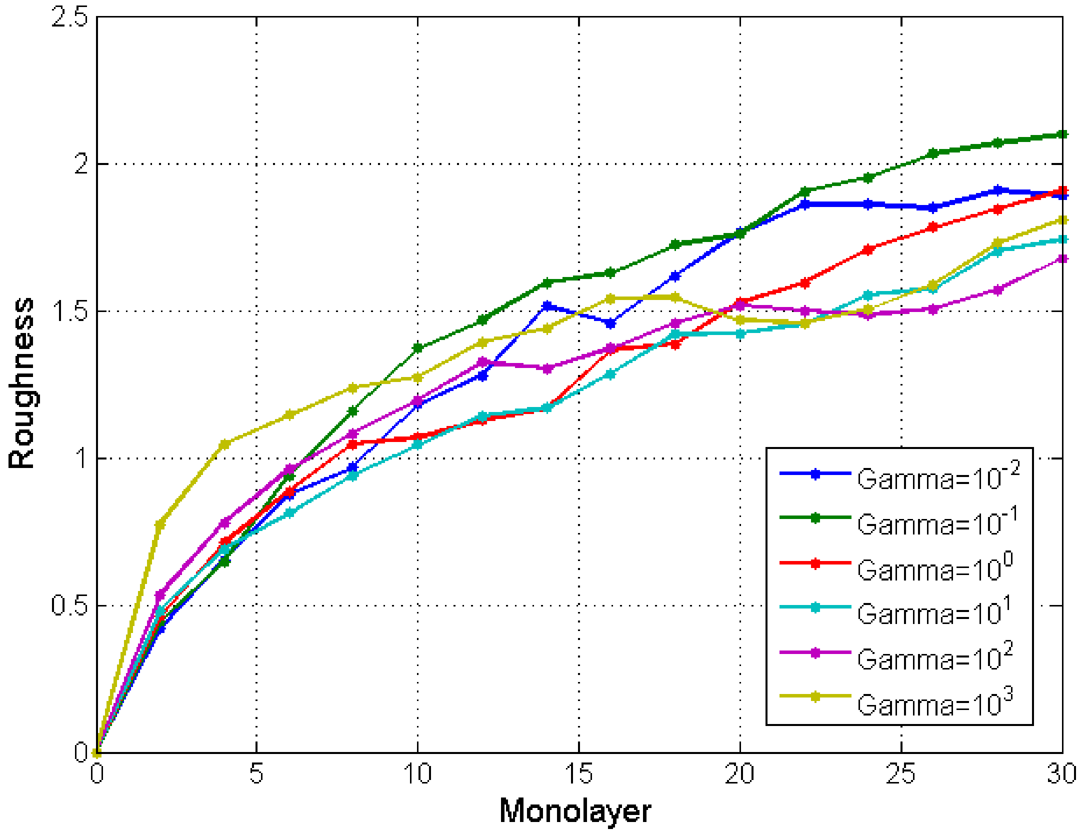


Figure 51: Time evolution of roughness at different γ values for Material 2. Roughness is not a function of γ.

As shown in Figure 52 below, porosity also behaved in a very different manner than the base case material or Material 1a. As γ increases from 10^{-2} to 10^{1}, porosity decreases as expected. However, as γ increases from 10^{1} to 10^{3}, porosity increases. Looking at the material in Figure 53 below, this porosity increase as γ increases from 10^{1} to 10^{3} is not seen. However, by zooming in on the material, shown in Figure 54, one sees that in this higher range of γ very small single-site pores form in such a quantity that many of these small pores create greater porosity than fewer large pores. By the highest γ, the system was in a state such that most of the bottom monolayer was unfilled. Essentially, the system had enough energy to assume any state.

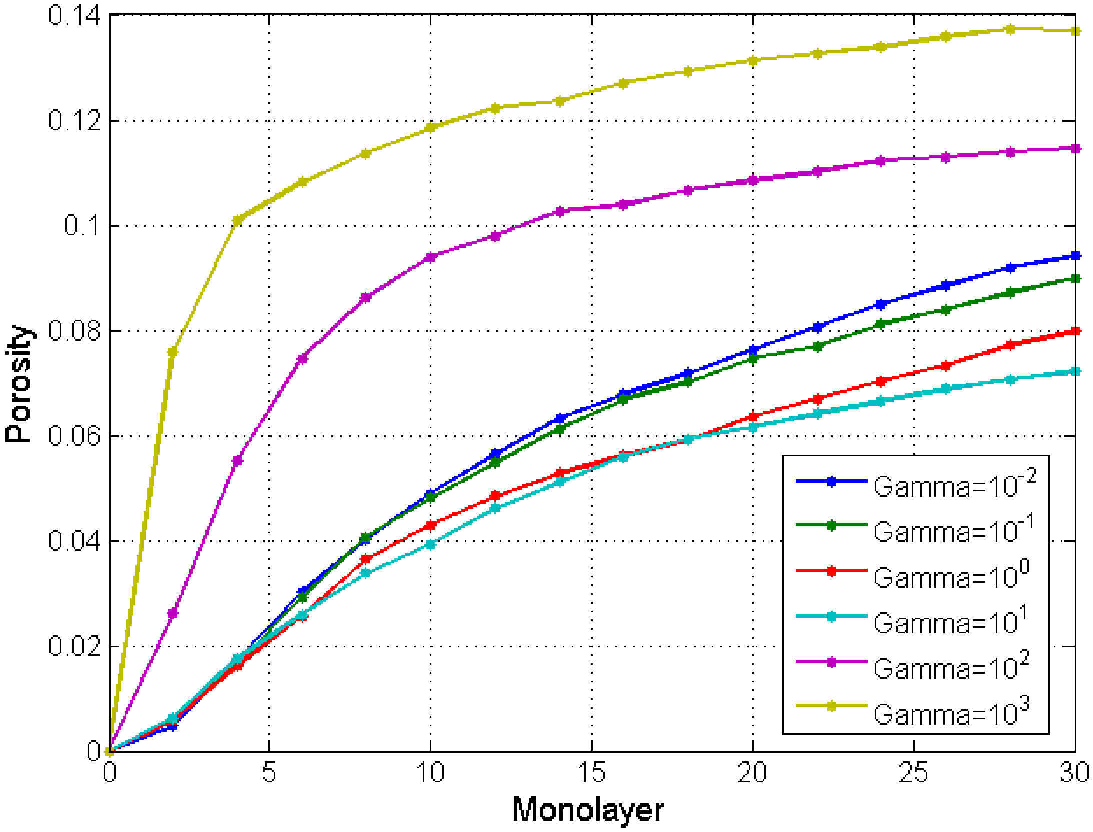


Figure 523: Time evolution of porosity at different γ values for Material 2. Porosity decreases with increasing γ, as expected, until many small single-site pores begin to form, at which point porosity increases with increasing γ.

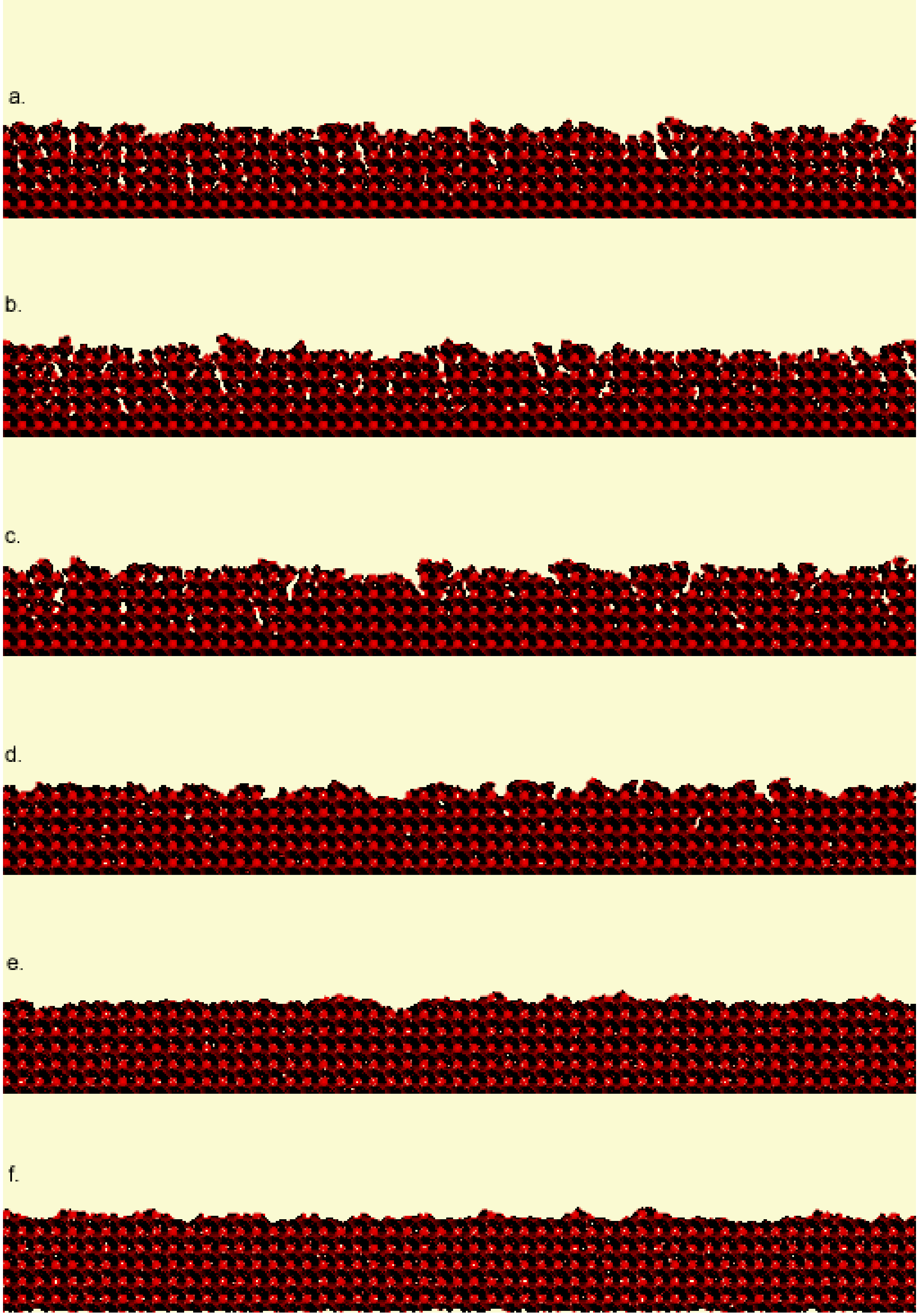


Figure 534: Film images of Material 2, with a square lattice, at 30 monolayers deposited. a: $\gamma=10^{-2}$. b: $\gamma=10^{-1}$. c: $\gamma=10^{0}$. d: $\gamma=10^{1}$. e: $\gamma=10^{2}$. f: $\gamma=10^{3}$.

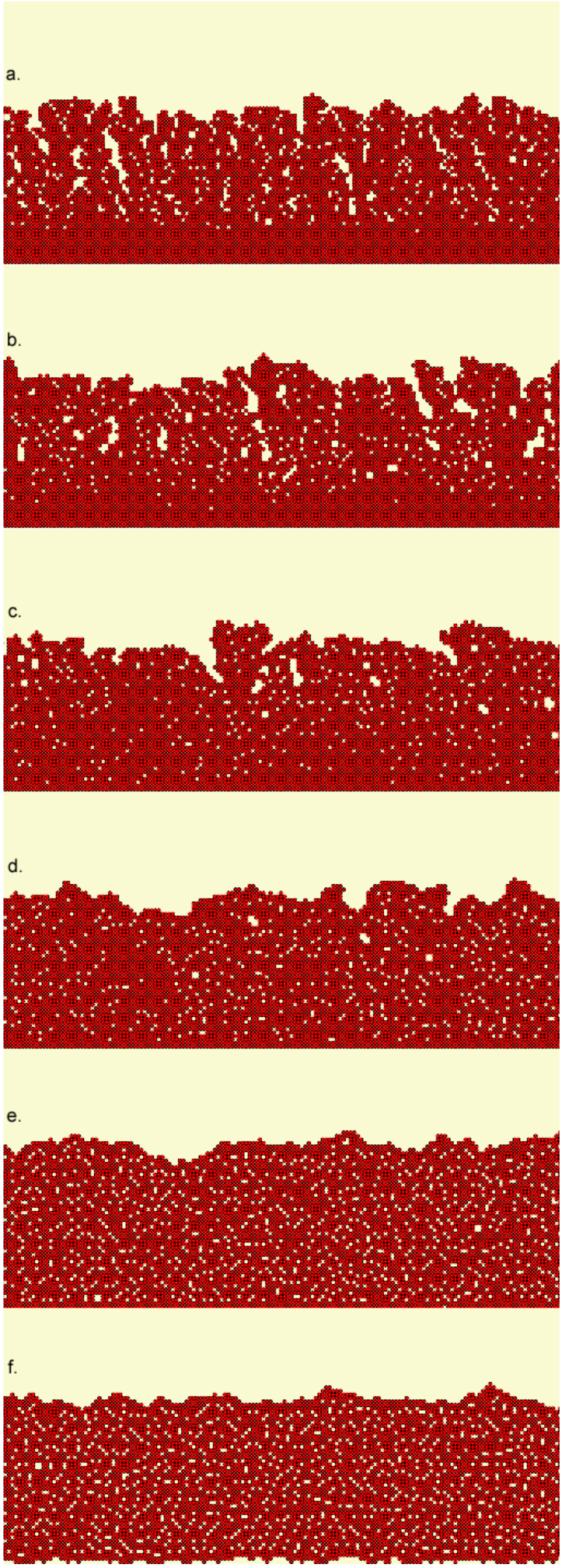


Figure 545: Film images of Material 2, with a square lattice, at 30 monolayers deposited. Zoom 3X. a: $\gamma=10^{-2}$. b: $\gamma=10^{-1}$. c: $\gamma=10^{0}$. d: $\gamma=10^{1}$. e: $\gamma=10^{2}$. f: $\gamma=10^{3}$. Many small, single-site pores are observed to form at high γ, increasing porosity.

7.2.2. Optimal Transient Profile

For Material 2, the same objective function weightings as were used with the base case material (6% roughness, 94% porosity) were used. As shown in Figure 55 below, the optimal steady-state γ was found to be 10^1.

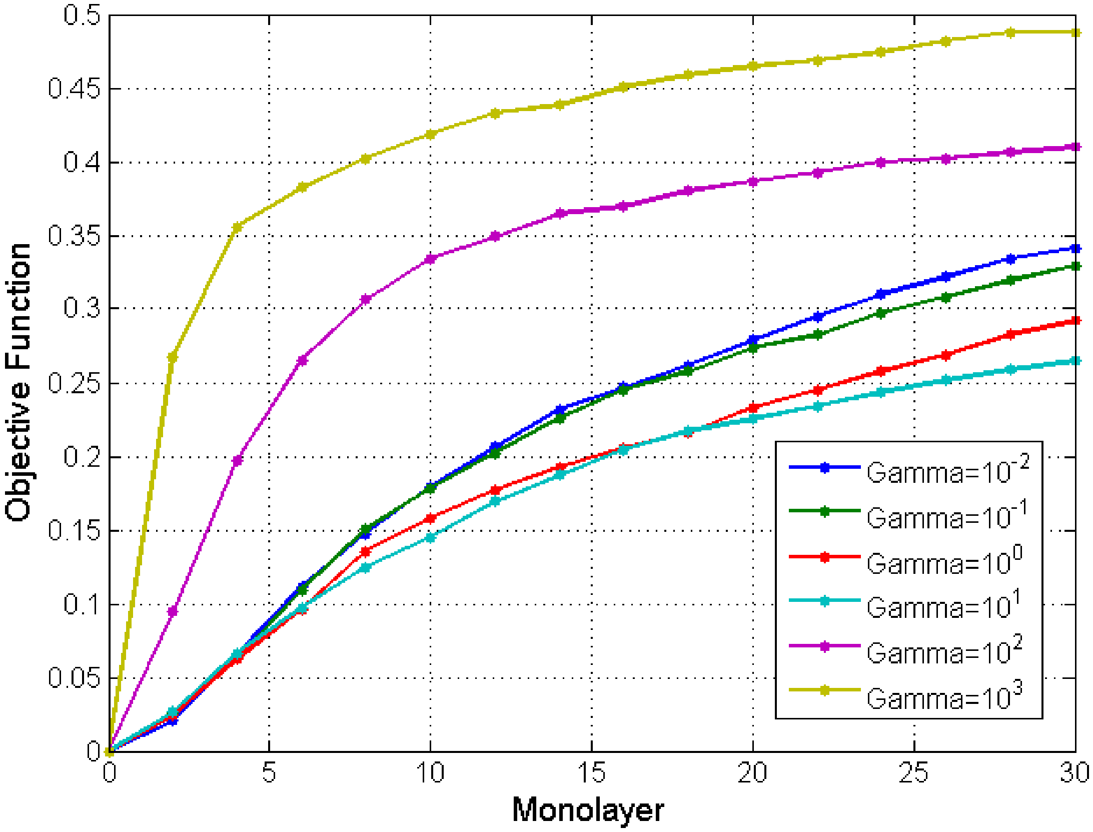


Figure 556: Time evolution of objective functions at different γ values for Material 2. Weightings for the objective function were 6% roughness and 94% porosity. The optimal steady-state γ was found to be 10^1.

The open-loop optimizer program was again started at the optimal γ. For Material 2, temperature was allowed to change 40 K per second, which is 20 K per monolayer. The γ profile with respect to monolayers deposited is shown below in Figure 56. As can be seen, the program quickly lowered γ, and then held it at a steady-state value. It is likely that the optimal steady-state γ was 2 instead of 10, but that Figure 55 did not have enough resolution. As shown in Figure 57 below, the transient profile was approximately as good as the best steady-state profile. Figure 58 shows the image of the best steady-state temperature film and the image of the open-loop optimizer film at 3X zoom, which look nearly identical.

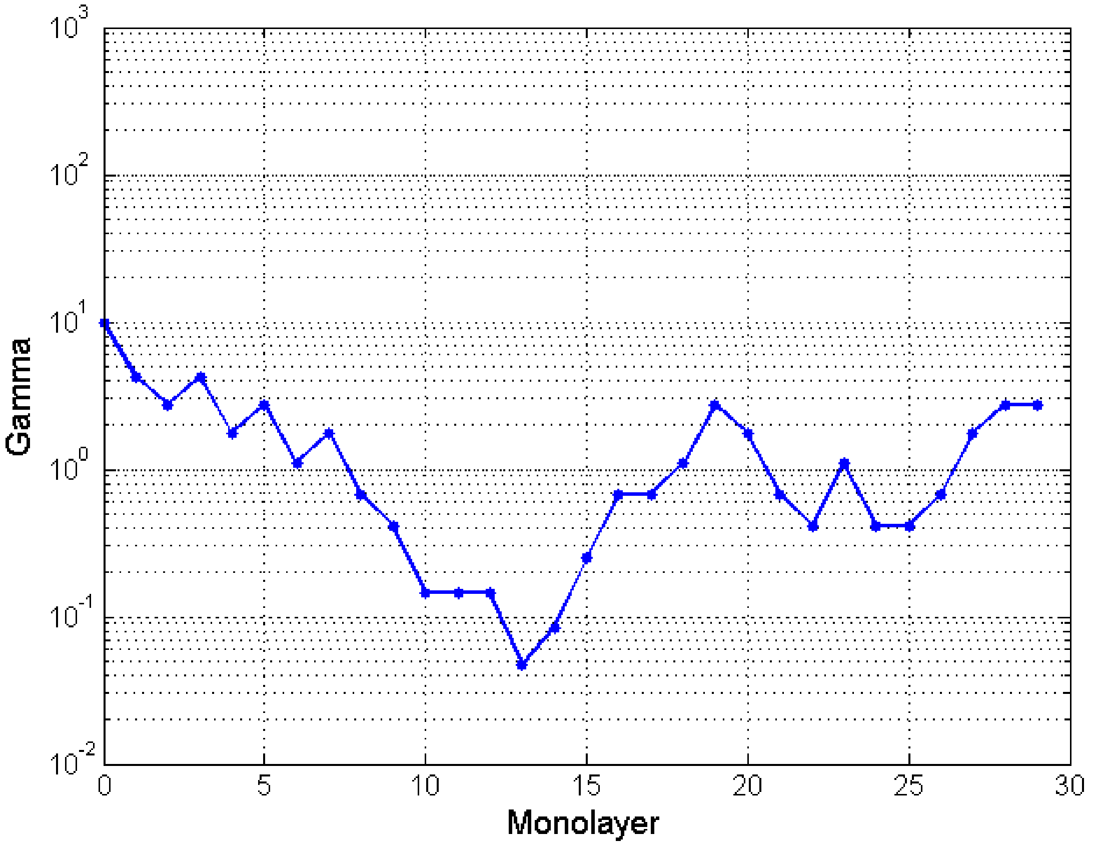


Figure 567: Profile of γ vs. Monolayer for the open-loop optimizer. Initial γ was 10. Temperature, which implicitly changes γ, was allowed to change by 20 K every monolayer. The deposition rate was 2 monolayers per second.

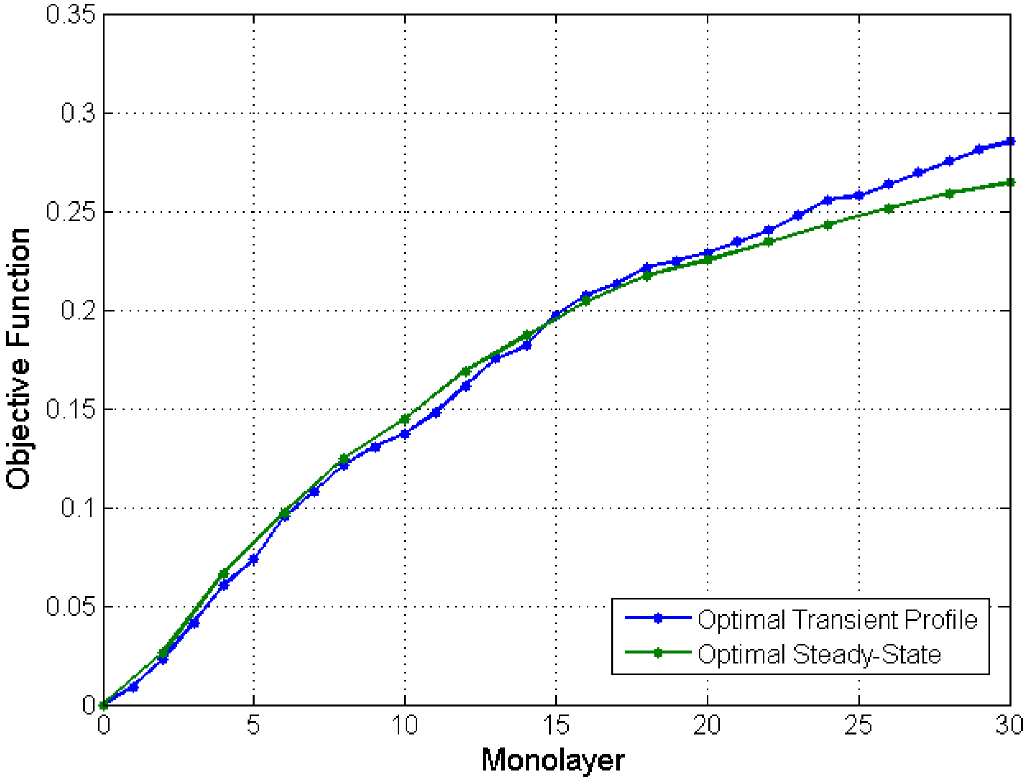


Figure 578: Profile of Objective Function vs. Monolayer for the open-loop optimizer. Initial γ was 10. Temperature, which implicitly changes γ, was allowed to change by 20 K every monolayer. The deposition rate was 2 monolayers per second. Weightings for the objective function were 6% roughness and 94% porosity. The open-loop optimizer did no better than the optimal steady-state temperature.

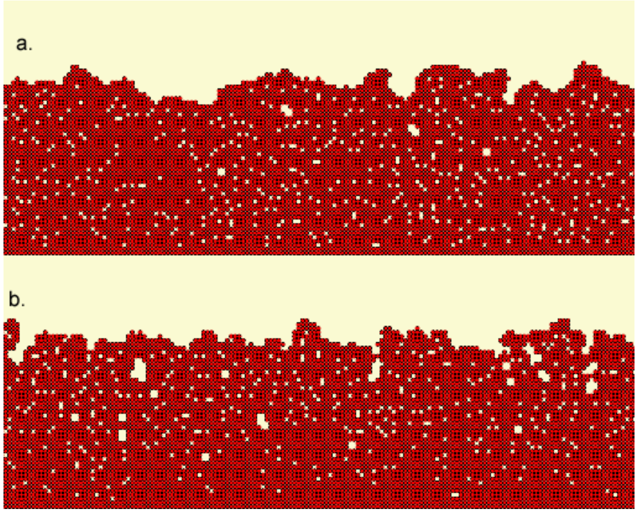


Figure 589: Film images of Material 2, with a square lattice, at 30 monolayers deposited. Zoom 3X. a: Optimal Steady-State. b: Optimal Transient Profile.

7.3 Discussion of Open Loop Optimizer Results

It is unclear why the optimal transient profiles of Material 1a and Material 2 had γ stay approximately constant while the optimal transient profile of the base case material had γ increasing. It is not possible to tell from the steady-state profiles how a material will change its temperature during the Open-Loop Optimizer program. It is possible that different materials simply exhibit different behavior in regard to the γ profile of their optimal transient pathway. Another explanation is that the full range of γ was not explored in the steady-state analysis of Material 1, due to physical limitations that were imposed upon the deposition rate and the temperature, while Material 2 was explored more fully. Thus, the value of γ that gave a universal minimum was found for the objective function for Material 2, while it is possible that only a local minimum was found for Material 1. It is very possible that had higher ranges of γ been explored for Material 1, the objective function might have been found to be declining indefinitely for larger values of γ. The implications of this would be that for some materials the objective function has a minimum at a finite value of γ, while for other materials the objective function always decreases with increasing values of γ. For the latter type of materials, another parameter, such as growth rate, would have to be added to the objective function in order for a control problem to exist. As stated earlier, without being able to model desorption this becomes difficult. Thus, the fully developed controller may need to use a simulation program that is not SPPARKS.

8. Material in 3D

8.1. Binding Model

Material 3 was again loosely modeled after a GaAs binding model, this time in 3 dimensions. The values of the bond energy, diffusion energy barrier, pre-exponential frequency factor, deposition rate, and temperature ranges are presented in Table 7 below. Second-nearest neighbor hops (Schwoebel hops) were allowed in this model. Binding energy was modeled as a linear function of the number of bonds. Desorption was not allowed. In order to maintain temperature and growth rate within a reasonable range, γ was constrained to values on the order of 1 to 10^5. A simple cubic lattice with 6 neighbors per site was used. Figure 59 below shows the lattice structure and neighbor associations. Due to computational power constraints, an 80x80 lattice size was used, and 20 monolayers were deposited.

Common Parameters		GaAs	Material 1
Single Bond Energy [eV]	E_{bond}	0.27	0.27
Nearest Neighbor Diffusion Barrier [eV]	Q_1	1.82	1.58
Schwoebel Hop Diffusion Barrier [eV]	Q_2	-	1.89
Diffusion Frequency Factor	k_{des}	$5.8*10^{13}$	10^{13}
Deposition Rate [monolayer/sec]	r_{dep}	-	1-4
Temperature [K]	T	-	500-1000

Table 7. Material 3 KMC Algorithm Parameters

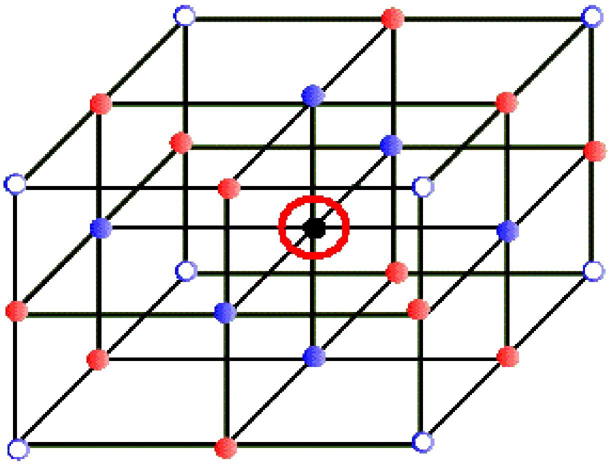


Figure 59. Simple Cubic Lattice Structure with 6 Neighbors. Black lines connect nearest neighbors. Consider the particle circled in red: 6 blue-colored particles are its nearest neighbors. 12 red-colored particles + additional 6 particles in the planes above, below, north, south, east, and west (not shown) make up 18 second-nearest neighbors.

8.2 Material 3 Characterization

To characterize the behavior of this material, simulations were run at several different γ values, depositing 20 monolayers in each simulation. Figures 60 through 71 show surface profiles, and Figure 72 shows cross-sectional areas of the film (YZ-plane) at x=40, after deposition of 20 monolayers at different γ values. Similarly to the base case material, the Material 3 surface has a sinusoidal profile in the x- and y- directions; in this case, with increasing wavelength and decreasing amplitude as γ increases. Note that the surface profiles' color scale varies with the difference between the two extreme colors held constant. Materials with higher porosity have greater height after the same amount of deposition.

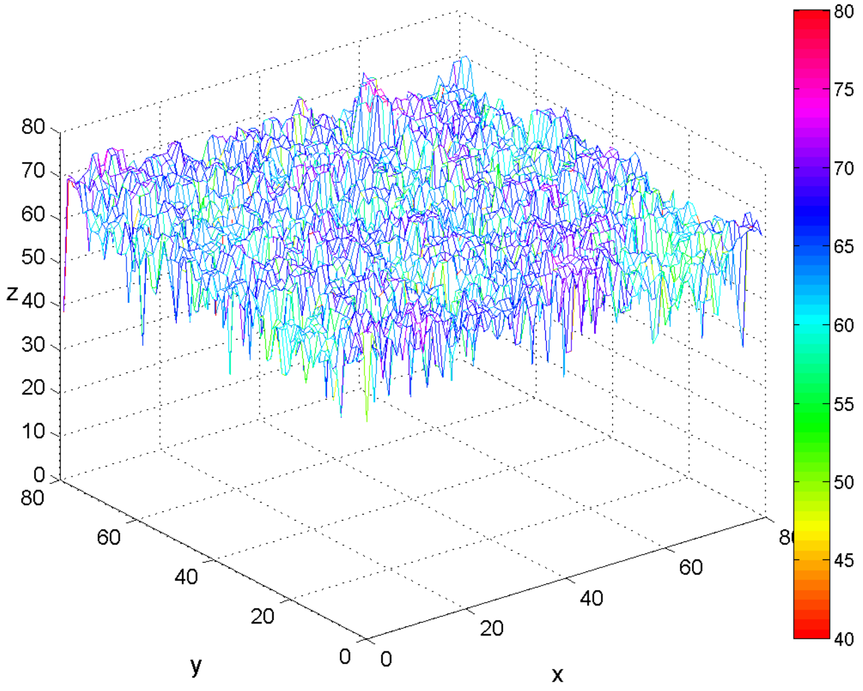


Figure 60. Surface Profile, 20 monolayers deposited; γ = 1

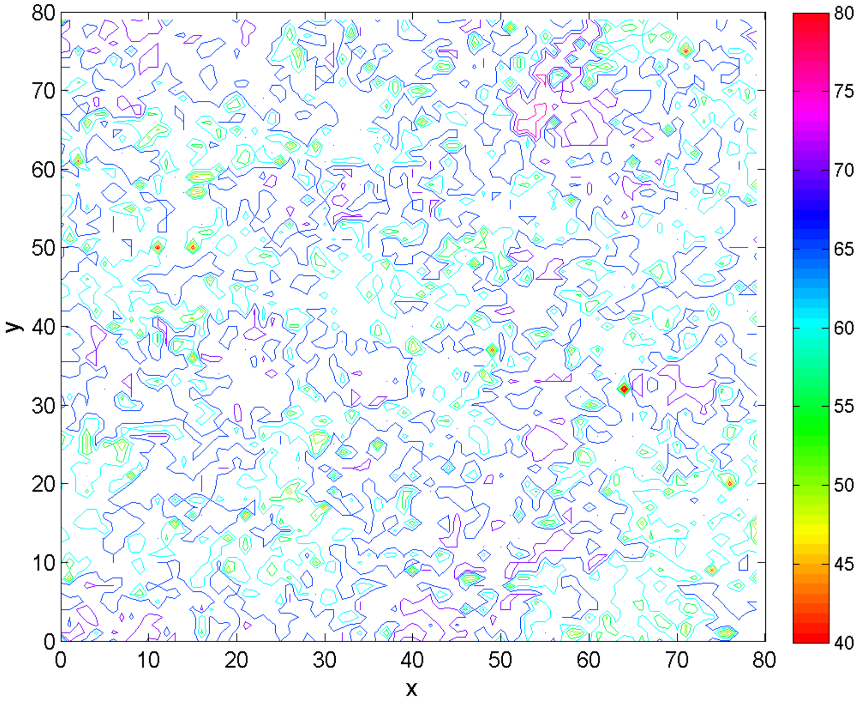


Figure 6110. Contour Plot, 20 monolayers deposited; γ=1

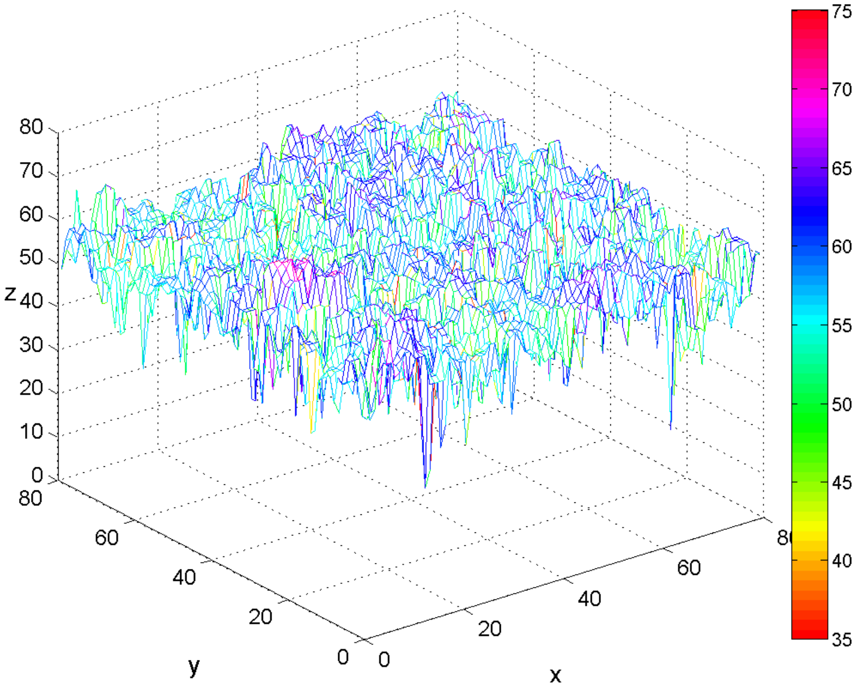


Figure 62. Surface Profile, 20 monolayers deposited; γ = 10

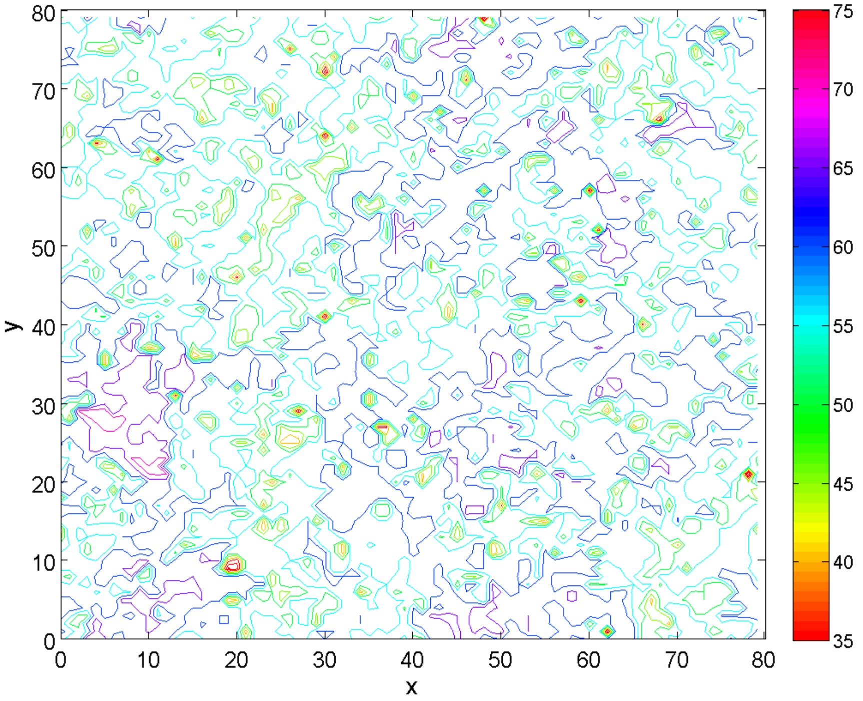


Figure 6311. Contour Plot, 20 monolayers deposited; γ=10

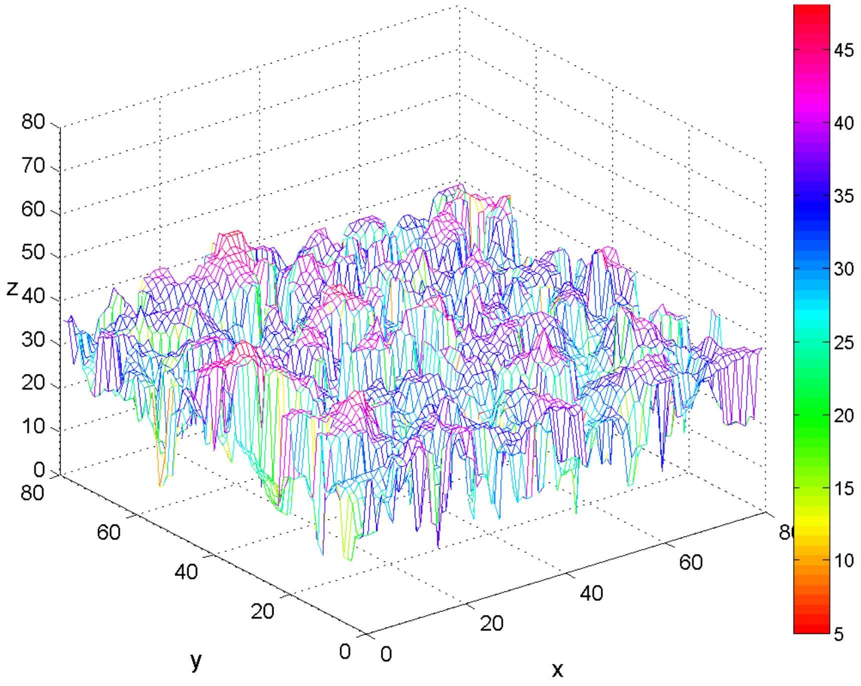


Figure 6412. Surface Profile, 20 monolayers deposited; $\gamma = 10^2$

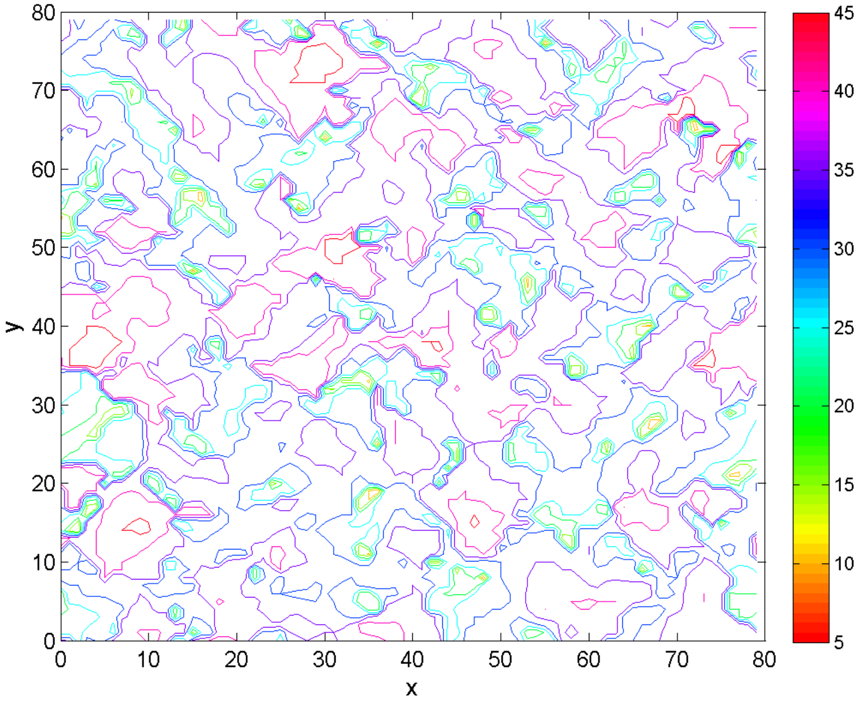


Figure 6513. Contour Plot, 20 monolayers deposited; $\gamma = 10^2$

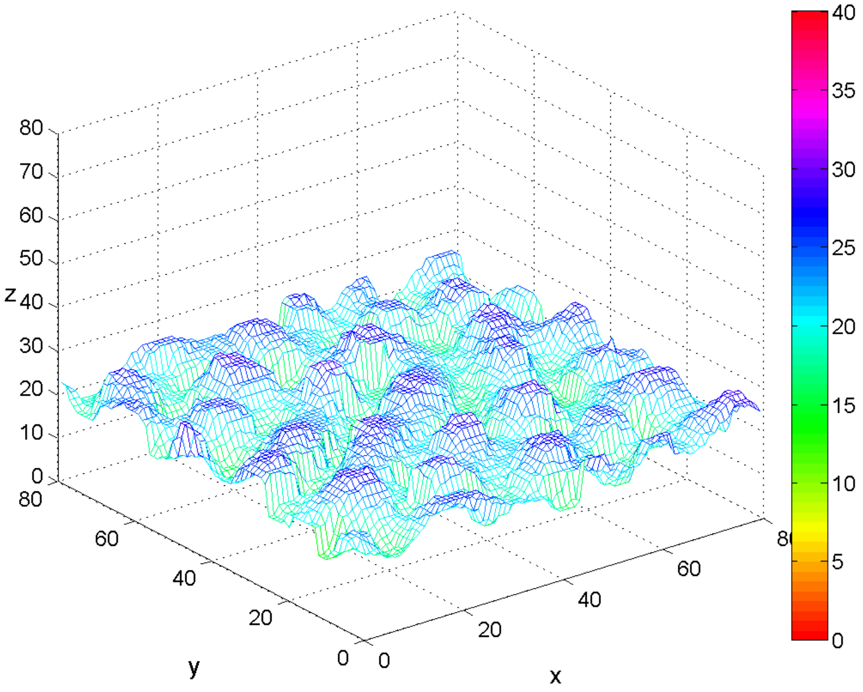


Figure 6614. Surface Profile, 20 monolayers deposited; $\gamma = 10^3$

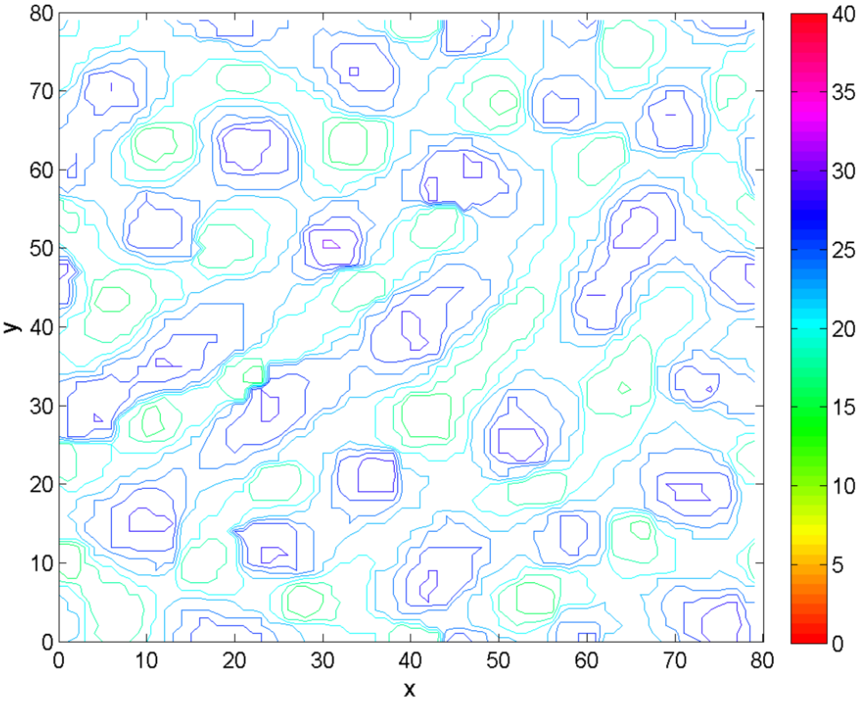


Figure 6715. Contour Plot, 20 monolayers deposited; $\gamma = 10^3$

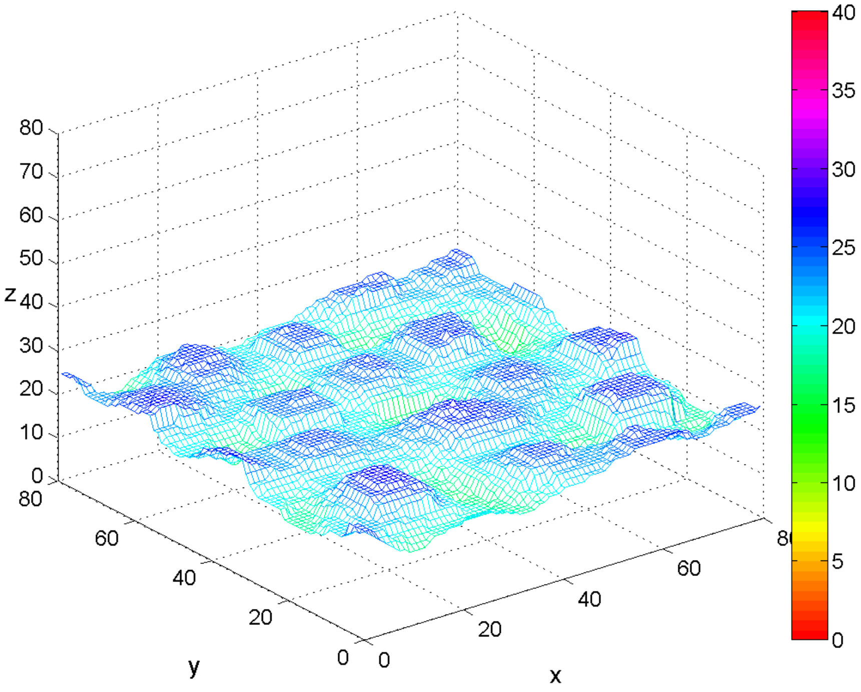


Figure 6816. Surface Profile, 20 monolayers deposited; $\gamma = 10^4$

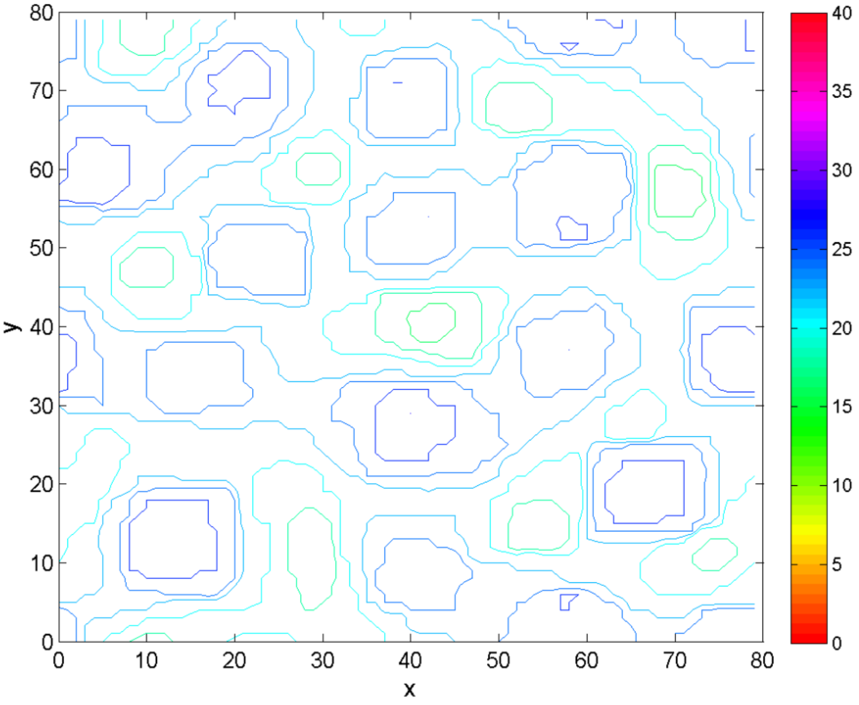


Figure 69. Contour Plot, 20 monolayers deposited; $\gamma = 10^4$

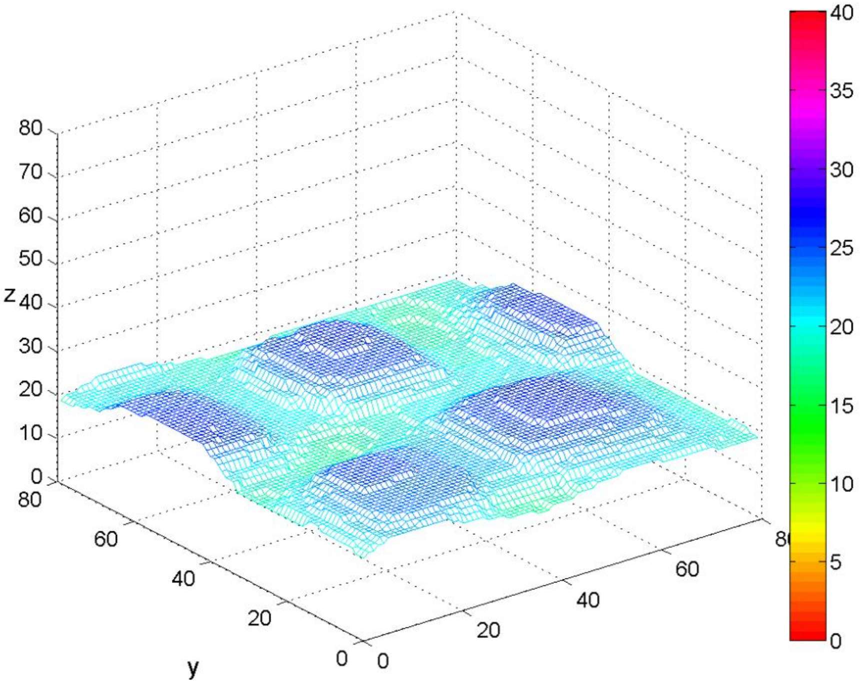


Figure 7017. Surface Profile, 20 monolayers deposited; $\gamma = 10^5$

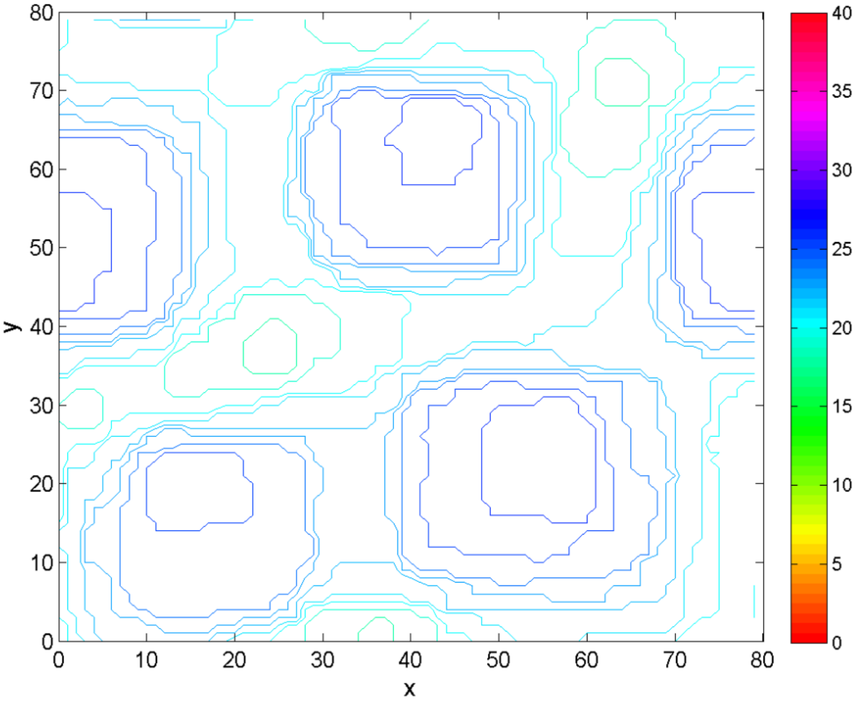


Figure 7118. Contour Plot, 20 monolayers deposited; $\gamma = 10^5$

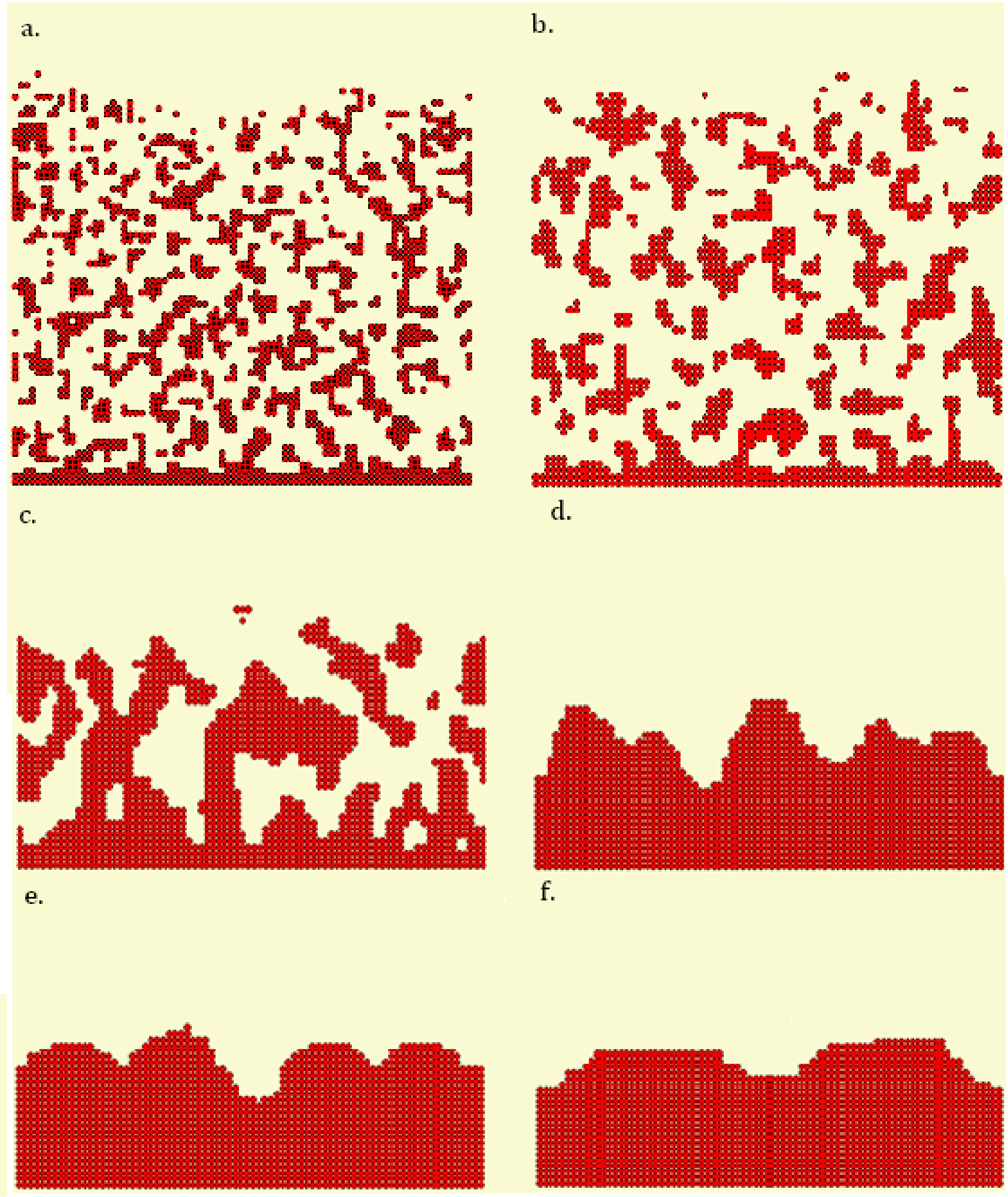


Figure 7219. Film Porosity. Cross-section snapshot of the YZ-plane at x=40. 20 monolayers deposited: a. γ=1; b. γ=10; c. γ=10^2; d. γ=10^3; e. γ=10^4; f. γ=10^5.

For values of γ ranging from 1 to 10^5, the time evolution of roughness is plotted in Figure 73. At low values of γ (below 10^2) the roughness features are quite pronounced and highly irregular; however, they are on a scale of several surface sites. The roughness filter, with a segment size of 10, deemphasizes roughness on this scale, so the roughness profiles at γ of 1 and 10 are relatively low. At γ of 10^2, roughness features begin to resemble wave-like structures, with high amplitudes, on a scale large enough to be detectable by the roughness filter. As γ increases further, the roughness features become more regular and decrease in amplitude. Roughness reaches a minimum when γ is on the order of 10^3. As the wavelength of the surface waves increases, such that the segment size becomes an increasingly smaller fraction of it, the deviation of the average segment height from the global average becomes larger. This results in increased roughness, since roughness is a direct measure of the standard deviation of the average segment size. Therefore, the roughness profiles at values of γ greater than 10^3 begin to increase. It should be noted, however, that the amplitude of the surface waves decreases as the wavelength increases, which may cause a decrease in roughness at high enough values of γ, when the effects of the decreasing amplitude overpower the effects of the increasing wavelength.

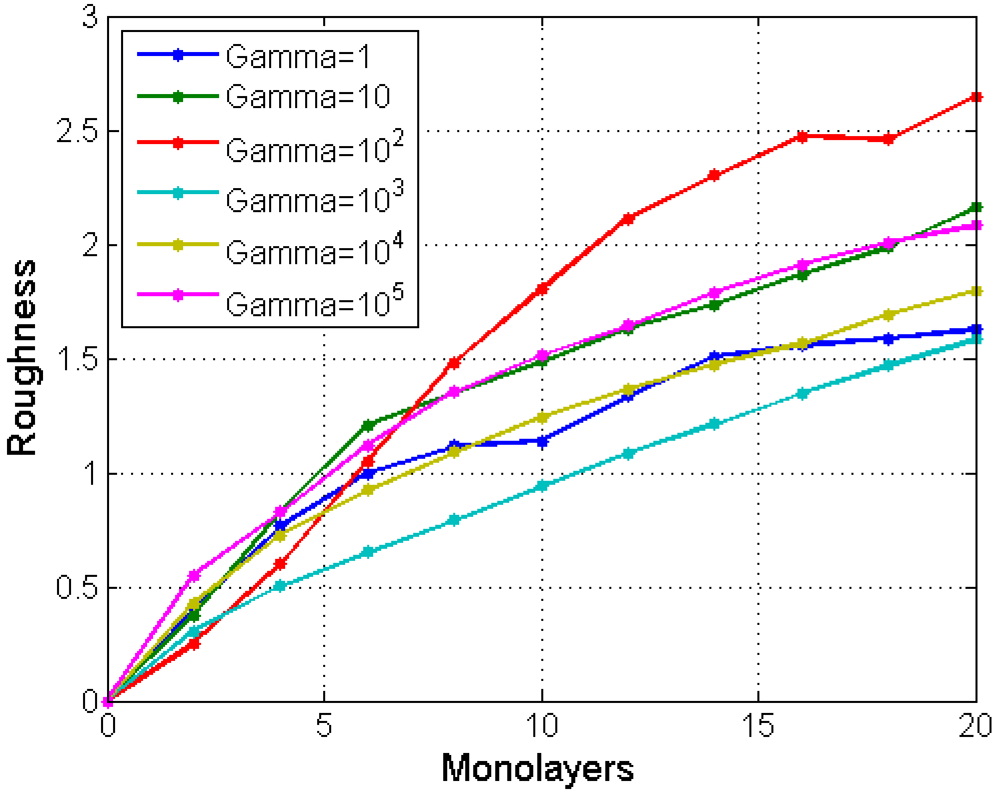


Figure 73. Time evolution of Roughness at different values of γ

Figure 74 shows the time evolution of porosity at different γ values. To visualize porosity, 2 dimensional images of vertical cross-sections with a thickness of one particle were shown in Figure 72. These do not contain complete information about porosity, as they only capture the porosity of a fraction of the film. They are intended to serve as visual complements to the porosity profiles ploted in Figure 74. As expected, similarly to the base case material, porosity decreases with increasing γ, but it does so much more rapidly. It can be seen, both from the porostiy profile plots and the film images, that porosity is virtually zero for orders of γ higher than 10^2.

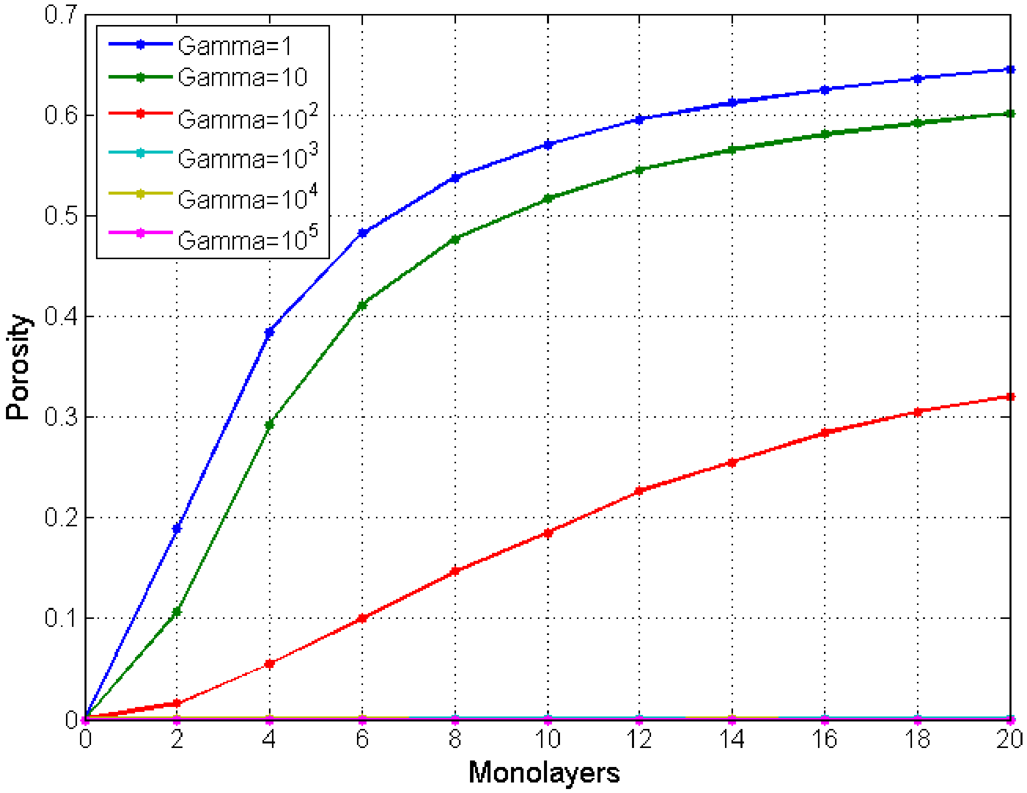


Figure 7420. Time evolution of Roughness at different values of γ

8.3 Optimum Steady-State Profile

An objective function for this film, constructed such that it places equal weightings on roughness and porosity, is plotted in Figure 75 below. Since porosity is high at low values of γ, the porosity weighting factor results in a high objective function at values of γ on the order of 10^2 or lower. At high values of γ, porosity is negligible but roughness increases, resulting again in a high objective function. When $\gamma = 10^3$, roughness reaches a minimum, and porosity is equal to zero. The optimum steady-state profile, therefore, corresponds to the objective function profile at $\gamma=10^3$. It should be noted that this is the optimum steady state profile only within the range of γ values considered. As γ increases to high enough values, roughness may begin to decrease as a function of γ.

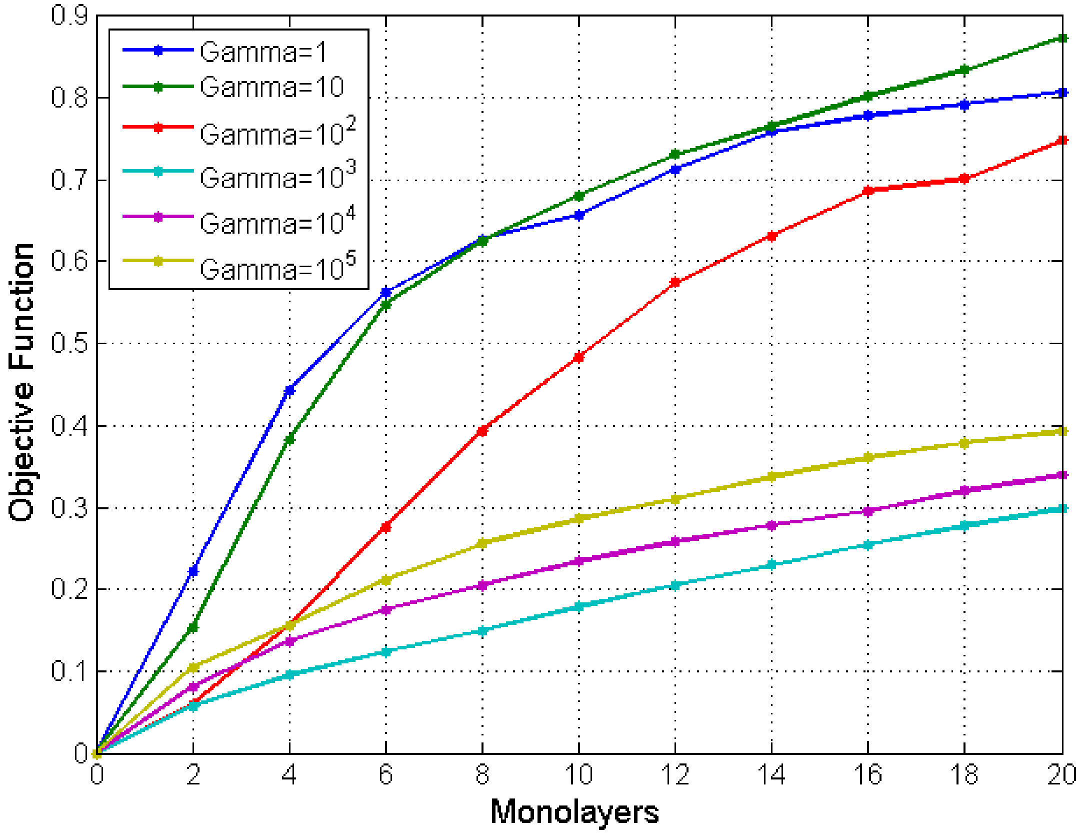


Figure 75: Objective Function Profiles at Steady-State for Material 3.

8.4 Optimal Transient Profile

The optimal transient profile for material 3 was obtained by running the open loop optimizer on a 80x80 lattice, starting at the best steady state γ value of 10^3, determined in section 11.3. Five temperatures in a range of 20 K above and below the system temperature were sampled for the predictive runs. Three predictive runs were averaged at every temperature. Figure 76 compares the optimal transient profile and the optimal steady state profile. The transient objective function is consistently lower than the best steady-state objective function.

Figure 77 plots the time evolution of γ for the optimizer run. It describes the controller action at every time step.

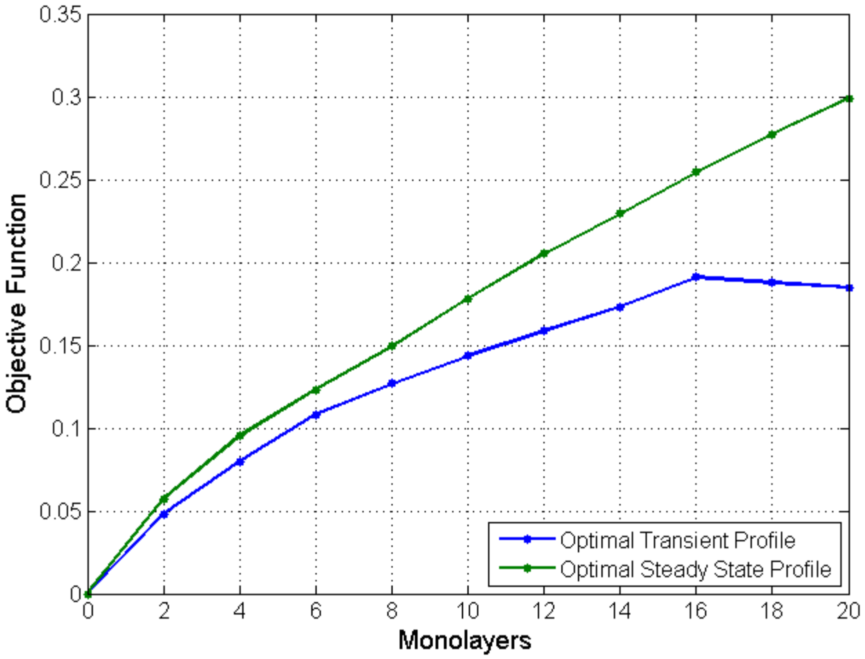


Figure 76: Objective Function Profile for Material 3. The optimal transient profile was better than the optimal steady-state profile.

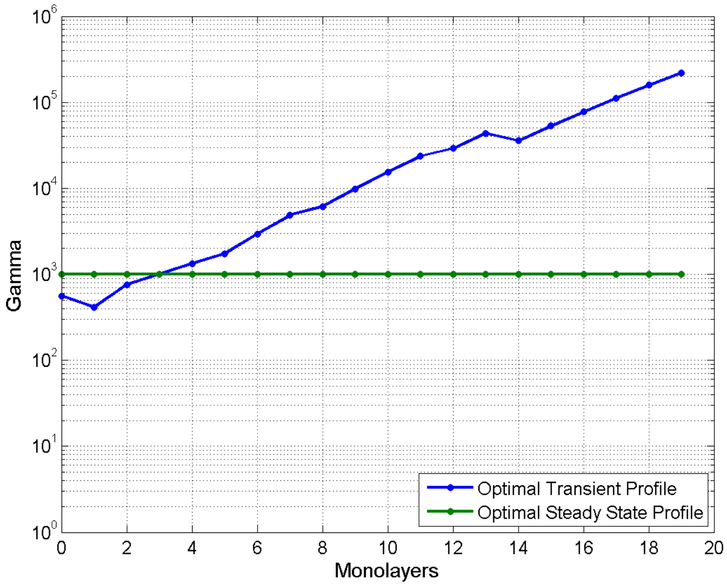


Figure 77: Gamma profile for Material 3. The open-loop optimizer consistently increases gamma, much as it did for the base case material.

It is evident from the plot that the effect of the controller is to continually increase γ. As was the case for the base case material, this is contrary to the expected result of the controller action, which is to keep γ at an intermediate value since at steady state values of γ a minimum was found at $\gamma=10^3$. Similarly to the base case material, since steady state profiles were obtained for a limited range of γ values which were deemed physically realistic, it is possible that the minimum at $\gamma = 10^3$ is only a local minimum. The controller is allowed to increase γ indefinitely, so it would be able to find a value of γ which results in a lower objective function. Figure 78 shows the surface profile after 20 monolayers have been deposited at the optimal transient γ, which is indeed smoother than the surface profile at $\gamma = 10^3$ (Figure 66 on page 88).

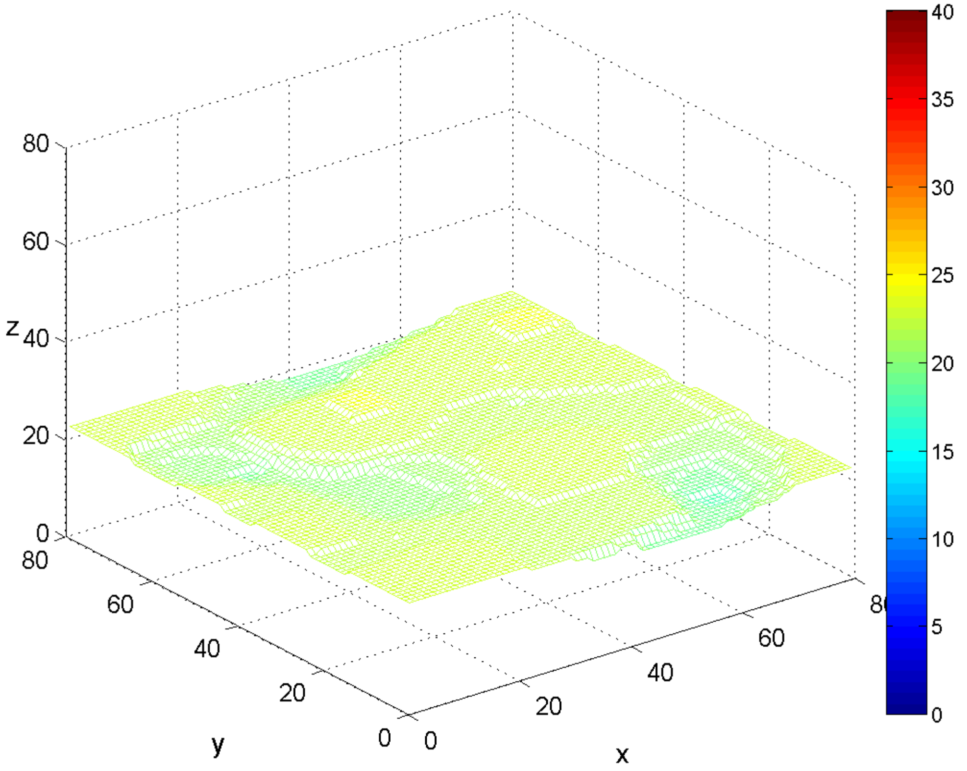


Figure 78: Surface Profile, 20 monolayers deposited; optimal transient run.

9. Financial Analysis

9.1 Start-Up Technology Company Goals

The company's product will be a control system software package. Before analyzing the market in which the product would be launched, an exploration of the goals of a start-up is in order. Technology in the chemical and physical vapor deposition markets is changing quickly; technologies that get introduced often have a lifetime of just a few years before they get replaced by a newer and better technology. With that in mind, the company expects that the initial software product will have a lifecycle lasting just five years.

With this short of a product lifecycle, the startup company would have to invest into Research and Development to prepare for the next generation of product. This money can be obtained in several ways: namely, by being acquired by a larger company, going public, or having a large influx of cash flows shortly after launch. The third option is clearly the most optimal, but for the company to be successful this cannot be assumed to be the case. Thus, the company will be focused on obtaining customers as soon as possible to demonstrate to the acquisition market and to the public market that it is worth investing in.

9.2 Thin Films Market Overview

The most important application of a system that controls a vapor deposition process for roughness, porosity, and throughput is the thin films market, particularly for photovoltaics. It is crucial that thin film solar cells have the correct roughness and porosity in order to have high conversion of solar energy and good mechanical stability. Other chemical vapor deposition and physical vapor deposition processes would benefit from such a controller, but companies that use such processes for purposes that are not thin-films do not have as much incentive to accurately control for surface microstructure; many of these companies make thick wafers in which the bulk is more important than the surface. Additionally, these wafer-making processes are more fully developed than the thin films market, so controllers that do not need to model the surface morphology may already be commercially available.

The thin films market data is presented in Table 8 and Figure 79 below. As one can see, the vast majority of thin film shipments are for photovoltaic applications, and this segment is expected to grow at a Compound Annual Growth Rate (CAGR) of 23.6% over the next several years. $3.3 billion dollars is expected to be spent in 2013 on photovoltaic thin films.

Type	2007	2008	CAGR % 2007-2008	2013	CAGR% 2008-2013
Photovoltaics	916.4	1160.9	26.7	3344.2	23.6
Fuel Cells	82.5	98.7	19.6	301.0	25.0
Batteries	36.0	39.2	8.9	98.1	20.1
Nuclear	25.2	25.9	2.8	31.3	5.0
Concentrating Solar Power	14.7	23.4	59.2	93.0	31.8
Geothermal	2.7	3.0	11.1	5.3	12.1
Total	1077.5	1351.1	25.4	3874.7	23.5

Table 8: Projected global shipments of thin films for energy, through 2013 ($ millions). Source: BCC Research.

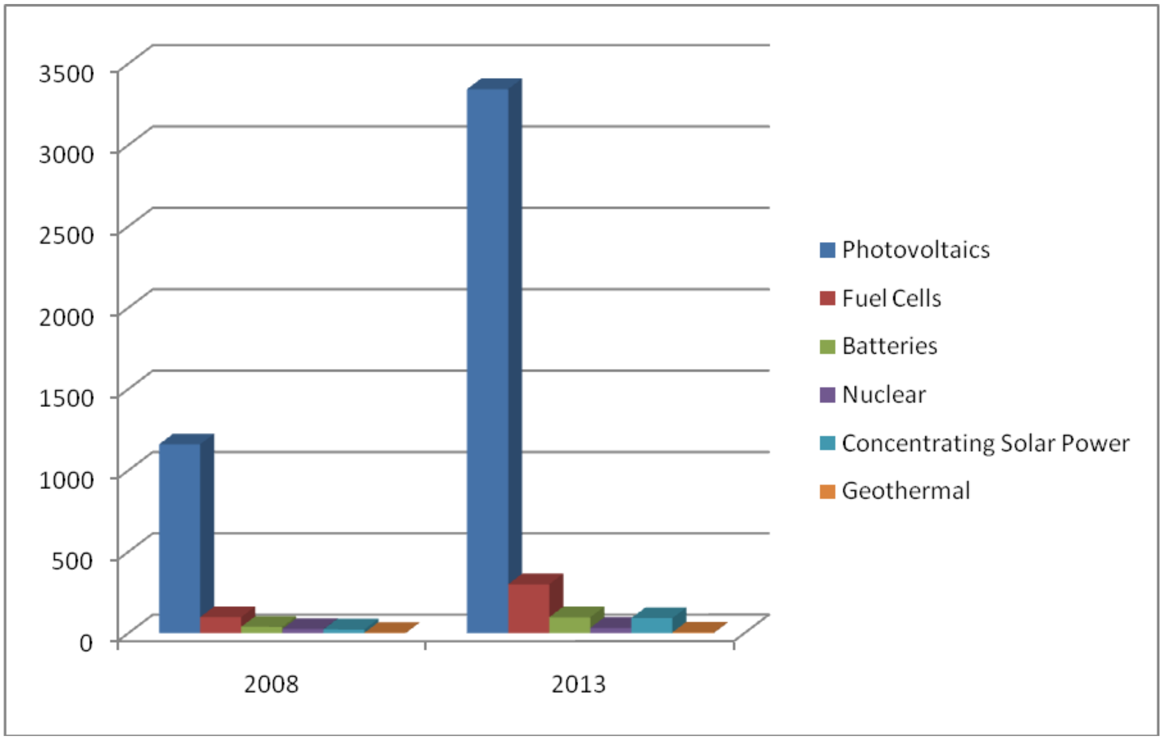


Figure 7921: Projected global shipments of thin films for energy, through 2013 ($ millions). Source: BCC Research.

In addition to deciding on the market sector, a geographic segment is a crucial factor in determining a young company's success. Data on global thin film photovoltaic manufacturing is presented in Table 9 and Figure 80 below. China and Asia/Pacific (largely Japan) seem to be the best markets, but succeeding in these markets would require a cross-national sales team that a small startup would be unable to initially afford. Therefore, the initial target market would be North America, which makes up no small portion of the total market. If the company were to launch by 2013, it would be entering a market that is 3344.2 million dollars large in total, with 18.3% being in North America. This comes to 612 million dollars in the market. However, much of this is spent on raw materials, equipment, and other non-software related supplies. A better estimate of the target market is given in Section 9.3.

Region	2007	2008	2013
Asia/Pacific	39.5	37.6	30.7
Europe	23.8	23.5	22.0
North America	16.7	16.9	18.3
Rest of World	11.0	10.7	8.9
China	9.0	11.3	20.1
Total	100.0	100.0	100.0

Table 9: Global thin film PV manufacturing by region, 2007–2013 (%). Source**:** BCC Research.

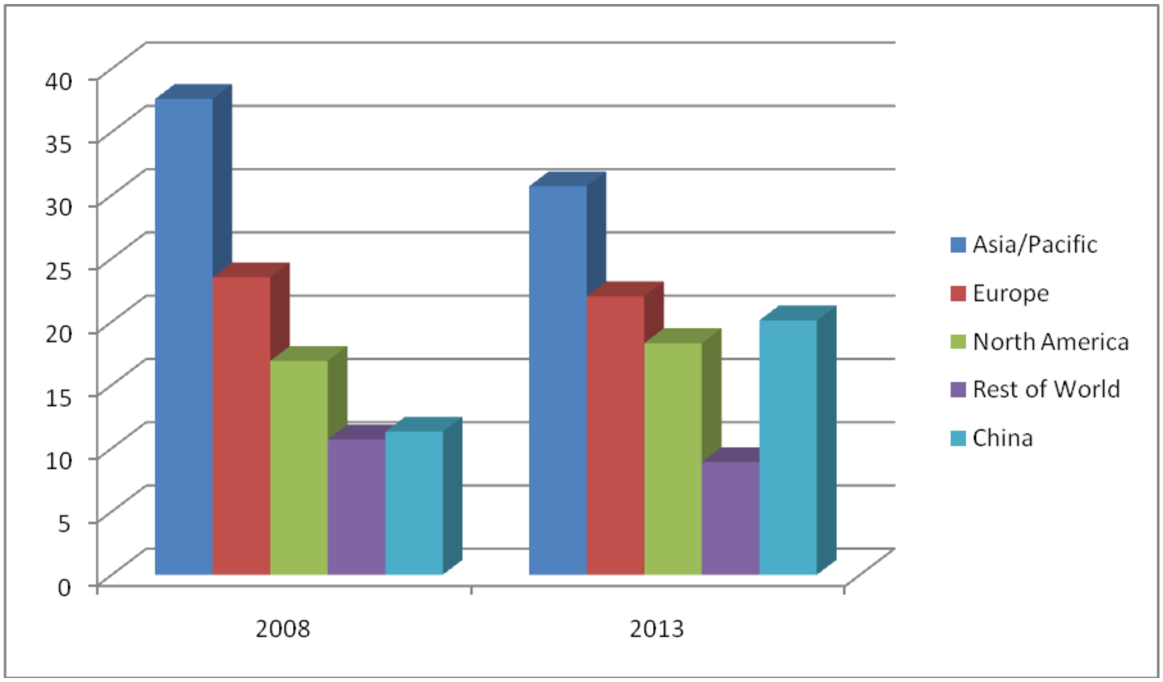


Figure 8022: Global thin film PV manufacturing by region, 2007–2013 (%). Source: BCC Research.

9.3 Competitive Environment

Several companies exist that provide for simulations of chemical vapor deposition, with many of them using the Monte Carlo method. Some of these companies (like Applied Materials) also offer control solutions for the photolithographic part of the chip fabrication process. However, of the companies investigated (Applied Materials, KLA-Tencor, STR Group, and Synopsys, Inc.), none seem to offer control solutions for the thin film deposition process itself. Thus, this control systems software seems to have found a niche that may be relatively unexplored.

However, upon introducing this controls software to the market, it is fully expected that companies that have existing simulation solutions on the market will quickly develop control systems themselves. Thus, it is crucially important for the company to gain a first-mover advantage in the new market it is opening up.

One company investigated, STR Group, provides software services focused on the modeling of crystal growth. One product, called CVDSim, models epitaxial growth in mass-production and research scale reactors. An annual single-node license of this software costs $11,500. STR Group’s product will not be competitive with the designed product, but this cost will later serve as a comparison.

An estimate of the number of systems that the company could be targeting can be derived from Applied Materials's 10-K, filed in December, 2010. It states that there are over 34,000 systems in place at fabrication facilities and that 12% of their 2010 sales were in North America. 20% market share is estimated. This leads to the calculation:

$$\boldsymbol{market\ size = \frac{34{,}000\ installed\ systems * 12\%\ North\ America\ sales}{20\%\ market\ share} = 20{,}400\ total\ systems\ in\ North\ America}$$

Equation 20

This estimate of 20,400 systems in the company's target market will be used to develop a pricing model in the Business Model section XX.

9.4. Business Model

With the company's goals, the market overview, and the competitive environment explored, a detailed financial analysis of the venture can now be made. Due to limited data, several assumptions need to be made. First, it assumed that the company will have no sales in the first year, as much more Research and Development needs to be conducted on the controller before it can be made into a commercial product. Due to the short product lifecycle, it is assumed that the company's first product will be on the market for a total of five years. It is difficult to evaluate how much customers would be willing to pay for this product on a value basis; instead, it is assumed that a 20% internal rate of return will be needed for a buyout, and the product is subjected to several different growth models to find what price will be required. 3% inflation is assumed.

SPPARKS, at least in its current form, will not be used for simulations in the final software product. Two major problems with the SPPARKS application that used are that it does not allow for the modeling of desorption and it can only run on a single processor. The company will need to hire software developers to either code a new Kinetic Monte Carlo solver or extend the current SPPARKS code to meet the simulator's needs. SPPARKS is distributed under the terms of the GNU Public License, which means that "anyone is free to use, modify, or extend SPPARKS in any way they choose, including for commercial purposes." Legal counsel will have to ensure that this interpretation is correct, in addition to ensuring that the company will be capable of acquiring intellectual property for a SPPARKS-incorporated software package.

In order to ensure that the simulations of the deposition process accurately predict what happens in real reactors, the company needs to be able to operate a thin film deposition reactor. This laboratory equipment is expensive, but necessary. A clean room will be needed to perform procedures on the wafer surfaces. Costs for computers and measurement instruments were conservatively estimated, as well as operating costs. Quotes for the bare module costs are found in Appendices A1 and A2.

A breakdown of costs the company expects to incur follows.

Cost Breakdown

Bare Module/Capital Costs	Unit Price	Units	Quantity	Total
Clean Room	$ 345.00	/sq ft	1600	$ 552,000.00
Deposition Reactor	$ 300,000.00	/reactor	1	$ 300,000.00
Fume Hood	$ 2,000.00	/hood	5	$ 10,000.00
Glove Box	$ 8,730.00	/box	2	$ 17,460.00
Computers	$ 5,000.00	/pc	9	$ 45,000.00
Servers	$ 15,000.00	/server	3	$ 45,000.00
AFM	$ 100,000.00	/microscope	1	$ 100,000.00
Profilometer	$ 15,000.00	/unit	1	$ 15,000.00
Refractometer	$ 15,000.00	/unit	1	$ 15,000.00
				$ 1,099,460.00

Operating Cost	Unit Price	Units	Quantity	Total
Office Space	$ 20.00	/sq ft/yr	5000	$ 100,000.00
Supplies and Utilities	$ 2,000.00	/yr	1	$ 2,000.00
Reactor Maintenance (1% purchase)	$ 3,000.00	/seat/yr	1	$ 3,000.00
Clean Room Maintenance (2% purchase)	$ 11,040.00	/yr	1	$ 11,040.00
Cost of Sales (10% sales)				
				$ 116,040.00

Labor Cost	Unit Price	Units	Quantity	Total
Software Developer	$ 85,000.00	/person/yr	3	$ 255,000.00
Lab Coordinator	$ 85,000.00	/person/yr	1	$ 85,000.00
Lab Technician	$ 65,000.00	/person/yr	2	$ 130,000.00
Marketing	$ 85,000.00	/person/yr	1	$ 85,000.00
Implementation/Customer Service	$ 65,000.00	/person/yr	3	$ 195,000.00
CEO	$ 175,000.00	/person/yr	1	$ 175,000.00
VP-Legal	$ 150,000.00	/person/yr	1	$ 150,000.00
				$ 1,075,000.00

Recurring Fixed Costs	$ 1,191,040.00
Capital Costs	$ 1,099,460.00
Initial Investment	**$ 2,290,500.00**

Table 10: Cost Breakdown

The internal rate of return (IRR) is the interest rate at which the net present value (NPV) of all cash flows is zero. In the semiconductor industry, a common firm’s IRR is in the range of 15-30%. Under the assumption that an IRR of 20% is satisfactory, sales necessary to achieve this IRR were calculated under three different scenarios. A moderate sales scenario calls for an initial market penetration of 1% in the second year, followed by 50% sales growth. An aggressive sales scenario estimates the same initial market penetration, but with 100% sales growth. Finally, a conservative model calls for an initial penetration of 0.5% with a yearly increase of 0.5%. Straight-line depreciation with a five-year life is assumed. It is also assumed that the company will be able to use its losses in the first year to carry forward to the following years to reduce taxes. A tax rate of 35% is used. The cash flow summary for the moderate scenario follows. Summaries for aggressive and conservative scenarios are presented in Appendix A.3 and A.4.

Cash Flow Summary (Moderate)

Year	% Market	Sales	Capital Costs	Fixed Costs	Costs of Sales	Depreciation Allowance
1	0.00%	$ -	$(1,099,460.00)	$(1,191,040.00)	$ -	$ -
2	1.00%	**$ 1,348,799.66**	$ -	$(1,226,771.20)	$ (134,879.97)	$ (219,892.00)
3	1.50%	$ 2,083,895.47	$ -	$(1,263,574.34)	$ (208,389.55)	$ (219,892.00)
4	2.25%	$ 3,219,618.51	$ -	$(1,301,481.57)	$ (321,961.85)	$ (219,892.00)
5	3.38%	$ 4,981,679.94	$ -	$(1,340,526.01)	$ (498,167.99)	$ (219,892.00)

Year (cont.)	Taxable Income	Income Tax Calculation	Income Tax Costs	Net Earnings	Annual Cash Flow	Interest Adjusted
1	$(1,191,040.00)	$ 416,864.00	$ -	$(1,191,040.00)	$(2,290,500.00)	$(2,290,500.00)
2	$ (232,743.51)	$ 81,460.23	$ -	$ (232,743.51)	$ (12,851.51)	$ (10,709.59)
3	$ 392,039.59	$ (137,213.86)	$ -	$ 392,039.59	$ 611,931.59	$ 424,952.49
4	$ 1,376,283.09	$ (481,699.08)	$ (120,588.71)	$ 1,255,694.38	$ 1,475,586.38	$ 853,927.30
5	$ 2,923,093.94	$(1,023,082.88)	$(1,023,082.88)	$ 1,900,011.06	$ 2,119,903.06	$ 1,022,329.79
					NPV	**$ 0.00**

Inflation=	3%
Tax Rate=	35%
Interest Rate=	20%

Table 11: Cash Flow Summary

With the estimate that $1,348,800 in sales are needed in the first year to achieve an IRR of 20%, the price the company needs to charge per unit can be estimated under the three different sales scenarios. This follows in Table 12.

Sales Breakdown

Aggressive

Year	Total Market Units	% of Market	Units Sold	Price
1	20400	0.00%	0	
2	20400	1.00%	204	$ 3,825.88
3	20400	2.00%	408	$ 3,940.66
4	20400	4.00%	816	$ 4,058.88
5	20400	8.00%	1632	$ 4,180.65

Moderate

Year	Total Market Units	% of Market	Units Sold	Price
1	20400	0.00%	0	
2	20400	1.00%	204	$ 6,611.76
3	20400	1.50%	306	$ 6,810.12
4	20400	2.25%	459	$ 7,014.42
5	20400	3.38%	689.52	$ 7,224.85

Conservative

Year	Total Market Units	% of Market	Units Sold	Price
1	20400	0.00%	0	
2	20400	0.50%	102	$ 10,821.50
3	20400	1.00%	204	$ 11,146.14
4	20400	1.50%	306	$ 11,480.53
5	20400	2.00%	408	$ 11,824.94

Table 12: Sales Breakdown

Prices needed to achieve a 20% IRR range from $3,826 to $6,612 to $10,822. Based on the $11,500 current market cost of STR Group’s CVDSim software, industry seems willing to pay these prices. In fact, even higher prices might be justified since this software incorporates a control element in addition to simulations. Thus, an IRR of 20% or more is expected.

10. Conclusions

An analysis of the current thin films marketplace shows that a small start-up with a versatile modeling and control software product is poised to be competitive in growing thin film markets such as photovoltaics. The proposed product will improve control over thin film surface and internal microstructure. With evidence that companies are paying on the order of $12,000 for a one year license for similar software products, the proposed startup company can remain competitive until it is bought by selling each software license for $3,826, $6,612, or $10,822 per year, depending on the growth of sales. These pricings will allow the company to maintain a 20% IRR for a five year horizon.

The software will have a wide range of applicability as a result of being able to model surface roughness and porosity. The software will aim to maximize the control objectives in order to minimize surface and internal defects. As computational efficiencies increase, more rigorous process simulations can be performed while more computationally intensive property calculations may become more accessible. Such a computationally intensive property is pore size distribution, which can be a control problem of interest for the catalyst industry. The easily modifiable modeling, optimization and control parameters allow for versatility in the study of various systems.

The control system that was developed in this report demonstrates that model predictive control is an effective tool in controlling micro-structural properties of thin films. The developmental work done to produce optimal films demonstrates the value of modeling a stochastic process. Modeling makes the search for optimal operating conditions possible. The reference trajectory was used successfully by the model predictive controller to correct a disturbed film, resulting in a film with almost optimal microstructure. This proves that this product will provide potential clients with more advanced control of the thin film manufacturing process.

11. Acknowledgements

The authors would like to express their gratitude for the advice and guidance provided by Professor Leonard A. Fabiano, Dr. Warren Seider, and Dr. Talid R. Sinno. Professor Fabiano provided the group with general guidance regarding the writing and timeline of the report. Dr. Seider provided assistance in learning about effective control strategies and he taught the fundamental skills required to complete a successful product design project through our design course last fall. Dr. Sinno explained the details of the simulation process and discussed much of the science behind the control issues. Additionally, the design consultants provided useful suggestions and advice from an industrial point of view.

12. Bibliography

Applied Materials, Inc. . (2010). Retrieved Abril 11, 2011, from Edgar Online: http://yahoo.brand.edgar-online.com/displayfilinginfo.aspx?FilingID=7602548-940-541878&type=sect&dcn=0000950123-10-112754

Christofides, P. D., Armaou, A., Lou, Y., & Varshney, A. (2009). *Control and Optimization of Multiscale Process Systems.* (W. S. Levine, Ed.) New York: Birkhauser.Boston.Basel.Berlin.

D., N., & Christofides, P. D. (2005). Dynamics and Control of Thin Film Microstructure in a Complex Deposition Process. *Chemical Engineering Science , 60,* 1603.

Franssila, S. (2010). *Introduction to Micro Fabrication* (First ed.). West Sussex, United Kingdom: John Wiley & Sons, Ltd.

Hu, G., Orkoulas, G., & Christofides, P. D. (2009). Model Predictive Contol of Film Porosity inThin Film Deposition. *2009 American Control Conference,* (pp. 4797-4804). St. Louis.

Hu, G., Orkoulas, G., & Christofides, P. D. (2009). Modeling and Control of Film Porosity in Thin Film Deposition. *Chemical Engineering Science , 64,* 3668-3682.

Hu, G., Orkoulas, G., & Christofides, P. D. (2009). Regulation of Film Thickness, Surface Roughness and Porosity in Thin Film Growth Using Deposition Rate. *Chemical Engineering Science , 64,* 3903-3913.

Hu, G., Orkoulas, G., & Christofides, P. D. (2009). Simultaneous Regulation of Surface Roughness and Porosity in Thin Film Growth. *Ind. Eng. Chem. Res. , 48* (14), 6690–6700.

Hu, G., Orkoulas, G., & Christofides, P. D. (2009). Stochastic Modeling and Simultaneous Regulation of Surface Roughness and Porosity in Thin Film Deposition. *Ind. Eng. Chem. Res. , 48,* 6690-6700.

Huang, J., Hu, G., Orkoulas, G., & Christofides, P. D. (n.d.). Model Predictive Control of Film Surface Root-Mean-Square Roughness and Slop: Application to Thin Film Solar Cells.

Lou, Y., & Christofides, P. D. (2003). Feedback Control of Growth Rate and Surface Roughness in Thin Film Growth. *AICHE Journal , 49,* 2099.

Mattox, D. M. (1998). *Handbook of Physical Vapor Deposition (PVD) Processing.* Westwood: Noyes Publications.

Sudarshan, T. S., & Park, J. H. (Eds.). (2001). *Chemical Vapor Deposition.* Materials Park, OH: ASM International.

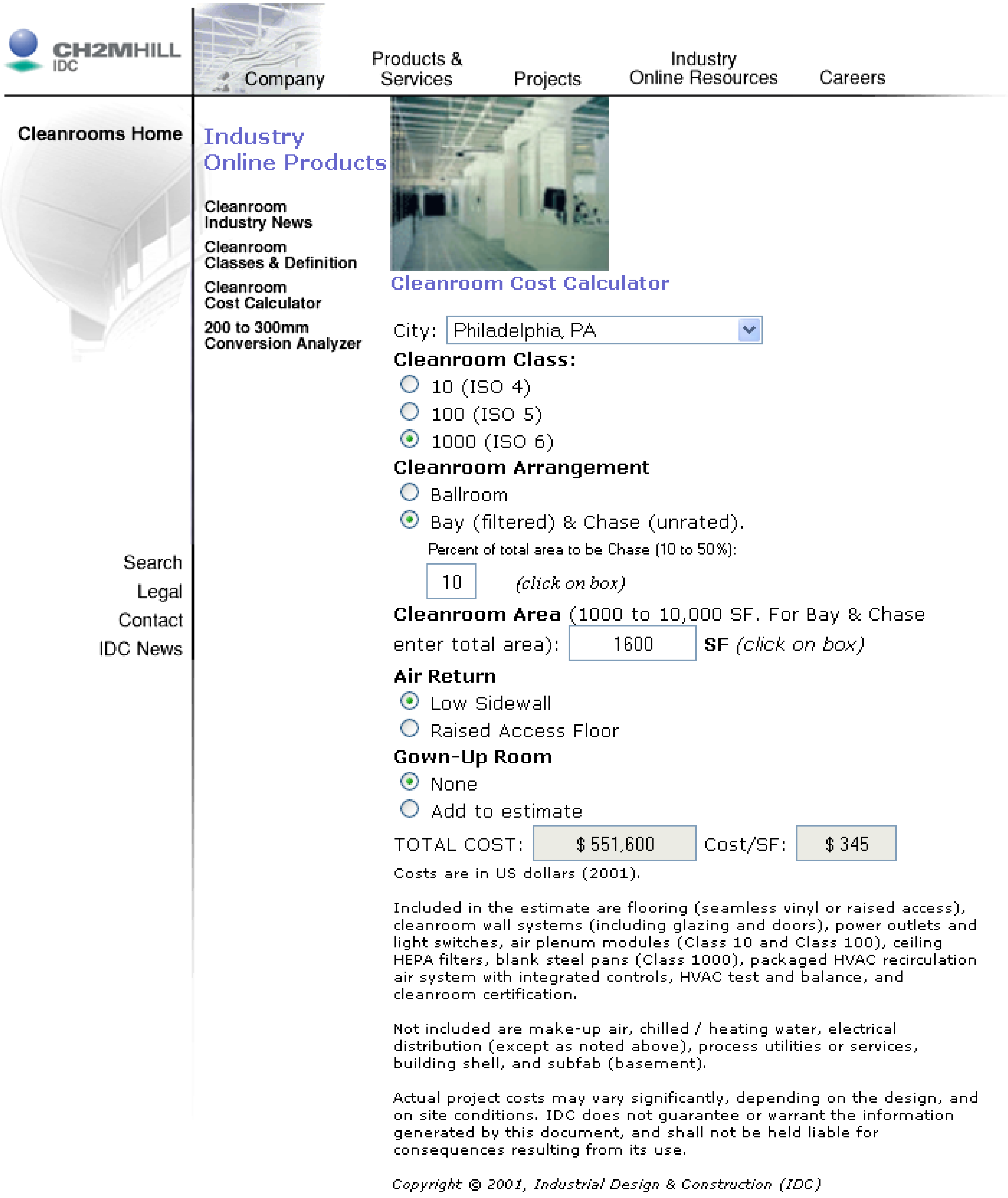
Willey, R. R. (2007). *Practical Monitoring and Control of Optical Thin Films* (2nd ed.). Michigan: Willey Optical, Consultants.

Zhang, X., Hu, G., Orkoulas, G., & Christofides, P. D. (2010). Predictive Contol of Surface Mean Slope and Roughness in a Thin Film Deposition Process. *Chemical Engineering Science* , *65*, 4720-4731.

Zhang, X., Hu, G., Orkoulas, G., & Panagiotis, C. D. (2010). Controller and Estimator Design for Regulation of Film Thickness, Surface Roughness, and Porosity ina Multiscale Thin Film Growth. *Ind. Eng. Chem.* , *49*, 7795-7806.

Appendix A Financial Details

A.1 Clean Room Quote



Cleanroom Cost Calculator

City: Philadelphia, PA

Cleanroom Class:

○ 10 (ISO 4)

○ 100 (ISO 5)

● 1000 (ISO 6)

Cleanroom Arrangement

○ Ballroom

● Bay (filtered) & Chase (unrated).

Percent of total area to be Chase (10 to 50%): 10 *(click on box)*

Cleanroom Area (1000 to 10,000 SF. For Bay & Chase enter total area): 1600 **SF** *(click on box)*

Air Return

● Low Sidewall

○ Raised Access Floor

Gown-Up Room

● None

○ Add to estimate

TOTAL COST: $ 551,600 Cost/SF: $ 345

Costs are in US dollars (2001).

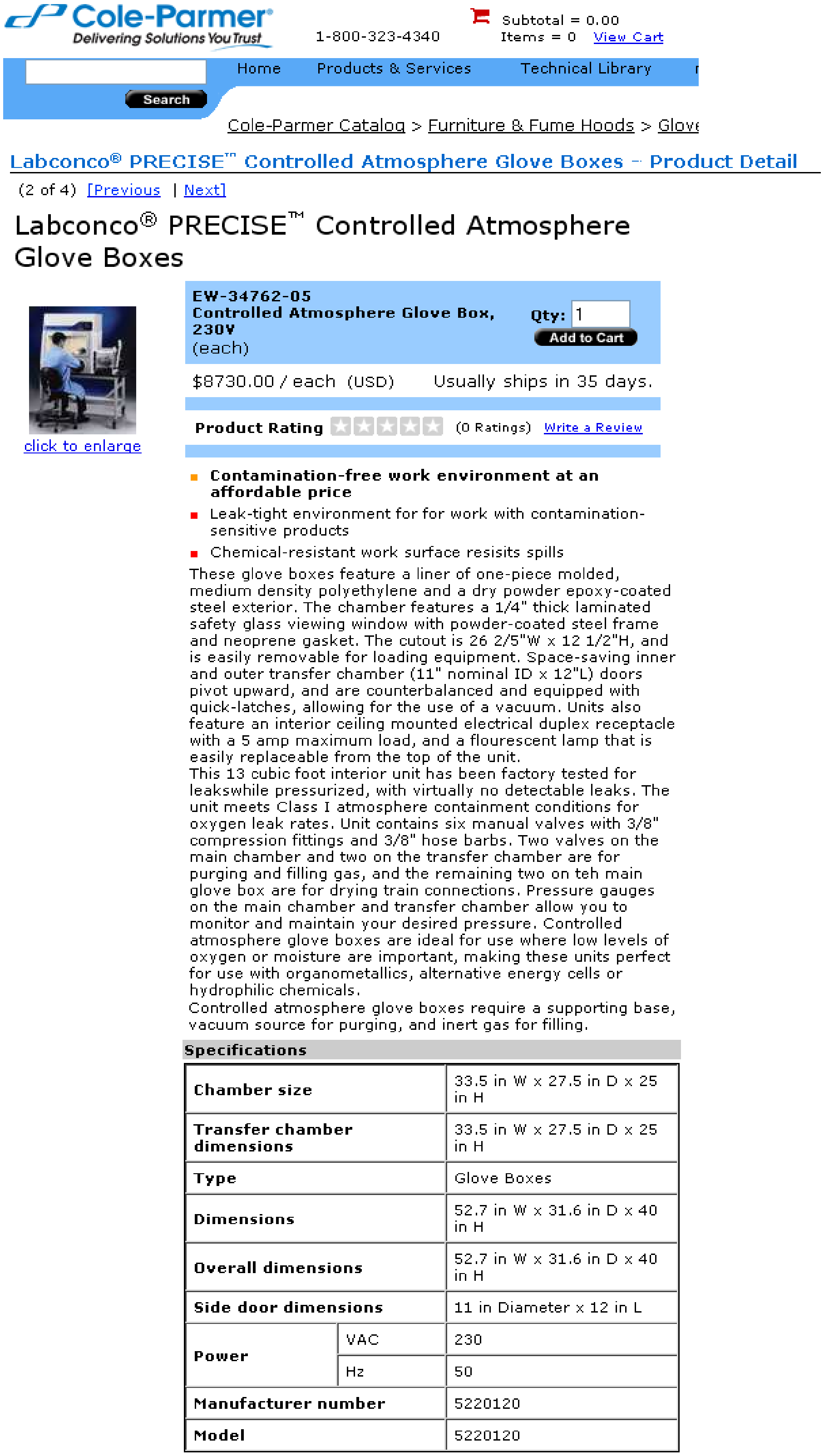
Included in the estimate are flooring (seamless vinyl or raised access), cleanroom wall systems (including glazing and doors), power outlets and light switches, air plenum modules (Class 10 and Class 100), ceiling HEPA filters, blank steel pans (Class 1000), packaged HVAC recirculation air system with integrated controls, HVAC test and balance, and cleanroom certification.

Not included are make-up air, chilled / heating water, electrical distribution (except as noted above), process utilities or services, building shell, and subfab (basement).

Actual project costs may vary significantly, depending on the design, and on site conditions. IDC does not guarantee or warrant the information generated by this document, and shall not be held liable for consequences resulting from its use.

Copyright © 2001, Industrial Design & Construction (IDC)

A.2 Glove Box Quote



Cole-Parmer Catalog > Furniture & Fume Hoods > Glove

Labconco® PRECISE™ Controlled Atmosphere Glove Boxes – Product Detail

(2 of 4) [Previous | Next]

Labconco® PRECISE™ Controlled Atmosphere Glove Boxes

EW-34762-05
Controlled Atmosphere Glove Box, 230V
(each)

Qty: 1

$8730.00 / each (USD) Usually ships in 35 days.

Product Rating ☆☆☆☆☆ (0 Ratings) Write a Review

- **Contamination-free work environment at an affordable price**
- Leak-tight environment for for work with contamination-sensitive products
- Chemical-resistant work surface resisits spills

These glove boxes feature a liner of one-piece molded, medium density polyethylene and a dry powder epoxy-coated steel exterior. The chamber features a 1/4" thick laminated safety glass viewing window with powder-coated steel frame and neoprene gasket. The cutout is 26 2/5"W x 12 1/2"H, and is easily removable for loading equipment. Space-saving inner and outer transfer chamber (11" nominal ID x 12"L) doors pivot upward, and are counterbalanced and equipped with quick-latches, allowing for the use of a vacuum. Units also feature an interior ceiling mounted electrical duplex receptacle with a 5 amp maximum load, and a flourescent lamp that is easily replaceable from the top of the unit.

This 13 cubic foot interior unit has been factory tested for leakswhile pressurized, with virtually no detectable leaks. The unit meets Class I atmosphere containment conditions for oxygen leak rates. Unit contains six manual valves with 3/8" compression fittings and 3/8" hose barbs. Two valves on the main chamber and two on the transfer chamber are for purging and filling gas, and the remaining two on teh main glove box are for drying train connections. Pressure gauges on the main chamber and transfer chamber allow you to monitor and maintain your desired pressure. Controlled atmosphere glove boxes are ideal for use where low levels of oxygen or moisture are important, making these units perfect for use with organometallics, alternative energy cells or hydrophilic chemicals.

Controlled atmosphere glove boxes require a supporting base, vacuum source for purging, and inert gas for filling.

Specifications

Chamber size		33.5 in W x 27.5 in D x 25 in H
Transfer chamber dimensions		33.5 in W x 27.5 in D x 25 in H
Type		Glove Boxes
Dimensions		52.7 in W x 31.6 in D x 40 in H
Overall dimensions		52.7 in W x 31.6 in D x 40 in H
Side door dimensions		11 in Diameter x 12 in L
Power	VAC	230
	Hz	50
Manufacturer number		5220120
Model		5220120

A.3 Cash Flow Summary

(Aggressive)

Year	% Market	Sales	Capital Costs	Fixed Costs	Costs of Sales	Depreciation Allowance
1	0.00%	$ -	$(1,099,460.00)	$(1,191,040.00)	$ -	
2	1.00%	**$ 780,480.36**		$(1,226,771.20)	$ (78,048.04)	$ (219,892.00)
3	2.00%	$ 1,607,789.54		$(1,263,574.34)	$ (160,778.95)	$ (219,892.00)
4	4.00%	$ 3,312,046.45		$(1,301,481.57)	$ (331,204.64)	$ (219,892.00)
5	8.00%	$ 6,822,815.68		$(1,340,526.01)	$ (682,281.57)	$ (219,892.00)

Year (cont.)	Taxable Income	Income Tax Calculation	Income Tax Costs	Net Earnings	Annual Cash Flow	Interest Adjusted
1	$(1,191,040.00)	$ 416,864.00	$ -	$(1,191,040.00)	$(2,290,500.00)	$(2,290,500.00)
2	$ (744,230.88)	$ 260,480.81	$ -	$ (744,230.88)	$ (524,338.88)	$ (436,949.06)
3	$ (36,455.75)	$ 12,759.51	$ -	$ (36,455.75)	$ 183,436.25	$ 127,386.28
4	$ 1,459,468.24	$ (510,813.88)	$ -	$ 1,459,468.24	$ 1,679,360.24	$ 971,851.99
5	$ 4,580,116.10	$(1,603,040.63)	$(1,423,750.20)	$ 3,156,365.90	$ 3,376,257.90	$ 1,628,210.79
					NPV	**$ 0.00**

Inflation=	3%
Tax Rate=	35%
Interest Rate=	20%

A.4 Cash Flow Summary

(Conservative)

Year	% Market	Sales	Capital Costs	Fixed Costs	Costs of Sales	Depreciation Allowance
1	0.00%	$ -	$(1,099,460.00)	$(1,191,040.00)	$ -	
2	0.50%	**$ 1,103,792.82**		$(1,226,771.20)	$ (110,379.28)	$ (219,892.00)
3	1.00%	$ 2,273,813.21		$(1,263,574.34)	$ (227,381.32)	$ (219,892.00)
4	1.50%	$ 3,513,041.41		$(1,301,481.57)	$ (351,304.14)	$ (219,892.00)
5	2.00%	$ 4,824,576.86		$(1,340,526.01)	$ (482,457.69)	$ (219,892.00)

Year (cont.)	Taxable Income	Income Tax Calculation	Income Tax Costs	Net Earnings	Annual Cash Flow	Interest Adjusted
1	$(1,191,040.00)	$ 416,864.00	$ -	$(1,191,040.00)	$(2,290,500.00)	$(2,290,500.00)
2	$ (453,249.66)	$ 158,637.38	$ -	$ (453,249.66)	$ (233,357.66)	$ (194,464.72)
3	$ 562,965.55	$ (197,037.94)	$ -	$ 562,965.55	$ 782,857.55	$ 543,651.08
4	$ 1,640,363.70	$ (574,127.29)	$ (195,663.86)	$ 1,444,699.84	$ 1,664,591.84	$ 963,305.46
5	$ 2,781,701.16	$ (973,595.41)	$ (973,595.41)	$ 1,808,105.76	$ 2,027,997.76	$ 978,008.18
					NPV	**$ 0.00**

Inflation=	3%
Tax Rate=	35%
Interest Rate=	20%

Appendix B SPPARKS Input Files

B.1 Base Case

```
log log.GaAs.1000wide.8493G.900K

#GaAs model from 2009 Senior Design report
#Energy and barrier values are in multiple of 0.27 eV, and temperature is in
#terms of T[K]*(kb[eV*K^-1]/0.27[eV]
#In other words, multiply temperature by 3133 to get Kelvin, and divide time
#by 10^13 to get seconds.
#Gamma = 52461

seed              325377

app_style         diffusion nonlinear hop

dimension         2
lattice           tri 1.0
region            simulationbox block 0 1000 0 100 -0.5 0.5
create_box        simulationbox
create_sites      box

barrier           hop 6

ecoord            0     100000000000000
ecoord            1     5
ecoord            2     4
ecoord            3     3
ecoord            4     2
ecoord            5     1
ecoord            6     0


set               site value 1
set               site value 2 if y < 10
set               site value 3 if y > 170
set               site value 3 if x = 1 if y = 0
set               site value 3 if x = 999 if y = 0
set               site value 3 if x = .5 if y < 1
set               site value 3 if x = 999.5 if y < 1

solve_style       linear

deposition        0.0000000002 0 -1 0 1 2 6
temperature       0.287

diag_style        diffusion stats yes

stats             10000000000000
dump              mydump 10000000000000 dump.GaAs.1000wide.8493G.900K

run               360000000000000.0
```

B.2 Material 1a with Schwoebel Hops

```
log log.GaAs.1000wide.8493G.900K

#GaAs model from 2009 Senior Design report
#Energy and barrier values are in multiple of 0.27 eV, and temperature is in
#terms of T[K]*(kb[eV*K^-1]/0.27[eV]
#In other words, multiply temperature by 3133 to get Kelvin, and divide time
#by 10^13 to get seconds.
#Gamma = 52461

seed                325377

app_style           diffusion nonlinear schwoebel 6 1

dimension           2
lattice             tri 1.0
region              simulationbox block 0 1000 0 100 -0.5 0.5
create_box          simulationbox
create_sites        box

barrier             hop 6
barrier             schwoebel 7

ecoord              0     100000000000000
ecoord              1     5
ecoord              2     4
ecoord              3     3
ecoord              4     2
ecoord              5     1
ecoord              6     0


set                 site value 1
set                 site value 2 if y < 10
set                 site value 3 if y > 170
set                 site value 3 if x = 1 if y = 0
set                 site value 3 if x = 999 if y = 0
set                 site value 3 if x = .5 if y < 1
set                 site value 3 if x = 999.5 if y < 1

solve_style         linear

deposition          0.0000000002 0 -1 0 1 2 6
temperature         0.287

diag_style          diffusion stats yes

stats               10000000000000
dump                mydump 10000000000000 dump.GaAs.1000wide.8493G.900K

run                 150000000000000.0
```

B.3 Material 2 with Square 8 Neighbor Lattice

```
log log.GaAs.1000wide.22G.700K

#GaAs model from 2009 Senior Design report
#Energy and barrier values are in multiple of 0.27 eV, and temperature is in
#terms of T[K]*(kb[eV*K^-1]/0.27[eV]
#In other words, multiply temperature by 3133 to get Kelvin, and divide time
#by 10^13 to get seconds.
#Gamma = 52461

seed              325377

app_style         diffusion nonlinear hop

dimension         2
lattice           sq/8n 1.0
region            simulationbox block 0 1000 0 150 -0.5 0.5
create_box        simulationbox
create_sites      box

barrier           hop 6

ecoord            0     100000000000000
ecoord            1     5
ecoord            2     4
ecoord            3     3
ecoord            4     2
ecoord            5     1
ecoord            6     0


set               site value 1
set               site value 2 if y < 13
set               site value 3 if y > 145
set               site value 3 if y < 2
set               site value 2 if x = 0 if y = 0

solve_style       linear

deposition        0.0000000002 0 -1 0 1 3 8
temperature       0.223428

diag_style        diffusion stats yes

stats             10000000000000
dump              mydump 10000000000000 dump.GaAs.1000wide.22G.700K

run               150000000000000.0
```

B.4 Base Case Material with Temperature Step Change

```
log log.GaAs.1000wide.unitstep

#GaAs model from 2009 Senior Design report
#Energy and barrier values are in multiple of 0.27 eV, and temperature is in
#terms of T[K]*(kb[eV*K^-1]/0.27[eV]
#In other words, multiply temperature by 3133 to get Kelvin, and divide time
#by 10^13 to get seconds.

seed        325377

app_style         diffusion nonlinear hop

dimension         2
lattice           tri 1.0
region            simulationbox block 0 1000 0 250 -0.5 0.5
create_box        simulationbox
create_sites      box

barrier           hop 6

ecoord            0     100000000000000
ecoord            1     5
ecoord            2     4
ecoord            3     3
ecoord            4     2
ecoord            5     1
ecoord            6     0


set               site value 1
set               site value 2 if y < 10
set               site value 3 if y > 429
set               site value 3 if x = 1 if y = 0
set               site value 3 if x = 999 if y = 0
set               site value 3 if x = .5 if y < 1
set               site value 3 if x = 999.5 if y < 1

solve_style       linear

deposition        0.0000000002 0 -1 0 1 2 6
temperature       0.1915

diag_style        diffusion stats yes

stats             10000000000000
dump              mydump 10000000000000 dump.GaAs.1000wide.unitstep

run               300000000000000.0

temperature       .2394 # Temperature Step Change

run               1200000000000000.0
```

B.5 Base Case Material with Initial Roughness Disturbance

```
log GaAs.1000w.rough

#GaAs model from 2009 Senior Design report
#Energy and barrier values are in multiple of 0.27 eV, and temperature is in
#terms of T[K]*(kb[eV*K^-1]/0.27[eV]
#In other words, multiply temperature by 3133 to get Kelvin, and divide time
#by 10^13 to get seconds.

seed              3253777

app_style         diffusion nonlinear hop

dimension         2
lattice           tri 1.0
region            simulationbox block 0 1000 0 120 -0.5 0.5
create_box        simulationbox
create_sites      box

barrier           hop 6

ecoord            0     100000000000000
ecoord            1     5
ecoord            2     4
ecoord            3     3
ecoord            4     2
ecoord            5     1
ecoord            6     0


set               site value 1
set               site value 2 if y < 1.75
set               site value 2 if y < 3.5       if x >= 0 if x < 25.5
set               site value 2 if y < 5.25      if x >= 25 if x < 50.5
set               site value 2 if y < 7         if x >= 50 if x < 75.5
set               site value 2 if y < 5.25      if x >= 75 if x < 100.5
set               site value 2 if y < 3.5       if x >= 100 if x < 125.5
set               site value 2 if y < 5.25      if x >= 125 if x < 150.5
set               site value 2 if y < 7         if x >= 150 if x < 175.5
set               site value 2 if y < 5.25      if x >= 175 if x < 200.5
set               site value 2 if y < 5.25      if x >= 225 if x < 250.5
set               site value 2 if y < 7         if x >= 250 if x < 275.5
set               site value 2 if y < 5.25      if x >= 275.5 if x < 300.5
set               site value 2 if y < 3.5       if x >= 300 if x < 325.5
set               site value 2 if y < 5.25      if x >= 350 if x < 375.5
set               site value 2 if y < 7         if x >= 375 if x < 400.5
set               site value 2 if y < 5.25      if x >= 400 if x < 425.5
set               site value 2 if y < 3.5       if x >= 425 if x < 450.5
set               site value 2 if y < 5.25      if x >= 450 if x < 475.5
set               site value 2 if y < 7         if x >= 475 if x < 500.5
set               site value 2 if y < 5.25      if x >= 500 if x < 525.5
set               site value 2 if y < 3.5       if x >= 525 if x < 550.5
set               site value 2 if y < 5.25      if x >= 550 if x < 575.5
set               site value 2 if y < 7         if x >= 575 if x < 600.5
set               site value 2 if y < 5.25      if x >= 600 if x < 625.5
set               site value 2 if y < 3.5       if x >= 625 if x < 650.5
set               site value 2 if y < 5.25      if x >= 650 if x < 675.5
set               site value 2 if y < 7         if x >= 675 if x < 700.5
set               site value 2 if y < 5.25      if x >= 700 if x < 725.5
set               site value 2 if y < 3.5       if x >= 725 if x < 750.5
set               site value 2 if y < 5.25      if x >= 750 if x < 775.5
set               site value 2 if y < 7         if x >= 775 if x < 800.5
set               site value 2 if y < 5.25      if x >= 800 if x < 825.5
set               site value 2 if y < 3.5       if x >= 825 if x < 850.5
set               site value 2 if y < 5.25      if x >= 850 if x < 875.5
set               site value 2 if y < 7         if x >= 875 if x < 900.5
set               site value 2 if y < 5.25      if x >= 900 if x < 925.5
set               site value 2 if y < 3.5       if x >= 925 if x < 950.5
set               site value 2 if y < 5.25      if x >= 950 if x < 975.5
set               site value 2 if y < 7         if x >= 975 if x < 1000.5

set               site value 3 if y > 170
set               site value 3 if x = 1 if y = 0
set               site value 3 if x = 999 if y = 0
set               site value 3 if x = .5 if y < 1
set               site value 3 if x = 999.5 if y < 1

solve_style       linear

deposition        0.0000000002 0 -1 0 1 2 6
temperature       0.3147

diag_style        diffusion stats yes
stats             100000000000000000

#dump             mydump 1e13 dump.SS.GaAs1000w_rough

#run              1.6e14
```

B.6 Material 2: Three-dimensional Simulation, GaAs Binding Energies , Simple Cubic 6 neighbors

```
# test10 sq/6n lattice GaAs binding model Schwoebel hops
# depositing 1 monolayer/second --> scale time by *10e13
#     (divide all rates by 10^13 to get spparks rates, therefore,
#      multiply time by 10^13 to get spparks time)

# temperature ranges from 0.20044(626.6K) to 0.32572 (1020.5K)
# gamma ranges from 1 to 10e5
# growing 20 monolayers
# gamma = 1000

log                  log.spparks.sc6.gaas.y.1e3.txt

seed                 987654

app_style            diffusion nonlinear schwoebel 6 1

dimension            3
lattice              sc/6n 1.0
region               simulationbox block 0 80 0 80 0 50
create_box           simulationbox
create_sites         box

barrier              hop         6
barrier              schwoebel   7

ecoord               0     1000000000000000
ecoord               1     5
ecoord               2     4
ecoord               3     3
ecoord               4     2
ecoord               5     1
ecoord               6     0


set                  site value 1
set                  site value 2 if z < 4
set                  site value 3 if z < 2
set                  site value 3 if z > 48.91
set                  site value 2 if x = 0 if y = 0 if z = 0


solve_style          linear

deposition           6.4e-10 0 0 -1 0.6 1 6
temperature          0.26058

diag_style           diffusion stats yes

stats                2e13
dump                 mydump 2e13 dump.sc6.gaas.y.1e3

run                  2e14
```

Appendix C Computer Code (C/C++)

C.1 Open Loop Optimizer Main.cpp

```
/*
* Creates an optimal control profile for a given input script. Input script
* must not include the following commands:
* log, dump, run, stats. Script must include deposition and temperature
* commands. Deposition command will be used the same throughout,
* changing the deposition rate to take into account desorption. Temperature *
* command will be used as an initial guess. Region command
* must in the form "region block xlo xhi ylo yhi zlo zhi". Currently cannot
* have particles floating in the air. User must input
* constants such as number of sub runs and runtime in beginning of main.
* Should check that the actual runtime of the process
* is close to the runtime you put in. If it isn't, an individual timestep
* might be running for much longer than how long you wanted
* it to run because events are happening slowly. Prints both the dump file of
* the main run (to dump.main)
* and the profile of that dump file to a file the user specifies.
*/

/* Open-Loop Optimizer
 * Date: 4-13-11
 * Programed by:  Efrem Braun
 *                Marija Mircevska
 *                Manuel Molina Villalba
 */

#include <iostream>
#include <fstream>
#include "library.h"
#include <stdio.h>
#include <math.h>
#include "CreateDataFile.h"
#include "ObjectiveFunctionCalculator.h"
#include "DumpToProfile.h"
#include "smallSitesData.h"
#include "smallKMCInput.h"

#define MAXLENGTHFILE 100
#define MAXLENGTHBUFFER 100

using namespace std;

int main(){
    /*User Inputs*/
    char inputscript[MAXLENGTHFILE] = "GaAs.1000wide"; //Name of input file.
      File must be in same directory as the main
    const int SUBRUNS = 5; //Number of sub runs of SPPARKS that will be run
      for each timestep to find optimal temperature. Must be an odd integer
    const double RUNTIME = 150000000000000.0; //Approximate length of total
      run in seconds.
    const double TIMESTEP = RUNTIME / 15.0; //Timestep at which it will run
      each subrun.
    const double ADVANCETIME = RUNTIME / 30.0; //Time it will advance the
```

```
  main SPPARKS run by after the control loop
const double dumpsize = RUNTIME / 30.0;
const int segmentsize = 10;     //Segment size for roughness calculator
const int SHRINKSIZE = 10;      //Number by which the main lattice will
                                  be shrunk for subruns.
const int SHRINKRUNS = 4;       //Number of shrunk runs that will be
                                  averaged for each subrun
//The maximum amount the temperature can change per timestep. Currently
20K
const double MAXTEMPCHANGE = 0.006384;

//Define variables.
double DBLSHRINKSIZE = (double) SHRINKSIZE;
char outputprofile [MAXLENGTHBUFFER], buffer[MAXLENGTHBUFFER],
  tempbuffer[MAXLENGTHBUFFER], depbuffer[MAXLENGTHBUFFER],
  runbuffer[MAXLENGTHBUFFER], timebuffer[MAXLENGTHBUFFER];
char blank[1] = "";
char *cptr = blank;
double temperature, temparray[SUBRUNS], objectivefunctionarray[SUBRUNS],
  miniobjectivefunctionarray[SHRINKRUNS], lowestobjectivefunction,
  deprate, netdeprate, netdeprateadj, dirx, diry, dirz, dO, currenttime,
  xlo, xhi, ylo, yhi;
int lo, hi, dimension, ctr1, ctr;
FILE *fpinputscript, *fpoutputtemp;

//Name of output profile
sprintf(outputprofile, "%s%s%s", "profile.", inputscript, ".txt");
ofstream newStream(outputprofile, fstream::out);
if(!newStream.is_open()){
    cerr << "Failed to open out put file ..."<<endl;
    exit(1);
}
newStream << "TIME ROUGHNESS GROWTHRATE POROSITY OBJECTIVE_FUNTION"
  <<endl;

/* Opens input file and gathers initial temperature, gross deposition
  rate(not taking desorption based on temperature into account),
  deposition vector, deposition capture distance, min and max
  coordination numbers for deposition, and box size. */
{
    //Open for reading the input script file
    if ((fpinputscript = fopen(inputscript, "r")) != NULL){   ;}
    else{
        fprintf(stderr, "\nError opening input script");
        fcloseall();
        exit(1);
    }
    //Gathers constants
    while (!feof(fpinputscript)){
        fgets(buffer, MAXLENGTHBUFFER, fpinputscript);{
            if (strncasecmp(buffer, "temperature", 11) == 0){
                sscanf(buffer, "%*s %lf", &temperature);
            }
            if (strncasecmp(buffer, "deposition", 10) == 0){
                sscanf(buffer, "%*s %lf %lf %lf %lf %lf %d %d", &deprate,
                    &dirx, &diry, &dirz, &dO, &lo, &hi);
            }
            if (strncasecmp(buffer, "region", 6) == 0){
                sscanf(buffer, "%*s %*s %*s %lf %lf %lf %lf", &xlo, &xhi,
                    &ylo, &yhi);
            }
        }
    }
}

//Open for writing the output temp profile
char tempprofilebuffer[MAXLENGTHBUFFER];
sprintf(tempprofilebuffer, "tempprofile.%s.txt", inputscript);
fstream temperatureProfile(tempprofilebuffer, fstream::out);
if (!temperatureProfile.is_open()){
    fprintf(stderr, "\nError opening output temp profile file");
    exit(1);
}

//Header of temp profile
//fprintf(fpoutputtemp, "timestep(approx) temperature\n");
temperatureProfile << "timestep(approx) temperature" <<endl;

//Open the main instance of SPPARKS
int a = 1;
int *ptra = &a;
int **ptrmain = &ptra;
spparks_open(1, &cptr, 0, (void **) ptrmain);

//Open a sub instance of SPPARKS
int b = 1;
int *ptrb = &b;
int **ptrsub = &ptrb;
spparks_open(1, &cptr, 0, (void **) ptrsub);

//Run the main instance of SPPARKS for 0 time. Collect the dimension.
  Makes a data file.
char logbuffer[MAXLENGTHBUFFER];
sprintf(logbuffer, "log log.%s", inputscript);
spparks_command(*ptrmain, logbuffer);
spparks_file(*ptrmain, inputscript);
char dumpbuffer[MAXLENGTHBUFFER];
sprintf(dumpbuffer, "dump mydump %lf dump.%s", dumpsize, inputscript);
spparks_command(*ptrmain, dumpbuffer);
//spparks_command(*ptrmain, "read_sites sites.data");
dimension = *((int*)spparks_extract(*ptrmain, "dimension"));
CreateDataFile(ptrmain, "datamain");

//Run the sub instance of SPPARKS for 0 time to input settings.
char minibuffer[MAXLENGTHBUFFER];
sprintf(minibuffer, "mini.%s", inputscript);
smallKMCInput(inputscript, minibuffer, SHRINKSIZE);
spparks_file(*ptrsub, minibuffer);
smallSitesData((void **)ptrmain, "datasub", SHRINKSIZE);

//Test (part 1 of 2)
FILE *fptest;
char testbuffer[MAXLENGTHBUFFER];
sprintf(testbuffer, "test.%s.txt", inputscript);
```

```
if ((fptest = fopen(testbuffer, "w")) != NULL){
        ;
    }
else{
    fprintf(stderr, "\nError opening test output file");
    fcloseall();
    exit(1);
}
fprintf(fptest, "timestep temperature roughness growthrate porosity
  objectivefunction\n");

//Control loop
for (ctr1 = 0; ctr1 < RUNTIME / ADVANCETIME; ctr1++){
    //Create a temperature array for the temperatures that the sub
    instances of SPPARKS will be running.
    for (ctr = 0; ctr < SUBRUNS; ctr++){
        //Temp can only change +-MAXTEMPCHANGE at most
        temparray[ctr] = temperature - MAXTEMPCHANGE + (2 * MAXTEMPCHANGE
              * ((double )ctr) / ((double) SUBRUNS - 1));
    }

    //Run the sub instance of SPPARKS SUBRUNS times for TIMESTEP seconds
    with temperature and deposition commands as given.
    for (ctr = 0; ctr < SUBRUNS; ctr++){
        temperature = temparray[ctr];
        if (temperature < 0) temperature = 0;

        if (dimension == 2)
        {
            netdeprate = deprate / DBLSHRINKSIZE;
            netdeprateadj = netdeprate * DBLSHRINKSIZE;
        }
        else if (dimension == 3)
        {
            netdeprate = deprate / (DBLSHRINKSIZE * DBLSHRINKSIZE);
            netdeprateadj = netdeprate * DBLSHRINKSIZE * DBLSHRINKSIZE;
        }
        else
        {
            fprintf(stderr, "\nError: dimension is not 2 or 3\n");
            exit(1);
        }
        //Run the subrun SHRINKRUNS times, and then take average
        objective function.
        for (int ctr2 = 0; ctr2 < SHRINKRUNS; ctr2++)
        {
            sprintf(depbuffer, "deposition %.30lf %lf %lf %lf %lf %d %d",
              netdeprate, dirx, diry, dirz, d0, lo, hi);
            sprintf(tempbuffer, "temperature %lf", temperature);
            sprintf(runbuffer, "run %lf", TIMESTEP);
            currenttime = ADVANCETIME * ((double) ctr1);
            sprintf(timebuffer, "reset_time %lf", currenttime);

            //Note: this reset is only approximate, and should always lag
            the main run's actual time.
            spparks_command(*ptrsub, timebuffer);
            spparks_command(*ptrsub, "read_sites datasub");
        spparks_command(*ptrsub, tempbuffer);
        spparks_command(*ptrsub, depbuffer);
        spparks_command(*ptrsub, runbuffer);

        miniobjectivefunctionarray[ctr2] =
          ObjectiveFunctionCalculator(ptrsub, netdeprateadj, deprate,
          segmentsize);

        //Test (part 2 of 2)
          fprintf(fptest, "%lf %lf %lf %lf %lf %lf\n", currenttime,
                temperature, RoughnessCalculator(ptrsub,
                segmentsize), netdeprateadj / deprate,
                PorosityCalculator(ptrsub),
                miniobjectivefunctionarray[ctr2]);
    }
    double subobjectivefunction = 0;
    for (int ctr2 = 0; ctr2 < SHRINKRUNS; ctr2++){
        subobjectivefunction += miniobjectivefunctionarray[ctr2];
    }

    objectivefunctionarray[ctr] = subobjectivefunction / SHRINKRUNS;
}

//Find the temperature which minimizes the objective function
temperature = temparray[0];
lowestobjectivefunction = objectivefunctionarray[0];
for (ctr = 1; ctr < SUBRUNS; ctr++){
    if (objectivefunctionarray[ctr] < lowestobjectivefunction){
        lowestobjectivefunction = objectivefunctionarray[ctr];
        temperature = temparray[ctr];
    }
}
//Output temp to temp profile
temperatureProfile <<currenttime<<" "<<temperature<<endl;

//Run the main instance of SPPARKS for TIMESTEP seconds with the best
temperature as given from the sub runs
sprintf(depbuffer, "deposition %.30lf %lf %lf %lf %lf %d %d",
deprate, dirx, diry, dirz, dO, lo, hi);
sprintf(tempbuffer, "temperature %lf", temperature);
sprintf(runbuffer, "run %lf", ADVANCETIME);
spparks_command(*ptrmain, "read_sites datamain");
spparks_command(*ptrmain, tempbuffer);
spparks_command(*ptrmain, depbuffer);
spparks_command(*ptrmain, runbuffer);

//Log Current Properties for post processing
newStream.precision(20);
newStream.width(20);
newStream << currenttime<<" ";

newStream.precision(20);
newStream.width(20);
newStream << RoughnessCalculator(ptrmain, segmentsize)<<" ";

newStream.precision(20);
newStream.width(20);
```

```
        newStream << netdeprateadj / deprate<<" ";

        newStream.precision(20);
        newStream.width(20);
        newStream << PorosityCalculator(ptrmain)<<" ";

        newStream.precision(20);
        newStream.width(20);
        newStream << ObjectiveFunctionCalculator(ptrmain, netdeprateadj,
            deprate, segmentsize)<<" "<<endl;

        //Create two new data files
        CreateDataFile(ptrmain, "datamain");
        smallSitesData((void **)ptrmain, "datasub", SHRINKSIZE);
    }

    //Close all instances of SPPARKS
    spparks_close(*ptrmain);
    spparks_close(*ptrsub);

    //Close open files
    fcloseall();

    //Print profile
    puts("Creating profile from dump file. Please wait.");
    char dumpbuffer2[MAXLENGTHBUFFER];
    sprintf(dumpbuffer2, "dump.%s", inputscript);
    DumpToProfile(dumpbuffer2, dimension, outputprofile, segmentsize,
            deprate);

    return 0;
}
```

C.2 Model Predictive Control Main.cpp

```
/*
 * CBE 459: Senior Design
 *
 * MODEL PREDICTIVE CONTROL V 1.0.0
 * Date: March 31, 2011
 * Authors:   Efrem Braum
 *            Marija Mircevska
 *            Manuel Molina Villalba
 */

#include <cstring>
#include <ctime>
#include <sys/time.h>
#include <cstdio>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <limits>
#include <map>
#include <string>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "growth.h"
#include "library.h"
#include "mpi.h"
#include "setPoints.h"
#include "simulationPoints.h"
#include "smallKMCInput.h"
#include "smallSitesData.h"
#include "objectiveFunction.h"
#include "porosity.h"
#include "roughness.h"
#define MAXBUFFER 100

using namespace std;

int main(){
    srand(time(0));                                 /* Random number generator seed */
    int seed = rand();                             /* User defined: To be used to
create new simulations */

    /**---------------------- MPC Variable Definitions BEGIN ----------------------**/
    /**User Set Variables to modify**/
    const double error = 0.001;   /* User defined: controller error tolerance for
                                        comparator, to skip optimization*/
    const double processTime = 200000000000000000;    /* User defined: Time real
                                 process is to be controlled for */
    const double simulationStep = 1000000000000000.0 * 0.01; /* User defined: Step
                                 command for spparks real simulation */
    const double predictiveStep = 1000000000000000.0* 0.02; /* User defined: Step
                                 command for spparks optimizer */
    const int denominator = 10;  /* User defined: Denominator of fraction of the
                                        lattice size */
    const int average = 8;        /* User defined: Number of different simulations to
                                        run and average at each small KMC run */
    const int SUBRUNS = 7;        /* User defined: No. of runs that are to be run by
                                        the optimizer */
    int segmentSize = 10;         /* User defined: The size of the segment to be used
```

```
                                    for roughness calculations */
const double tempRange = 0.00638365784; /* Temperature Range of the optimizer
                                    below and above the current temperature*/

/**Simulation Variables do not modify !!!**/
int **real;
int **model;
double currentTime = 0;       /* Time of real process */
vector<double> temporarySPProperties;      /* Vector to hold set point roughness,
                                           growth rate, porosity for optimizer*/
vector<double> temporaryProperties;        /* Vector to hold roughness, growth
                                           rate, porosity for model simulations*/
vector<double> currentProperties;          /* Vector to hold roughness, growth
                                           rate, porosity for current simulation*/
vector<double> setPointProperties;         /* Vector to hold set point roughness,
                                           growth rate, porosity*/
double controlError = 0;                   /* Hold the current control error*/
double currentTimer = 0;                   /* Times each control action*/
double pastTimer = 0;                      /* Holds the time for control action
                                           for logging*/

/* After the first real process step, this deposition rate, netDepositionsRate,
   will  take over as SPPARKS deposition command input for the real process
   simulation. This will fluctuate around the variable depositionRate. */
double netDepositionRate;

/* Files to manage: SPPARKS input file, MPC data file, Reference Trajectory */
char inputFilename[MAXBUFFER] =  "GaAs.1000wide.unitstep";
char referenceTrajectory[MAXBUFFER] = "profile.GaAs.1000wide.unitstep";
ofstream logMPC;
ofstream optimizerLog;

/* Real Simulation variables  */
double temperature;
int dimension;

/* depositionRate WILL NOT CHANGE THROUGHOUT THE MPC CONTROL. This is the initial
   deposition rate that is to correspond to the constant deposition at constant
   gas flow. Will be used in the netDepositionRate calculations.*/
double depositionRate;
double depX, depY, depZ, depD0;            /* Deposition incident angle variables
                                           XYZ and d0 */
int depLo, depHi;                          /* Deposition variables low and high
                                           coordination limit */
double xLo, xHi,yLo,yHi;                   /* Region length and with variables */

char tempBuffer[MAXBUFFER];
char depBuffer[MAXBUFFER];
char resetBuffer[MAXBUFFER];
/**----------------------- MPC Variable Definitions END ------------------------**/

/**-------------------------------------------------------------------------------
 ----------------------- Model Predictive Controller BEGIN -----------------------
 ------------------------------------------------------------------------------**/

/**--------- Extract Temperature and Deposition from input file BEGIN ---------**/

/** Open SPPARKS input file **/
fstream inputFile(inputFilename);
if(!inputFile){
         cerr << "Error opening SPPARKS Input file" << endl;
                exit(1);
}
```

```
    /** Extract Temperature and Deposition variables from the SPPARKS input file **/
       string line;
       while(inputFile >> line){
              int bad_input;
              if((line == "temperature") | (line == "deposition") | (line == "region")
                     | (line == "seed")){
                     do{
                            bad_input = 0;
                            char buffer[MAXBUFFER];
                            if(line == "seed")
                                   inputFile >> seed;
                            else if(line == "temperature")
                                   inputFile >> temperature;
                            else if(line == "deposition"){
                                   inputFile.getline(buffer,
                                          numeric_limits<streamsize>::max(),'\n');
                                   sscanf(buffer, "%lf %lf %lf %lf %lf %d %d",
                                   &netDepositionRate, &depX, &depY, &depZ, &depDO,
                                          &depLo, &depHi);
                                   depositionRate = netDepositionRate;
                            }
                            else if(line == "region"){
                                   inputFile.getline(buffer,
                                          numeric_limits<streamsize>::max(),'\n');
                                   sscanf(buffer, "%*s %*s %lf %lf %lf %lf", &xLo, &xHi,
                                          &yLo, &yHi);
                            }
                            if(!inputFile){
                                   bad_input = 1;
                                   inputFile.clear();
                                   inputFile.ignore(numeric_limits<streamsize>::max(),'
                                          ');
                            }
                     }while(bad_input);
              }
    }
    inputFile.close();
    cout <<"netDepositionRate, depX, depY, depZ, depDO, depLo, depHi ";
    cout <<netDepositionRate<<" "<<depX<<" "<<depY<<" "<<depZ<<" "<<depDO<<"
              "<<depLo<<" "<<depHi<<endl<<endl;

/************* Extract Temperature and Deposition from input file END **************/

    /** Create a MPC log file **/
    char MPCdata[MAXBUFFER];
    strcpy(MPCdata, inputFilename);
    logMPC.open(strcat(MPCdata,"_MPC.log"), fstream::trunc);
    if(!logMPC.is_open()){
        cerr << "Error opening MPC log file" << endl;
        exit(1);
    }
    /* LOG HEADERS */
    logMPC << "TIME          TEMPERATURE   NET_DEPOSITION_RATE  ROUGHNESS ";
    logMPC << "GROWTHRATE    POROSITY      OBJECTIVE_FUNCTION   spROUGHNESS ";
    logMPC << "spGROWTHRATE  spPOROSITY    spOBJECTIVE_FUNCTION ERROR
              CONTROL_TIME"<<endl;

    /** Create a optimizer log file **/
    char optimizerName[MAXBUFFER];
    strcpy(optimizerName, inputFilename);
    optimizerLog.open(strcat(optimizerName,"_optimizer.log"), fstream::trunc);
    if(!optimizerLog.is_open()){
```

```
    cerr << "Error opening Optimizer Log" << endl;
    exit(1);
}
/* LOG HEADERS */
optimizerLog << "TEMPERATURE   NET_DEPOSITION_RATE ROUGHNESS ";
optimizerLog << "GROWTH_RATE   POROSITY            OBJECTIVE_FUNCTION
   spROUGHNESS ";
optimizerLog << "spGROWTHRATE  spPOROSITY          spOBJECTIVE_FUNCTION
   OPTIMIZER_OF"<<endl;

/** Begin SPPARKS objects **/
char blank[1] = "";
char *cptr = blank;
int *ptr = 0;
real = &ptr;
int *point = 0;
model = &point;
/*SPPARKS object that is to be used for the real process model*/
spparks_open(1, &cptr, 0, (void**)real);                /*SPPARKS Command*/

/*SPPARKS object that is to be used for the small KMC model*/
spparks_open(1, &cptr, 0, (void**)model);               /*SPPARKS Command*/

/** Begin real SPPARKS simulation **/
spparks_file(*real, inputFilename);                     /*SPPARKS Command*/
//spparks_command(*real, "read_sites porous.sites");    /*SPPARKS Command*/
spparks_command(*real, "dump mydump 100000000000000.0 dump.GaAs.1000wide.rough");
dimension = *((int*)spparks_extract(*real, "dimension")); /*Get Dimension of
   simulation*/

/** Begin model SPPARKS simulation**/
smallKMCInput(inputFilename,"smallKMCInput",denominator);   /*Create an input file
   for the small KMC model*/
spparks_file(*model, "smallKMCInput");                      /*SPPARKS Command*/
spparks_command(*model, "log GaAs.1000wide_smallKMCSPPARKS.log");
                                                            /*SPPARKS Command*/
/** Obtain the set point and current properties at this time **/
setPointProperties = setPoints(currentTime, referenceTrajectory);
currentProperties = simulationPoints((void**) real, netDepositionRate,
   depositionRate, segmentSize);

/* Check if the reference file is empty */
/* If the setPointProperties vector contains all -1 the file was parsed and the
   time was not found meaning that the currentTime is greater than the last time
   in the reference trajectory file. The control loop should stop here if that is
   the case. */
int stop = 0;
for(vector<double>::iterator i = setPointProperties.begin();i <
         setPointProperties.end(); ++i){
    if(*i != -1){
        ++stop;
    }
}
if(!stop){
    cout << "Reference Trajectory has no values. Now exiting..."<<endl;
    return 0;
}

/***************************** Begin Control Loop *******************************/
while(currentTime <= processTime){
    /** Log values for the current time to the MPC log **/
    logMPC.width(13);   logMPC <<currentTime<<" ";
    logMPC.width(11);   logMPC <<temperature<<" ";
logMPC.width(19);   logMPC <<netDepositionRate<<" ";
logMPC.width(13);   logMPC <<currentProperties[0]<<" ";
logMPC.width(10);   logMPC <<currentProperties[1]<<" ";
logMPC.width(13);   logMPC <<currentProperties[2]<<" ";
logMPC.width(18);   logMPC <<objectiveFunction(currentProperties)<<" ";
logMPC.width(13);   logMPC <<setPointProperties[0]<<" ";
logMPC.width(13);   logMPC <<setPointProperties[1]<<" ";
logMPC.width(13);   logMPC <<setPointProperties[2]<<" ";
logMPC.width(20);   logMPC <<objectiveFunction(setPointProperties)<<" ";
logMPC.width(13);   logMPC <<controlError<<" ";
logMPC.width(13);   logMPC <<pastTimer<<endl;

cout << endl << "Current time of the real process is: " << currentTime <<endl
     << endl;

/**Current Timer Start**/
timeval t1;
gettimeofday(&t1, NULL);

/** Run real process to step K **/
char run[MAXBUFFER];
char seedCommand[MAXBUFFER];
sprintf(run, "run %lf", simulationStep);

cout <<endl<<run<<endl<<endl;

spparks_command(*real, run);                  /*Run Real Process to step K*/

/** Increment current time of the MPC **/
currentTime += simulationStep;

/** Obtain current simulation properties and set point properties at the
current time**/
/*These are to be logged at the next loop iteration*/
currentProperties = simulationPoints((void**) real, netDepositionRate,
     depositionRate,segmentSize);
setPointProperties = setPoints(currentTime, referenceTrajectory);

/** Run model of process to step K + 1 **/
sprintf(run, "run %e", predictiveStep);
sprintf(tempBuffer,"temperature %lf", temperature);
switch(dimension){
    case 2:{
        sprintf(depBuffer,"deposition %.30lf %lf %lf %lf %lf %d
            %d",(depositionRate/denominator),depX,depY,depZ,
            depDO, depLo, depHi);
        break;
    }
    case 3:{
        sprintf(depBuffer,"deposition %.30lf %lf %lf %lf %lf %d
            %d",(depositionRate/(denominator*denominator)),
            depX,depY,depZ, depDO, depLo, depHi);
        break;
    }
}

/** Create sites data file for SPPARKS read_sites command **/
smallSitesData((void**)real,"smallKMC.data",denominator);/*Create sites data
     file for SPPARKS read_sites command*/
sprintf(resetBuffer, "reset_time %e", currentTime);

 /** Obtain the set point at time K + 1**/
temporarySPProperties = setPoints((currentTime + predictiveStep),
         referenceTrajectory);

    /** Check if the reference file is empty**/
    int stop = 0;
    for(vector<double>::iterator i = temporarySPProperties.begin();
                i < temporarySPProperties.end(); ++i){
        if(*i != -1)
            ++stop;
    }
    if(!stop){
        cout << "All reference points used. Control run is finished. Now
                exiting..."<<endl;
        logMPC.close(); optimizerLog.close(); spparks_close(*real);
                spparks_close(*model);
        return 0;
    }

    /** Average multiple model SPPARKS runs to reduce stochasticity **/
    /* The amount of simulations to average can be specified with the variable
   average*/
    vector< vector<double> > toAverage;
    for(int i = 0; i < average; ++i){
        sprintf(seedCommand, " seed %d", i + seed);
        spparks_command(*model,resetBuffer);                  /*SPPARKS Command*/
        spparks_command(*model,"read_sites smallKMC.data");   /*SPPARKS Command*/
        spparks_command(*model, seedCommand);                 /*SPPARKS Command*/
        spparks_command(*model, run);                         /*SPPARKS Command*/
        toAverage.push_back(simulationPoints((void**) model, netDepositionRate,
                depositionRate,segmentSize));
        //cout <<endl<< "Running simulation "<<average<<endl<<endl;
    }

    /** Obtain the average current properties at time K + 1**/
    vector<double> temporary;
    for(int j = 0; j < average; ++j){
        double sum = 0;
        for(vector<vector<double> >::size_type k = 0; k < toAverage.size(); ++k){
            sum += toAverage[k][j];
        }
        temporary.push_back(sum/average);;
    }
    temporaryProperties = temporary;

    /** Objective function comparator **/
    double currProp = objectiveFunction(temporaryProperties);
    double spProp = objectiveFunction(temporarySPProperties);
    controlError =   (currProp - spProp) / spProp;

    /**Control action timer stop**/
    timeval t2;
    gettimeofday(&t2,NULL);
    currentTimer = currentTimer + (t2.tv_sec - t1.tv_sec);

    cout <<endl<< "currProp and spProp"<<currProp<< ", " <<spProp<<endl;
    cout <<endl << "Error is " << controlError <<endl;

    /**-------------------------- Optimizer BEGIN ----------------------------**/
    if(fabs(controlError) > error){
        cout <<endl << "*************BEGIN OPTIMIZER*************"<<endl<<endl;

        cout <<"Current time of real process is "<<currentTime<<endl<<endl;

        /** Obtain the set point at time K + 1**/
```

```
temporarySPProperties = setPoints(currentTime + predictiveStep,
        referenceTrajectory);
/** Check if the reference file is empty**/
int stop = 0;
for(vector<double>::iterator i = temporarySPProperties.begin();
        i < temporarySPProperties.end(); ++i){
    if(*i != -1)
        ++stop;
}
if(!stop){
    cout << "All reference points used. Control run is finished. Now
        exiting..."<<endl;
    logMPC.close(); optimizerLog.close(); spparks_close(*real);
        spparks_close(*model);
    return 0;
}

/**Timer start add to current timer**/
gettimeofday(&t1,NULL);

 /** Create arrays to hold the values for each test simulation **/
 double tempArray[SUBRUNS];
 double objectiveArray[SUBRUNS];
 for(int i = 0; i < SUBRUNS; ++i){
     //Temp can only change by tempRange / 2 at most
     tempArray[i] = temperature - tempRange + (2 * tempRange * ((double
        )i) / ((double) SUBRUNS - 1));
    if(tempArray[i] < 0) tempArray[i] = 0;
 }

 /** Calculate objective function for each test simulation j **/
 double netDepRateBig;
 for(int j = 0; j < SUBRUNS; ++j){
        double netDepRate;
        switch(dimension){
        case 2:
               netDepRate = depositionRate/ denominator;
               netDepRateBig = depositionRate * denominator;
               break;
        case 3:
               netDepRate = depositionRate / denominator *  denominator;
               netDepRateBig = depositionRate * denominator * denominator;
               break;
        default:
               cout << endl << "Error: dimesion is not 2 or 3." << endl;
               exit(1);
               break;
        }

        /** Begin model SPPARKS simulation at step K + 1 **/
        sprintf(depBuffer,"deposition %.30lf %lf %lf %lf %lf %d
               %d",netDepRate,depX,depY,depZ, depDO, depLo, depHi);
        sprintf(tempBuffer,"temperature %lf", tempArray[j]);
        sprintf(resetBuffer, "reset_time %e", currentTime);
        sprintf(run, "run %e", predictiveStep);

        /* Average multiple model SPPARKS runs to reduce stochasticity */
        toAverage.clear();
        for(int i = 0; i < average; ++i){
               /** Begin model SPPARKS simulation at step K**/
               /*SPPARKS Commands to model simulation*/
               sprintf(seedCommand, " seed %d", i + seed);
               spparks_command(*model,"read_sites smallKMC.data");
```

```
              /*SPPARKS Command*/
              spparks_command(*model, seedCommand);
              /*SPPARKS Command*/
              spparks_command(*model, resetBuffer);
              /*SPPARKS Command*/
              spparks_command(*model, depBuffer);
              /*SPPARKS Command*/
              spparks_command(*model, tempBuffer);
              /*SPPARKS Command*/
              spparks_command(*model, run);
              /*SPPARKS Command*/
              toAverage.push_back(simulationPoints((void**) model,
              netDepRateBig, depositionRate,segmentSize));
   }
   temporary.clear();
   for(int m = 0; m < average; ++m){
       double sum = 0;
       for(vector<vector<double> >::size_type k = 0; k <
              toAverage.size(); ++k){
           sum += toAverage[k][m];
       }
       temporary.push_back(sum/average);
   }

    /** Obtain the average current properties at time K + 1 **/
   temporaryProperties = temporary;

   /** Objective function to be minimized by the optimizer **/
       objectiveArray[j] = pow(objectiveFunction(temporarySPProperties) -
              objectiveFunction(temporaryProperties),2);

   /** End and add to current control action timer**/
   gettimeofday(&t2,NULL);
   currentTimer = currentTimer + (t2.tv_sec - t1.tv_sec);

   /** Log all the test simulation, j, values to the Optimizer Log **/
   optimizerLog.width(15); optimizerLog <<tempArray[j]<<" ";
   optimizerLog.width(19); optimizerLog <<netDepRate<<" ";
   optimizerLog.width(15); optimizerLog <<temporaryProperties[0]<<" ";
   optimizerLog.width(15); optimizerLog <<temporaryProperties[1]<<" ";
   optimizerLog.width(15); optimizerLog <<temporaryProperties[2]<<" ";
   optimizerLog.width(18);
  optimizerLog <<objectiveFunction(temporaryProperties)<<" ";
   optimizerLog.width(15); optimizerLog <<temporarySPProperties[0]<<" ";
   optimizerLog.width(15); optimizerLog <<temporarySPProperties[1]<<" ";
   optimizerLog.width(15); optimizerLog <<temporarySPProperties[2]<<" ";
   optimizerLog.width(20);
  optimizerLog <<objectiveFunction(temporarySPProperties);
   optimizerLog <<" "<<objectiveArray[j]<<endl;
}
optimizerLog <<endl;

/** Start and add to current timer **/
gettimeofday(&t1,NULL);

/** Search for the temperature that minimizes the objective function **/
double objectiveTemp = tempArray[0];
double minObjective = objectiveArray[0];
for(int i = 1; i < SUBRUNS; ++i){
       if(objectiveArray[i] < minObjective){
              minObjective = objectiveArray[i];
              objectiveTemp = tempArray[i];
       }
```

```
             }

             /**End and add to current control action timer**/
             gettimeofday(&t2,NULL);
              currentTimer = currentTimer + (t2.tv_sec - t1.tv_sec);

             /** Change the temperature in the real process with the one that yields
             the best objective and change the netDepositionRate of the real process
             to the new net deposition rate corresponding to the new temperature. **/
             temperature = objectiveTemp;

             /*** Issue new SPPARKS commands to the real process ***/
             char depBuffer[MAXBUFFER];
             sprintf(depBuffer,"deposition %e %lf %lf %lf %lf %d %d",netDepositionRate
                    ,depX,depY,depZ, depDO, depLo, depHi);
             char tempBuffer[MAXBUFFER];
             sprintf(tempBuffer,"temperature %lf", temperature);
             spparks_command(*real, tempBuffer);             /*SPPARKS Command*/

             cout <<endl<< "New temperature and net deposition rates are:
                    "<<temperature<<" and " <<netDepositionRate<<endl<<endl;
             cout <<endl << "*************END OPTIMIZER*************"<<endl<<endl;

        }
        /**------------------------------ Optimizer END ---------------------------**/

        /** Save the currentTimers time and reset it for the next loop **/
        pastTimer = currentTimer;
        currentTimer = 0;
    }
    /**------------------------------ Control Loop END ----------------------------**/


    /**--------------------------------------------------------------------------------
     ------------------------- Model Predictive Controller END -----------------------
     -----------------------------------------------------------------------------**/

    /** Close All Streams **/
    logMPC.close(); optimizerLog.close(); spparks_close(*real); spparks_close(*model);
    return 0;
}
```

C.3 Property Calculators

C.3.1 Objective Function Calculator

```
/* Function that returns the objective function It accepts a pointer to a
SPPARKS object */

#ifndef OBJECTIVEFUNCTIONCALCULATOR_H_INCLUDED
#define OBJECTIVEFUNCTIONCALCULATOR_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include "library.h"
#include "RoughnessCalculator.h"
#include "PorosityCalculator.h"

using namespace std;

double ObjectiveFunctionCalculator (int ** model, double netdeprate, double
deprate, int segmentsize);

#endif // OBJECTIVEFUNCTIONCALCULATOR_H_INCLUDED



#include "ObjectiveFunctionCalculator.h"

double ObjectiveFunctionCalculator (int ** model, double netdeprate, double
deprate, int segmentsize)
{
    double wtroughness = 100.0;
    double wtgrowthrate = 0.0;
    double wtporosity = 1.0;

    double roughness = RoughnessCalculator(model, segmentsize);
    double growthrate = netdeprate / deprate;
    double porosity = PorosityCalculator(model);

    double objectivefunction = (wtroughness * roughness) - (wtgrowthrate *
growthrate) + (wtporosity * porosity);

    return objectivefunction;
}
```

C.3.2 Porosity Calculator

```
/* Function that returns the roughness. It accepts a pointer to a SPPARKS
object */

#ifndef POROSITYCALCULATOR_H_INCLUDED
#define POROSITYCALCULATOR_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "library.h"
#include <fstream>
#include <iostream>
#include <iomanip>
#include <map>
#include <math.h>

using namespace std;

double PorosityCalculator (int ** model);

#endif // POROSITYCALCULATOR_H_INCLUDED



#include "PorosityCalculator.h"

double PorosityCalculator (int ** model){
    int *dimension = ((int*)spparks_extract(*model, "dimension"));
    int *sites = ((int*) spparks_extract(*model, "nglobal"));
    int *atomicState = ((int*) spparks_extract(*model, "site"));
    double **atomicGrid = ((double**) spparks_extract(*model, "xyz"));

    double porosity;

    if(*dimension == 3){
        map<double, map<double, double> > xlattice;
        double **start = atomicGrid;
        for(int *begin = atomicState, *end = atomicState + *sites;begin !=
                    end; ++begin, ++start){
            map<double, double> *ylattice = &xlattice[**start];
            double zlattice = (*ylattice)[*(*start + 1)];
            if (*begin == 2){
                if (*(*start + 2) > zlattice)
                    (*ylattice)[*(*start + 1)] = *(*start + 2);
            }
        }
        double sumheight = 0;
        for (map <double, map <double, double> >::iterator i =
                    xlattice.begin(); i != xlattice.end(); ++i){
            map <double, double> temp = (*i).second;
            for (map <double, double>::iterator j = temp.begin(); j !=
                    temp.end(); ++j){
```

```
                sumheight += (*j).second;
            }
        }

        double unfilled = 0;
        double total = 0;
        start = atomicGrid;
        for(int *begin = atomicState, *end = atomicState + *sites;begin !=
                  end; ++begin, ++start){
            map<double, double> *ylattice = &xlattice[**start];
            double zlattice = (*ylattice)[*(*start + 1)];
            if ((*begin == 1) && (*(*start + 2) <= zlattice)){
                    unfilled++;
            }
            if (*(*start + 2) <= zlattice){
                total++;
            }
        }
        porosity = unfilled / total;
    }
    else if(*dimension == 2){
        map<double, double> lattice;
        double **start = atomicGrid;
        for(int *begin = atomicState, *end = atomicState + *sites ;begin !=
                  end; ++begin, ++start){
            double y = lattice[**start];
            if (*begin == 2){
                if (*(*start + 1) > y)
                    lattice[**start] = *(*start + 1);
            }
        }
        double unfilled = 0;
        double total = 0;
        start = atomicGrid;
        for (int *begin = atomicState, *end = atomicState + *sites; begin !=
                  end; ++begin, ++start){
            if ((*begin == 1) && (*(*start + 1) <= lattice[**start])){
                unfilled++;
            }
            if (*(*start + 1) <= lattice[**start]){
                total++;
            }
        }
        porosity = unfilled / total;
    }
    return porosity;
}
```

C.3.3 Roughness Calculator

```
/* Function that returns the roughness. It accepts a pointer to a SPPARKS
object */

#ifndef ROUGHNESSCALCULATOR_H_INCLUDED
#define ROUGHNESSCALCULATOR_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "library.h"
#include <fstream>
#include <iostream>
#include <iomanip>
#include <map>
#include <math.h>

using namespace std;

double RoughnessCalculator (int ** model, int segmentsize);

#endif // ROUGHNESSCALCULATOR_H_INCLUDED



#include "RoughnessCalculator.h"

double RoughnessCalculator (int ** model, int segmentsize){
    int ctr = 0, ctr2 = 0, ctr3 = 0, ctr4 = 0, ctr5 = 0;
    double sumsegmentheight, averagesegmentheight;
    int *dimension = ((int*)spparks_extract(*model, "dimension"));
    int *sites = ((int*) spparks_extract(*model, "nglobal"));
    int *atomicState = ((int*) spparks_extract(*model, "site"));
    double **atomicGrid = ((double**) spparks_extract(*model, "xyz"));
    double roughness = 0;

    /* 3 Dimensional Roughness */
    if(*dimension == 3){
        map<double, map<double, double> > xlattice;
        double **start = atomicGrid;

        for(int *begin = atomicState, *end = atomicState + *sites;begin !=
                  end; ++begin, ++start){
            map<double, double> *ylattice = &xlattice[**start];
            double zlattice = (*ylattice)[*(*start + 1)];
            if (*begin == 2){
                if (*(*start + 2) > zlattice)
                    (*ylattice)[*(*start + 1)] = *(*start + 2);
            }
        }

        double sumheight = 0;
        for (map <double, map <double, double> >::iterator i =
                  xlattice.begin(); i != xlattice.end(); ++i){
```

```
        map <double, double> temp = (*i).second;
        for (map <double, double>::iterator j = temp.begin(); j !=
              temp.end(); ++j){
            sumheight += (*j).second;
        }
    }
    double averageheight = sumheight / (xlattice.size()*xlattice.begin()-
              >second.size());

    //Calculate roughness
    double roughnessnumer = 0;
    ctr = 0;
    for (map <double, map <double, double> >::iterator i =
              xlattice.begin(); ctr < ((int)xlattice.size() - segmentsize
              + 1); ++i, ++ctr){
        map <double, double> temp = (*i).second;
        ctr2 = 0;
        for (map <double, double>::iterator j = temp.begin(); ctr2 <
              ((int)xlattice.begin()->second.size() - segmentsize + 1);
              ++j, ++ctr2){
            sumsegmentheight = 0;
            ctr3 = 0;
            for (map <double, map<double, double> >::iterator i2 = i;
                    ctr3 < segmentsize; ++i2, ++ctr3){
                map <double, double> temp2 = (*i2).second;
                ctr4 = 0;
                for (map <double, double>::iterator j2 = temp2.begin();
                          ctr4 < segmentsize; ++j2, ++ctr4){
                    for (ctr5 = 0; ctr5 < ctr2 && ctr4 < 1; ctr5++) j2++;
                    sumsegmentheight += j2->second;
                }
            }
            averagesegmentheight = sumsegmentheight / (segmentsize *
                    segmentsize);
            roughnessnumer += pow((averagesegmentheight- averageheight),
                    2.0);
        }
    }

    roughness = sqrt(roughnessnumer / ((xlattice.size() - segmentsize +
        1) * (xlattice.begin()->second.size() - segmentsize + 1)));
}
/* 2 Dimensional Roughness */
else if(*dimension == 2){
    map<double, double> lattice;
    double **start = atomicGrid;

    for(int *begin = atomicState, *end = atomicState + *sites ;begin !=
              end; ++begin, ++start){
        double y = lattice[**start];
        if (*begin == 2){
            if (*(*start + 1) > y)
                lattice[**start] = *(*start + 1);
        }
    }

    double sumheight = 0;
```

```
        map<double, double>::iterator end = lattice.end();
        for (map<double, double>::iterator it = lattice.begin(); it != end;
                  ++it)
                sumheight += it->second;

        double averageheight = sumheight / (lattice.size());

        //Calculate roughness
        double roughnessnumer = 0;
        ctr2 = 0;
        for (map<double, double>::iterator it = lattice.begin(); ctr2 <
                  ((int)lattice.size() - segmentsize + 1); ++it, ++ctr2){
                sumsegmentheight = 0;
                ctr = 0;
                for (map<double, double>::iterator it2 = it; ctr <
                        segmentsize; ++it2, ++ctr){
                    sumsegmentheight += it2->second;
                }
                averagesegmentheight = sumsegmentheight / segmentsize;
                roughnessnumer += pow((averagesegmentheight - averageheight),
                  2.0);
            }
        roughness = sqrt(roughnessnumer / (lattice.size() - segmentsize +
            1));
    }
    return roughness;
}
```

C.3.4 Growth Calculator

```
#ifndef GROWTH_H_INCLUDED
#define GROWTH_H_INCLUDED

using namespace std;

double growth(double netDepRate, double depRate);

#endif // GROWTH_H_INCLUDED
```

```
#include "growth.h"

double growth(double netDepRate, double depRate){
    return netDepRate / depRate;
};
```

C.4 Simulation Manipulations

C.4.1 Create Model Simulation Input File

```
/* Create a SPPARKS input file for the model simulations. This function takes
a SPPARKS input file for the real simulation and modifies the commands
according to the given denominator, which is the ratio of the size of the
real simulation lattice to the size of the desired reduce lattice size. */

#ifndef SMALLKMCINPUT_H_INCLUDED
#define SMALLKMCINPUT_H_INCLUDED
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <sstream>
#include <stdlib.h>

using namespace std;

void smallKMCInput(string inputFilename, string newInputFilename, int
denominator);

#endif // SMALLKMCINPUT_H_INCLUDED


#include "smallKMCInput.h"
void smallKMCInput(string inputFilename, string newInputFilename, int
denominator){
    cout <<endl<< "Creating small KMC input file ..." <<endl<<endl;
    int dimension = 0;
    double xlo, xhi, ylo, yhi, zlo, zhi;
    double rate;
    int x, y, z, d, lo, hi;
    ifstream inputFile(inputFilename.c_str());
    ofstream newInputFile(newInputFilename.c_str());
    if(!inputFile.is_open()){
            cerr << "Error opening SPPARKS Input file" << endl;
                  exit(1);
    }
    if(!newInputFile.is_open()){
            cerr << "Error opening SPPARKS small KMC Input file" << endl;
                  exit(1);
    }
      while(!inputFile.eof()){
        string line;
          getline(inputFile,line);
          stringstream currentLine(line);
          if(line.find("dimension") != string::npos){
              string discard;
            currentLine >> discard;
            currentLine >> dimension;
            newInputFile << line <<endl;
          }
          else if(line.find("deposition") != string::npos){
              string discard;
            currentLine >> discard;
            currentLine.precision(30);
            currentLine >>rate>>x>>y>>z>>d>>lo>>hi;
            switch(dimension){
                case 2:{
                    newInputFile <<"deposition"<<'\t';
                    newInputFile.precision(30);
                    newInputFile << rate / denominator <<" ";
                    newInputFile <<x<<" "<<y<<" "<<z<<" "<<d<<" "<<" "<<lo<<"
                        "<<hi<<endl;
                    break;
                }
                case 3:{
                    newInputFile <<"deposition"<<'t'<< rate /
                        (denominator*denominator) <<" ";
                    newInputFile <<" "<<x<<" "<<y<<" "<<z<<" "<<d<<" "<<lo<<"
                        "<<hi<<" "<<endl;
                }
            }
          }
          else if(line.find("set") != string::npos){
            continue;
          }
          else if(line.find("region") != string::npos){
            stringstream newLine;
            string region;
            string regionName;
            string regionArgument;
            currentLine >> region>>regionName >> regionArgument;
            currentLine >>xlo>>xhi>>ylo>>yhi>>zlo>>zhi;
            newLine.precision(11);
            switch(dimension){
                case 2:{
                    newLine <<region<<'\t'<<'\t'<<" "<<regionName<<"
                        "<<regionArgument<<" ";
                    newLine <<xlo<<" "<<xhi/denominator;
                    newLine <<" "<<ylo<<" "<<yhi<<" "<<zlo<<" "<<zhi<<endl;
                    break;
                }
                case 3:{
                    newLine <<region<<'\t'<<'\t'<<regionName<<"
                        "<<regionArgument<<" ";
                    newLine <<xlo<<" "<<xhi/denominator<<" ";
                    newLine <<ylo<<" "<<yhi/denominator<<" "<<zlo<<"
                        "<<zhi<<endl;
                    break;
                }
            }
            newInputFile << newLine.str();
            }
            else{
            newInputFile << line <<endl;
            }
      }
      inputFile.close();
      newInputFile.close();}
```

C.4.2 Atomic Arrangement Approximation

```
/* This function crops a 'fraction' of the atomic arrangement of the supplied
SPPARKS simulation from the (0,0,0) coordinate of the simulation the
specified value calculated as (Lattice sites of the SPPARKS
Simulation)/Fraction. This function outputs a datafile of name filename to be
used by the read_sites command by SPPARKS */

#ifndef SMALLSITESDATA_H_INCLUDED
#define SMALLSITESDATA_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include "library.h"
#include "mpi.h"
using namespace std;

void smallSitesData(void **spk, const char *filename, int fraction);

#endif // SMALLSITESDATA_H_INCLUDED
```

```
#include "smallSitesData.h"

void smallSitesData(void **spk, const char *filename, int fraction){
    srand(time(0));
    cout <<endl<<"Creating sites for small KMC model"<<endl<<endl;
    /* SPPARKS simulation state variables */
    const int *sites = ((int*) spparks_extract(*spk, "nglobal"));
    const int dimension = *((int*)spparks_extract(*spk, "dimension"));
    int *id = ((int*) spparks_extract(*spk, "id"));
    int *atomicState = ((int*) spparks_extract(*spk, "site"));
    double **atomicGrid = ((double**) spparks_extract(*spk, "xyz"));
    double xlo = *((double*) spparks_extract(*spk, "boxxlo"));
    double xhi = *((double*)spparks_extract(*spk, "boxxhi"));
    double ylo = *((double*) spparks_extract(*spk, "boxylo"));
    double yhi = *((double*) spparks_extract(*spk, "boxyhi"));
    double zlo = *((double*)spparks_extract(*spk, "boxzlo"));
    double zhi = *((double*)spparks_extract(*spk, "boxzhi"));

    /* Open modelData file to write sites data into */
    ofstream siteData(filename,fstream::trunc);
    if(!siteData.is_open()){
            cout << "Error opening SPPARKS Site Data file" << endl;
                  exit(1);
    }

    vector<int> siteID;
    vector<int> siteType;
    int newXSize = xhi / fraction;
int newYSize = yhi / fraction;
int newSites = 1;
int *state = atomicState;
double **grid = atomicGrid;
switch(dimension){
    case 2:{
        int firstX = 0;
        for(int*iterator=id,*end=id+*sites;iterator!=end;++iterator,
              ++state,++grid){
            if(**grid >= firstX && **grid < (firstX + newXSize)){
                if((**grid >=(firstX + newXSize - 1)&&*(*grid + 1)== 0) |
                //site to the left of 0,0
                   (**grid == firstX + 1 && *(*grid + 1) == 0) |
                   //site to the right of 0,0
                   (**grid==firstX&&*(*grid + 1)<=3&&*(*grid + 1)!= 0)|
                   //site above 0,0
                   (**grid>=(firstX + newXSize - 1)&&*(*grid + 1)<= 3) |
                   //site upperleft of 0,0
                   (**grid<=firstX+1&&*(*grid+1)<=3&&**grid!=firstX)){
                   //site upperright of 0,0
                    siteID.push_back(newSites);
                    siteType.push_back(3);
                    ++newSites;
                }
                else{
                    siteID.push_back(newSites);
                    siteType.push_back(*state);
                     ++newSites;
                }
            }
        }
        break;
    }
    case 3:{
        int firstX = 0;
        int firstY = 0;
        for(int*iterator=id,*end=id+*sites;iterator!=end;++iterator,
              ++state, ++grid){
            if(**grid>=firstX&&*(*grid+1)>=firstY&&**grid<=(firstX+
                    newXSize)&&*(*grid+1)<=(firstY+newYSize)){
                   if(**grid>=(firstX+newXSize-2)&&*(*grid+2)==(firstY+
                          newYSize-2)){
                        siteID.push_back(newSites);
                        siteType.push_back(3);
                        ++newSites;
                   }
                   else{
                        siteID.push_back(newSites);
                        siteType.push_back(*state);
                        ++newSites;
                   }
            }
        }
        break;
    }
}
  siteData.precision(14);
```

```
      siteData << "Skip This Line" << endl<<endl;
      siteData << dimension << " dimension" << endl<<endl;
      siteData << newSites-1 << " sites"<<endl<<endl;
      switch(dimension){
        case 2:{
            siteData << xlo << " " << newXSize << " xlo xhi" << endl;
            siteData << ylo << " "<< yhi << " ylo yhi" << endl;
            break;
        }
        case 3:{
            siteData << xlo << " " << newXSize << " xlo xhi" << endl;
            siteData << ylo << " "<< newYSize << " ylo yhi" << endl;
            break;
        }
      }
    siteData << zlo << " " << zhi << " zlo zhi" << endl<<endl;
    siteData << endl << "Values" << endl << endl;
    for(int i = 0; i < newSites - 1; i++){
        siteData << siteID[i] <<" "<< siteType[i] <<endl;
    }
};
```

C.4.3 Properties Manipulator: Model Predictive Control

```
/* Call the property calculators and organize the data for manipulation for
the Model Predictive Control Simulation. The properties are calculated from
the SPPARKS simulation, and a vector of the properties, {Roughness, Growth
Rate, Porosity}, is returned*/

#ifndef SIMULATIONPOINTS_H_INCLUDED
#define SIMULATIONPOINTS_H_INCLUDED

#include <vector>
#include "roughness.h"
#include "growth.h"
#include "porosity.h"

using namespace std;

vector<double> simulationPoints(void **spk, double netDepostionRate, double
depositionRate,int segmentsize);

#endif // SIMULATIONPOINTS_H_INCLUDED
```

```
#include "simulationPoints.h"

vector<double> simulationPoints(void **spk, double netDepRate, double
depRate,int segmentsize){
    vector<double> toReturn(3,0);
    toReturn[0] = roughness(spk,segmentsize);
    toReturn[1] = growth(netDepRate,depRate);
    toReturn[2] = porosity(spk);
    return toReturn;
};
```

C.4.4 Reference Trajectory Manipulator: Model Predictive Control

```
/* This function manipulates data in the reference trajectory to return a
vector of property set points, {Roughness, Growth Rate, Porosity}, for use in
the Model Predictive Control Simulation. Linear interpolations will be used
if a specific point is not found in the trajectory.  If the list of set
points in the reference trajectory file is exhausted, this function returns
the vector {-1, -1, -1}. This signals that the end of the profile has been
reached signaling to the Model Predictive Controller to stop the simulation.
*/

#ifndef SETPOINTS_H_INCLUDED
#define SETPOINTS_H_INCLUDED

#include <iostream>
#include <iomanip>
#include <vector>
#include <fstream>
#include <limits>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

vector<double> setPoints(const double &xa, const char *reference);

#endif // SETPOINTS_H_INCLUDED


#include "setPoints.h"
```

```
vector<double> setPoints(const double &xa, const char *referenceTrajectory){
  /*Variables needed for possible interpolation in the form (x1,y1) and (x2,y2)*/
  double x1 = 0, x2 = 0;
  vector<double> properties1(3,0);
  vector<double> properties2(3,0);
  int bad_input = 0;
  ifstream reference;
  /*Open the reference trajectory file*/
  reference.open(referenceTrajectory, fstream::out);
  if(!reference.is_open()){
    cout << "Failed to open reference trajectory." << endl;
    reference.close();
    exit(1);
  }
  /*Ignore the first line*/
    reference.ignore(numeric_limits<streamsize>::max(),'\n');
    /*Loop through the entire file looking for the correct time interpolate
`     if necesary*/
    while(!reference.eof()){
        do{
            //cout<<"Looking for next x2..."<<endl;
            bad_input = 0;
            reference.precision(30);
            reference >> x2;
            if(!reference){
                bad_input = 1;
                reference.clear();
                reference.ignore(numeric_limits<streamsize>::max(),' ');
            }
        }while(bad_input);
        for(int i = 0; i < 3; ++i){
            double prop;
            do{
                bad_input = 0;
                reference.precision(30);
                reference >> prop;
                if(!reference){
                    bad_input = 1;
                    reference.clear();
                    reference.ignore(numeric_limits<streamsize>::max(),' ');
                }
            }while(bad_input);
            properties2[i] = prop;
        }
        if(xa == x2){
            reference.close();
            cout<<endl;
            return properties2;
        }
        else if (x1 <= xa && xa <= x2){
            vector<double> toReturn(3);
            for(int k = 0; k < 3; ++k)
                toReturn[k] = properties1[k] + (xa-x1)/(x2-
                  x1)*(properties2[k] - properties1[k]);
            reference.close();
            cout<<endl;
            return toReturn;
        }
        /*Ignore the rest of this line*/
        reference.ignore(numeric_limits<streamsize>::max(),'\n');
        x1 = x2;
        properties1 = properties2;
        /*A Blank line indicated by immediate '\n' character indicates
    *following this line is the end of reference data

         */
        if(reference.peek() == '\n')
            break;
    }
    reference.close();
      /* Upon returning the default vector the MPC will stop due to an
      indication of a failed attempt to retrieve data or the end of the
      reference trajectory */
    vector<double> defaultReturn(3,-1);
    return defaultReturn;
};
```

C.5 SPPARKS Post-Processing

C.5.1 SPPARKS Dump to Profile

```
#include <iostream>
#include "DumpToProfile.h"

#define MAXLENGTHFILE 100

using namespace std;

int main()
{
    char DumpFile[MAXLENGTHFILE] = "dump.GaAs.1000wide.unitstep";
    int dimension = 2;
    char OutputFile[MAXLENGTHFILE] = "profile.GaAs.1000wide.unitstep.txt";
    int segmentsize = 10;
    double deprate = 0.0000000002;

    cout << "Please wait while profile is generated" << endl;

  //  DumpToProfile("dump.GaAs.1000wide.porositydisturbed", dimension,
"profile.GaAs.1000wide.porositydisturbed.txt", segmentsize, deprate);
    DumpToProfile("dump.GaAs.1000wide.unitstep", dimension,
"profile.GaAs.1000wide.unitsteptxt", segmentsize, deprate);

    return 0;
}


#ifndef DUMPTOPROFILE_H_INCLUDED
#define DUMPTOPROFILE_H_INCLUDED

#include <string.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "DumpObjectiveFunctionCalculator.h"

#include <map>

void DumpToProfile(char *DumpFile, int dim, char *OutputFile, int
segmentsize, double deprate);

#endif // DUMPTOPROFILE_H_INCLUDED
```

```
//Accepts dump file in the form id, type, x, y, z with 1 or more timesteps,
and prints a text file with timestep
//in the first column, roughness of that timestep in the second column,
growth rate in the third column.
//and porosity in the fourth column. Roughness is as defined in

//"Dynamics and control of thin film surface microstructure in a complex
deposition process" by "Dong Ni et. al."
//The denominator is the number of lattice sites in x (2D) or in x and y
(3D).
//If 2-D growth on a 1-D substrate, numerator of equation has only 1
summation, and denominator has only 1 N.
//Cannot compare roughness between simulations of different size lattice.
Does not do 1D.
//Do not compare roughnesses between lattices of different types (apples and
oranges) because,
//for example, a perfectly flat sc6n will have a roughness of 0, but a
perfectly flat fcc will have a roughness greater
//than 0, simply because not all sites at the top of an fcc lattice have the
same height.
//Bottom row(s) (rows if the bottom row does not contain all x or x,y values)
must be filled with type 2.

#include "DumpToProfile.h"
using namespace std;

void DumpToProfile(char *DumpFile, int dim, char *OutputFile, int
segmentsize, double deprate){
    FILE *fpinputdump, *fpoutput;
    const int MAXBUFFER = 100;
    char buffer[MAXBUFFER];
    int ctr = 0, ctr2 = 0, ctr3 = 0, ctr4 = 0, ctr5 = 0, ctrtimesteps = 0,
            sites;
    long position;
    double timestep = 0, timestep_last, xmin, xmax, ymin, ymax, zmin, zmax,
            sumheight, sumsegmentheight, averageheight, averagesegmentheight,
            roughnessnumer, roughness, netdeprate, porosity, sites_filled =
            0, sites_filled_last, sites_filled_initial;

    //Open for reading the input dump file
    if ((fpinputdump = fopen(DumpFile, "r")) != NULL){      ;}
    else{
        fprintf(stderr, "\nError opening read file\n");
        fcloseall();
        exit(1);
    }

    //Open for writing the output file, unless user entered a "no"
    if (strcasecmp(OutputFile, "no") != 0){
        if ((fpoutput = fopen(OutputFile, "w")) != NULL){
                ;
            }
        else{
            fprintf(stderr, "\nError opening write file\n");
            fcloseall();
            exit(1);
        }
```

```
}

//Header of output file
fprintf(fpoutput, "timestep roughness growthrate porosity
  objectivefunction\n");

while (!feof(fpinputdump))
{
    //Skip first line, while testing it for a blank which would signal
      end of file, Get timestep, then Skip line
    fgets(buffer, MAXBUFFER, fpinputdump);
    if (feof(fpinputdump))
    {
        goto locationend;
    }
    fgets(buffer, MAXBUFFER, fpinputdump);
    timestep_last = timestep;
    timestep = atof(strchr(buffer, ' '));
    fgets(buffer, MAXBUFFER, fpinputdump);

    //Get number of sites and box bounds
    fgets(buffer, MAXBUFFER, fpinputdump);
    sites = atoi(buffer);
    fgets(buffer, MAXBUFFER, fpinputdump);
    fgets(buffer, MAXBUFFER, fpinputdump);
    xmin = atof(buffer);
    xmax = atof(strchr(buffer, ' '));
    fgets(buffer, MAXBUFFER, fpinputdump);
    ymin = atof(buffer);
    ymax = atof(strchr(buffer, ' '));
    fgets(buffer, MAXBUFFER, fpinputdump);
    zmin = atof(buffer);
    zmax = atof(strchr(buffer, ' '));
    fgets(buffer, MAXBUFFER, fpinputdump);

    //Save file's position indicator
    position = ftell(fpinputdump);

    //Reset sites_filled value for each timestep. Remember last value of
    sites_filled
    sites_filled_last = sites_filled;
    sites_filled = 0;

    if (dim == 2){
        map <double, double> lattice;
        for (ctr = 0; ctr < sites; ctr++)
        {
            double x, y, z;
            int type;
            fgets(buffer, MAXBUFFER, fpinputdump);
            type = atoi(strchr(buffer, ' '));
            x = atof(strchr(strchr(buffer, ' ')+1,' '));
            y = atof(strchr(strchr(strchr(buffer, ' ')+1,' ')+1,' ')+1);
            z = atof(strchr(strchr(strchr(strchr(buffer,' ')+1,' ')+ 1,'
            ')+ 1,' ')+1);

            double tmpy = lattice[x];
            if (type == 2){
                if (y > tmpy){
                    lattice[x] = y;
                }
            }

            //Get number of filled sites for growth rate
            if (type ==2){
                sites_filled += 1;
            }
        }

        //Calculate average height
        sumheight = 0;
        map<double, double>::iterator end = lattice.end();
        for (map<double, double>::iterator it = lattice.begin(); it !=
              end; ++it){
            sumheight += it->second;
        }
        averageheight = sumheight / (lattice.size());

        //Calculate roughness
        roughnessnumer = 0;
        ctr2 = 0;
        for (map<double,double>::iterator it =lattice.begin();ctr2<
              ((int)lattice.size()-segmentsize+1);++it,++ctr2){

                sumsegmentheight = 0;
                ctr = 0;
                for (map<double, double>::iterator it2 = it; ctr <
                          segmentsize; ++it2, ++ctr){
                    sumsegmentheight += it2->second;
                }
                averagesegmentheight = sumsegmentheight / segmentsize;
                roughnessnumer += pow((averagesegmentheight -
                    averageheight), 2.0);
            }
        roughness=sqrt(roughnessnumer/(lattice.size()-segmentsize+
              1));

        //Porosity calculation
        double unfilled = 0;
        double total = 0;
        if (fseek(fpinputdump, position, SEEK_SET) != NULL)
        {
            fprintf(stderr, "\nError using fseek().");
            fcloseall();
            exit(1);
        }
        for (ctr = 0; ctr < sites; ctr++)
        {
            double x, y, z;
            int type;

            fgets(buffer, MAXBUFFER, fpinputdump);
            type = atoi(strchr(buffer,' '));
```

```
           x = atof(strchr(strchr(buffer,' ')+1,' '));
           y = atof(strchr(strchr(strchr(buffer,' ')+1,' ')+1,' ')+1);
           z = atof(strchr(strchr(strchr(strchr(buffer, ' ')+1,' ')+1,'
             ')+1,' ')+1);
           for (map<double, double>::iterator it=lattice.begin();
                   it!=end; ++it){
               if ((x == it->first)&&(type == 1)&&(y <= it->second)){
                   unfilled++;
               }
               if ( (x == it->first) && (y <= it->second)){
                   total++;
               }
           }
       }
       porosity = unfilled / total;
   }

if (dim == 3){
       map<double, map<double, double> > xlattice;
       for (ctr = 0; ctr < sites; ctr++){
           fgets(buffer, MAXBUFFER, fpinputdump);
           int type = atoi(strchr(buffer,' '));
           double x = atof(strchr(strchr(buffer,' ')+1,' '));
           double y = atof(strchr(strchr(strchr(buffer,' ')+1,' ')+1,'
                   ')+1);
           double z = atof(strchr(strchr(strchr(strchr(buffer, ' ')+1,'
                   ')+1,' ')+1,' ')+1);
           map<double, double> *ylattice = &xlattice[x];
           double zlattice = (*ylattice)[y];

            if (type == 2){
               if (z > zlattice){
                   (*ylattice)[y] = z;
               }
           }

           //Get number of filled sites for growth rate
           if (type ==2){
               sites_filled += 1;
           }
       }

       //Calculate average height
       sumheight = 0;
       for (map <double, map <double, double> >::iterator i =
             xlattice.begin(); i != xlattice.end(); ++i){
           map <double, double> temp = (*i).second;
           for (map <double, double>::iterator j = temp.begin(); j !=
                   temp.end(); ++j){
               sumheight += (*j).second;
           }
       }
       averageheight = sumheight / (xlattice.size()*xlattice.begin()-
             >second.size());

       //Calculate roughness
       roughnessnumer = 0;
```

```
ctr = 0;
for (map <double, map <double, double> >::iteratori=
      xlattice.begin();ctr<((int)xlattice.size()-
      segmentsize+1);++i,++ctr){
    map <double, double> temp = (*i).second;
    ctr2 = 0;

    for (map <double, double>::iterator j = temp.begin(); ctr2 <
            ((int)xlattice.begin()->second.size() - segmentsize +
            1); ++j, ++ctr2){
        sumsegmentheight = 0;
        ctr3 = 0;
        for (map <double, map<double, double> >::iterator i2 = i;
            ctr3 < segmentsize; ++i2, ++ctr3){
            map <double, double> temp2 = (*i2).second;
            ctr4 = 0;
            for (map <double, double>::iterator j2 =
                  temp2.begin(); ctr4 < segmentsize; ++j2,
                  ++ctr4){
                for (ctr5 = 0; ctr5 < ctr2 && ctr4 < 1; ctr5++)
                        j2++;
                  sumsegmentheight += j2->second;
            }
        }
        averagesegmentheight = sumsegmentheight / (segmentsize *
            segmentsize);

        roughnessnumer += pow((averagesegmentheight-
            averageheight), 2.0);
    }
}
roughness = sqrt(roughnessnumer / ((xlattice.size() - segmentsize
      + 1) * (xlattice.begin()->second.size() - segmentsize +
1)));

//Porosity calculation
double unfilled = 0;
double total = 0;
if (fseek(fpinputdump, position, SEEK_SET) != NULL)
{
    fprintf(stderr, "\nError using fseek().");
    fcloseall();
    exit(1);
}
for (ctr = 0; ctr < sites; ctr++)
{
    double x, y, z;
    int type;

    fgets(buffer, MAXBUFFER, fpinputdump);
    type = atoi(strchr(buffer, ' '));
    x = atof(strchr(strchr(buffer,' ')+1,' '));
    y = atof(strchr(strchr(strchr(buffer,' ')+1,' ')+1,' ')+1);
    z = atof(strchr(strchr(strchr(strchr(buffer,' ')+1,' ')+1,'
            ') + 1, ' ') + 1);
    for (map <double, map <double, double> >::iterator i =
            xlattice.begin(); i != xlattice.end(); ++i)
```

```
                {
                    map <double, double> temp = (*i).second;
                    for (map <double, double>::iterator j = temp.begin(); j
                        != temp.end(); ++j)
                    {
                        if ( (x == i->first) && (y == j->first) && (type ==
                              1) && (z <= j->second) )
                        {
                            unfilled++;
                        }
                        if ( (x == i->first) && (y == j->first) && (z <= j-
                              >second))
                        {
                            total++;
                        }
                    }
                }
            }
            porosity = unfilled / total;
            netdeprate = (sites_filled - sites_filled_last) / (timestep -
                  timestep_last);
            if (ctrtimesteps == 0) sites_filled_initial = sites_filled;
                  ctrtimesteps++;

            if (strcasecmp(OutputFile, "no") != 0){
                  fprintf(fpoutput, "%lf %lf %.20lf %lf %lf\n", timestep,
                  roughness, netdeprate, porosity,
                  DumpObjectiveFunctionCalculator(roughness, netdeprate,
                  deprate, porosity));
        }
        printf("Done with snapshot %d\n", ctrtimesteps - 1);
    }

    locationend: ;

    if (strcasecmp(OutputFile, "no") != 0)
    {
        netdeprate = (sites_filled - sites_filled_initial) / timestep;
        fprintf(fpoutput, "Cumulative Profile: %lf %.20lf %lf %lf\n",
            roughness, netdeprate, porosity,DumpObjectiveFunctionCalculator(
            roughness, netdeprate, deprate, porosity));
    }

    fcloseall();
}
```

C.5.2 Slice Machine: 3-D Simulation to 2-D Images

```
#include <iostream>
#include "SliceMachine.h"

#define MAXLENGTHFILE 100
#define MAXLENGTHBUFFER 100

using namespace std;

int main()
{
    //User Inputs
    char inputdump[MAXLENGTHFILE] = "dump.Experiment39";
    const char constantcross = 'z';
    const double crossselect = 1.0; //If constantcross is x or y, this is the
    cross-section selection. If constantcross is z, this is the z at which it
    will cutt off all z above or at this point. For z, I recommend selecting
    the next highest point.


    char outputdump[MAXLENGTHBUFFER];
    if (constantcross == 'x' || constantcross == 'y') sprintf(outputdump,
            "%s.constant%c=%.1lf",inputdump, constantcross, crossselect);
    if (constantcross == 'z') sprintf(outputdump, "%s.%c<%.1lf",inputdump,
            constantcross, crossselect);
    SliceMachine(inputdump, outputdump, constantcross, crossselect);

    return 0;

}
```

```
#ifndef DUMPTOPROFILE_H_INCLUDED
#define DUMPTOPROFILE_H_INCLUDED

#include <string.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <map>

void SliceMachine(char *DumpFile, char *OutputFile, char constantcross,
double crossselect);

#endif // DUMPTOPROFILE_H_INCLUDED
```

```
//Accepts dump file in the form id, type, x, y, z with 1 or more timesteps,
and prints a text file with timestep
```

```
//in the first column, roughness of that timestep in the second column,
growth rate in the third column.
//and porosity in the fourth column. Roughness is as defined in

//"Dynamics and control of thin film surface microstructure in a complex
deposition process" by "Dong Ni et. al."
//The denominator is the number of lattice sites in x (2D) or in x and y
(3D).
//If 2-D growth on a 1-D substrate, numerator of equation has only 1
summation, and denominator has only 1 N.
//Cannot compare roughness between simulations of different size lattice.
Does not do 1D.
//Do not compare roughnesses between lattices of different types (apples and
oranges) because,
//for example, a perfectly flat sc6n will have a roughness of 0, but a
perfectly flat fcc will have a roughness greater
//than 0, simply because not all sites at the top of an fcc lattice have the
same height.
//Bottom row(s) (rows if the bottom row does not contain all x or x,y values)
must be filled with type 2.

#include "SliceMachine.h"
using namespace std;

void SliceMachine(char *DumpFile, char *OutputFile, char constantcross,
double crossselect){
    FILE *fpinputdump, *fpoutputdump;
    const int MAXBUFFER = 100;
    char buffer[MAXBUFFER];
    long position;
    int constant;

    if (constantcross == 'x') constant = 1;
    else if (constantcross == 'y') constant = 2;
    else if (constantcross == 'z') constant = 3;
    else fprintf(stderr, "Constant cross must be x or y");

    //Open for reading the input dump file
    if ((fpinputdump = fopen(DumpFile, "r")) != NULL){  ;}
    else{
        fprintf(stderr, "\nError opening read file\n");
        fcloseall();
        exit(1);
    }

    //Open for writing the output file
    if ((fpoutputdump = fopen(OutputFile, "w")) != NULL){    ;}
    else{
        fprintf(stderr, "\nError opening write file\n");
        fcloseall();
        exit(1);
    }

    //Read the input dump's first timestep to figure out how many atoms there
      will be and what box bounds to use.
    for (int ctr = 0; ctr < 4; ctr++) fgets(buffer, MAXBUFFER, fpinputdump);
```

```
int totalsites = atoi(buffer);
for (int ctr = 0; ctr < 2; ctr++) fgets(buffer, MAXBUFFER, fpinputdump);
double xmin = atof(buffer);
double xmax = atof(strchr(buffer, ' '));
fgets(buffer, MAXBUFFER, fpinputdump);
double ymin = atof(buffer);
double ymax = atof(strchr(buffer, ' '));
fgets(buffer, MAXBUFFER, fpinputdump);
double zmin = atof(buffer);
double zmax = atof(strchr(buffer, ' '));
fgets(buffer, MAXBUFFER, fpinputdump);
int subsites = 0;

for (int ctr = 0; ctr < totalsites; ctr++)
{
    fgets(buffer, MAXBUFFER, fpinputdump);
    double x = atof(strchr(strchr(buffer, ' ')+1,' '));
    double y = atof(strchr(strchr(strchr(buffer,' ')+1,' ')+1,' ')+1);
    double z = atof(strchr(strchr(strchr(strchr(buffer,' ')+1,' ')+1,' ')
        +1,' ')+1);
    switch (constant)
    {
        case 1:
            if (x == crossselect) subsites++;
            break;
        case 2:
            if (y == crossselect) subsites++;
            break;
        case 3:
            if (z < crossselect) subsites++;
    }
}
rewind(fpinputdump);

while (!feof(fpinputdump)){
    //Skip first line, while testing it for a blank which would signal

    end of file, Get timestep, then Skip line
    fgets(buffer, MAXBUFFER, fpinputdump);
    if (feof(fpinputdump)){
        goto locationend;
    }
    fprintf(fpoutputdump, "%s", buffer);
    for (int ctr = 0; ctr < 2; ctr++){
        fgets(buffer, MAXBUFFER, fpinputdump);
        fprintf(fpoutputdump, "%s", buffer);
    }
    fgets(buffer, MAXBUFFER, fpinputdump);
    fprintf(fpoutputdump, "%d\n", subsites);
    fgets(buffer, MAXBUFFER, fpinputdump);
    fprintf(fpoutputdump, "%s", buffer);
    switch (constant){
        case 1:{
            fprintf(fpoutputdump, "%lf %lf\n", ymin, ymax);
            fprintf(fpoutputdump, "%lf %lf\n", zmin, zmax);
            fprintf(fpoutputdump, "0 0\n");
            break;
```

```
                }
            case 2:{
                fprintf(fpoutputdump, "%lf %lf\n", xmin, xmax);
                fprintf(fpoutputdump, "%lf %lf\n", zmin, zmax);
                fprintf(fpoutputdump, "0 0\n");
                break;
                }
            case 3:{
                fprintf(fpoutputdump, "%lf %lf\n", xmin, xmax);
                fprintf(fpoutputdump, "%lf %lf\n", ymin, ymax);
                fprintf(fpoutputdump, "%lf %lf\n", zmin, crossselect);
            }
        }

        for (int ctr = 0; ctr < 4; ctr++)
            fgets(buffer, MAXBUFFER, fpinputdump);
        fprintf(fpoutputdump, "%s", buffer);

        for (int ctr = 0; ctr < totalsites; ctr++)
        {
            fgets(buffer, MAXBUFFER, fpinputdump);
            int id = atoi(buffer);
            int type = atoi(strchr(buffer, ' '));
            double x = atof(strchr(strchr(buffer,' ')+1,' '));
            double y = atof(strchr(strchr(strchr(buffer,' ')+1,' ')+1,' ')+
                  1);
            double z = atof(strchr(strchr(strchr(strchr(buffer,' ')+1,' ')+
                  1,' ')+1,' ')+1);
            switch (constant){
                case 1:
                    if (x == crossselect) fprintf(fpoutputdump, "%d %d %lf
                        %lf 0\n", id, type, y, z);
                    break;
                case 2:
                    if (y == crossselect) fprintf(fpoutputdump, "%d %d %lf
                        %lf 0\n", id, type, x, z);
                    break;
                case 3:
                    if (z < crossselect) fprintf(fpoutputdump, "%d %d %lf %lf
                        %lf\n", id, type, x, y, z);
            }
        }
    }
    locationend: ;
    fcloseall();
}
```

Appendix D SPPARKS

The SPPARKS software can be used to model several different types of processes, via the *app_style* command. For the purposes of this project, the *diffusion* application is used. It is an on-lattice application which performs diffusive hops on a lattice whose sites are occupied or unoccupied (vacant). It can be used to model diffusion on 2D or 3D lattices. It is equivalent to a 2-state Ising model with Kawasaki dynamics, in which neighboring sites exchange their spins (1 for vacant, 2 for occupied) as the model evolves. SPPARKS allows the user to specify how energy is used in computing the probability of executing a diffusive event through the *estyle* command, and which kind of diffusive hops are allowed via the *dstyle* command.

Three options are available for the *estyle* setting: *off*, *linear*, and *nonlinear*. They are related to the computation of the Hamiltonian of the system. If *estyle* is set to *off*, the Hamiltonian for an occupied or vacant site is set to zero, which simply means that energy is not used in calculating the probability of executing an event. The *barrier* command is used instead, to specify a diffusive hop barrier in units of energy. For the *estyle* setting *linear*, the Hamiltonian of an occupied site is defined as follows:

$$H_i = \sum_j \delta_{ij}, \qquad \textbf{Equation 20}$$

where δ_{ij} is zero if site *j* is occupied, and 1 is site *j* is vacant.

If *estyle* is set to *nonlinear*, the Hamiltonian of an occupied site is calculated via the following equation:

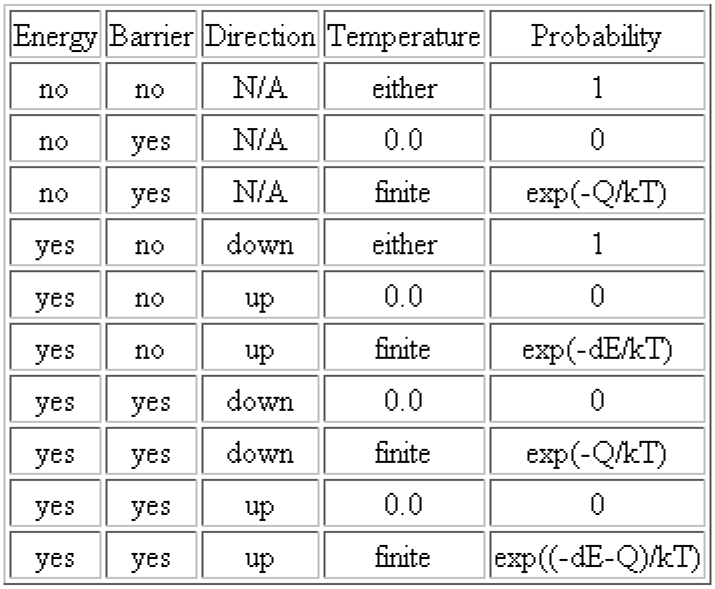
$$H_i = Eng\left(\sum_j \delta_{ij}\right), \qquad \textbf{Equation 21}$$

where δ_{ij} is 1 if site *j* is occupied and zero if site *j* is vacant. This is equivalent to calculating the coordination number of site *i*. The function *Eng()* is a tabulated function specified via the *ecoord* command which assigns a value in units of energy to an occupied site with a given coordination number. The Hamiltonian of a vacant site in all cases is zero.The energy of the system is computed as the sum of H_i over all sites *i*.

System temperature is specified via the *temperature* command, in units of energy. The probability of executing a diffusive event for different combination of parameters is given in Table 13 below.

The *dstyle* setting can be set to *hop*, which allows nearest neighbor hops only, or *schwoebel*, which allows second-nearest-neighbor (Schwoebel) hops as well. The energy barrier for a Schwoebel hop can also be set via the *barrier* command, with arguments *schwoebel*, and *value*.

Dimension for the simulation (2D or 3D) has to be defined via the *dimension* command. A region is created via the *region* command, which specifies a region ID and style. For the *diffusion* application, the region style is *block*, which takes minimum and maximum x, y, and z coordinate values, in units of length, as arguments. For a 2-dimensional simulation, the z_{min} and z_{max} are set to -0.5 and 0.5, respectively. After the region is defined, the simulation box is created via the *create_box* application, which takes the region ID as its argument. The *lattice* command defines the type of lattice structure, whose dimension must match that specified by the *dimension* command, and the *create_sites* command creates a site at every lattice point.



Energy	Barrier	Direction	Temperature	Probability
no	no	N/A	either	1
no	yes	N/A	0.0	0
no	yes	N/A	finite	exp(-Q/kT)
yes	no	down	either	1
yes	no	up	0.0	0
yes	no	up	finite	exp(-dE/kT)
yes	yes	down	0.0	0
yes	yes	down	finite	exp(-Q/kT)
yes	yes	up	0.0	0
yes	yes	up	finite	exp((-dE-Q)/kT)

Table 13: Probability of Executing a Diffusion Event. In column 1, *Energy* is *yes* if *estyle* setting is *linear* or *nonlinear*, and *no* if *estyle* setting is *off*. In column 2, *Barrier* is *yes* if an energy barrier is specified via the *barrier* command, and *no* otherwise. In column 3, *Direction* is *down* if the diffusive hop lowers the energy of the system, i.e. if the particle's coordination number increases after it migrates, and *up* if the hop increases the energy of the system, as discussed in the *estyle* section. In column 4, *Temperature* is *0.0* or *finite*, specified via the *temperature* command. Column 5 gives the probability of an event with a combination of parameters defined in columns 1-4. *Q* is the diffusive energy barrier specified by the *barrier* command, *dE* is the system energy change defined in the *estyle* section, *k* is the Boltzman constant, and *T* is the system temperature.

The *set* command can be used to set specific site values. The allowed site values for the *diffusion* application are 1 for a vacant site, 2 for an occupied site, and 3 for an occupied site of type which cannot diffuse or bind to particles of type 2. Site value 3 is useful in several ways. The simulation box has periodic boundary conditions in every direction, which means that particles in the bottom layer are allowed to diffuse to the top of the simulation box when diffusing down beyond the box's boundaries. Setting the sites at the top layer to value 3 prevents this from happening. Additionally, the *deposition* command discussed below requires that site (x,y,z) = (0,0,0) be occupied by a particle of type 2 throughout the simulation. This particle cannot migrate away from that site because this turns off the *deposition* command for the rest of the simulation.Once this happens, no more particles are deposited when that happens. Therefore, sites that are neighbors of site (0,0,0) are set to value 3, preventing the particle at (0,0.0) from diffusing. This is believed to be a software bug inherent to the construction of SPPARKS. This was an ad hoc solution that was found during development. Site values can also be specified via the *read_sites* command, which reads a provided datafile containing a matrix of site IDs and site values, and sets the site values in the simulation box accordingly.

The *diffusion* application allows the use of a *deposition* command, which has the following syntax:

deposition *rate dirx diry dirz d0 lo hi*

This command adds particles of type 2 to the simulation box at candidate deposition sites, at the specified *rate* inparticles per second. Deposition events compete with diffusion events. Every time a deposition event is selected, a random starting point at the top of the simulation box is chosen, and a trajectory is projected along an incident direction specified by the 3-dimensional vector *dirx, diry, dirz*. For a 3-dimensional simulation, it is required that $dirz < 0$. For a 2-dimensional simulation, $dirz = 0$, and $diry < 0$. Candidate deposition sites are characterized by the parameters *d0, lo,* and *hi*. *d0* specifies a radius, measured perpendicularly to the incident direction, within which a cadidate site for deposition canlie. *lo* and *hi* specify the minimum and maximum coordination number that a site is allowed to have, in order to qualify as a candidate deposition site. Of all the candidate sites for a selected event, the one closest to the starting point along the incident trajectory is selected for deposition.

The frequency with which deposition and diffusion events occur is given by Equations 3 and 4 in section 2.2 on page 14, where the rate of diffusion is specified by Equation 2, with the exponential term following the rules in Table 13 for the probability of executing a diffusion event. The pre-exponential A_m factor cannot be specified in SPPARKS. Therefore, all time components are scaled by this factor. This means that the supplied deposition rate is the actual deposition rate divided by A_m, and the specified simulation run time is the actual run time multiplied by A_m.

The *solve_style* command specifies a kMC solver to be used. The solver picks events to perform from a list of events and their associated probabilities, using a standard Gillespie or BKL algorithm, which also computes a timestep during which a chosen event occurs. The difference between the various solver styles available is the algorithm they use to select events, which affects their speed and scalability as a function of the number of events available to choose from. The *linear* style is suitable for simulations with few events, while the *tree* or *group* solers should be used for larger simulations.

The *diag_style, stats,* and *dump* commands specify the format of the SPPARKS simulation output. The *diag_style* command and the *stats* command output statistical information such as time elapsed, number of diffusion events, number of deposition events, and computational time.The *dump* command outputs surface information at every time step, such as site ID, (x,y,z) coordinates, and site value. Sandia National Laboratories provides the pizza.py toolkit, which can process and image a SPPARKS generated dumpfile. Additionally, the information contained in the dumpfile can be used to calculate roughness and porosity of the film, as well as generate MATLAB surface profiles. The programs that are used for these purposes are present in the following sections.