SIMULATION AND DEVELOPMENT OF NANOSCALE DEPOSITION TECHNIQUES USING KINETIC MONTE CARLO

by

COREY CLARK

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2007

ACKNOWLEDGMENTS

I would like to express my appreciation and gratitude to Dr. Choong-Un Kim as well as the rest of my committee for their guidance. I thank them for the opportunity and experience I gained while working on my dissertation at the University of Texas at Arlington. I would also like to thank God as well as my wife for their support and help throughout the entire graduate program, with out them I would have never made it through to the end.

December 11, 2006

ABSTRACT

SIMULATION AND DEVELOPMENT OF NANOSCALE
DEPOSITION TECHNIQUES USING
KINETIC MONTE
CARLO

Publication No._____

Corey Clark, Ph.D.

The University of Texas at Arlington, 2007

Supervising Professor: Choong-Un Kim

Modeling of deposition processes has become of extreme importance due to the small scale of devices and features as well as the reduction of time to market required by industry. Current modeling procedures have focused on individual aspects of growth as well as made assumptions that cause simplification of the problem to the point the models usefulness is limited. The Kinetic Monte Carlo (KMC) model developed in this work combines rate transitions that have been commonly found in KMC simulation along with energy density equations that help explain the transitions and formations of island in alloy deposition.

Specifically the model developed in this study, analyzes both the surface energy and strain energy of the film, which are incorporated to show the dependence of strain

relaxation to surface energy during island formation. The developed model also incorporates the anisotropy of crystalline structures to accommodate for the changes in growth rate and morphology based upon crystal orientation. This leads to a more versatile model that will accommodate multiple material sets as well allow for quick simulation results for the development of new devices. Work was also done in increasing the randomness of site selection while minimizing errors due to standard uniform number generators. The developed KMC model incorporates a pseudo random number generator for the purpose of site selection, which reduce the amount of cluster processing that can occur with random number generators. A focus was also placed on the ability to describe flux distributions that are not commonly found in semiconductor device manufacturing. This was done to allow for the expansion of this model into non-planar environments that might be found in industries such as MEMS/NEMS. This extension also allows for evaluation of non-planar deposition process such as via deposition.

The expansion done in this model allow for a wider variety of applications with in the semiconductor field. Accounting for crystal orientation allows for increased accuracy as well as further insight into its affect on island formation within alloy growth. The changes in random surface site selection during simulation have helped reduce cluster processing which allows for an accurate depiction of surface morphology.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF GRAPHS

CHAPTER 1

INTRODUCTION

Modeling of deposition process has become of extreme importance due to the small scale of devices and features as well as the reduction of time to market required by industry. An interest has been shown in industry for a model that can quickly and accurately describes surface features during deposition along with the flexibility to be used for multiple applications. Currently growth models focus on a single growth parameter and apply simplification that can often limit the usefulness of the model in other applications. A need has been demonstrated for a model that can couple the affects of multiple growth parameters and describe the surface morphology. The work done in this study will show the driving forces of various types of growth and how variations of those parameters will cause variations in the surface. Work has also been done to increase the effectiveness and accuracy of random site selection. This led to a reduction in cluster processing that can be found when uniform random number generators are exclusively used throughout the model.

1.1 Need for Growth Simulation Tool

Growth simulation work is critical in device development. A tool is needed that can analyze multiple growth experiments in both monatomic and alloy deposition scenarios. Alloy material sets have become of extreme importance for current device research and development. Nanoscale deposition techniques used in the formation of

Self Aligned Quantum Dots (SAQDs) is an area of considerable interest. A simulation tool is required that can accurately depict the formation of island structures during alloy growth. This study will look at the feasibility of forming SAQDs during deposition as well as how growth parameters such as mixture concentration and temperature affect island growth and topology.

1.2 Common Modeling Techniques

Two major groups of simulations are currently being developed. Each technique has its strengths and weakness. In this section molecular dynamics and statistical processing will be compared to explain there similarities and limitations, as well as showing why Kinetic Monte Carlo was implemented for this model.

1.2.1 Molecular Dynamics

Molecular Dynamics is a numerical solution technique for the time dependent equations of motion for a molecular system. Molecular dynamics requires the coupling of multiple closed form equations that can accurately describe atom-to-atom interactions. These equations will be from several fields such as quantum and Newtonian physics, thermodynamics, and chemistry. The coupling of the equations is a very time intensive process and usually requires a very time intensive calculations. The models created with molecular dynamics usually have a high correlation with the experimental results yet some experimental systems are too complex to describe in a single closed form solution model. Molecular dynamics (MD) can also have the ability to reverse the results of an

experiment or simulation output and determine starting conditions and parameters. The major downfall of MD is the complexity and computation time of simulations.

The complexity of the models can often prevent multiple parameters being evaluated at once. MD models will often analyze a single feature of surface morphology and will only be valid for the specific application in which it was designed for. Changing portions of the experiment can require extensive changes to the model. Processing time for MD simulations is often much longer than there statistical counterparts. This is due the large amount of computations that must be calculated. This leads to the second type of simulations that will be considered in this dissertation, statistical simulations

1.2.2 Statistical Simulations

Statistical simulations differ from MD in many ways. These models are derived by the statistics and probabilities of surface morphologies. This allows for a dramatic simplification in the governing model equations. With this simplification comes some limitations. Statistical models usually can not describe surface reactions with the detail that is found in MD. This is due to the nature of the governing equations. In MD the governing equations were closed form solution that would describe atomic movement from start to finish. Statistical models calculate the probabilities for atomic movement and then randomly select a movement based upon the output of a random number generator (RNG). This simplification allows for the introduction of error, since the selections are based upon probabilities and RNG. The error introduced is usually acceptable if overall surface features is what is desired from the simulation, not individual atomic movement.

Another disadvantage of statistical simulations is that they are not reversible. The model can take a set of results and reverse them back to starting conditions. This is also due to the nature of not having closed form governing equations. This can often be worked around or possibly not of interest depending on the application. Although statistical models cannot reverse a process they can perform a simulation multiple times, while changing parameters slightly and determine a possible starting parameters based upon the output that has the highest correlation. Which leads to the advantages of statistical processing.

Statistical processing often has a dramatically lower computation time than MD. This allows for multiple simulations to be calculated in the time one MD simulation is completed. This can often be a huge advantage and excellent reason to pursue this simulation methodology. Another advantage of statistical processing is the flexibility and ability of adding multiple parameters and being able to change readily change them. This is due to the fact that all of governing equations do not have to be coupled together, yet only the probabilities for an event occurring need to be determined. This allows for multiple parameters to be evaluated in a single model with a considerable reduction in complexity from MD.

For the purposes of this research a statistical processing technique of Kinetic Monte Carlo was selected which allowed quick simulations that could couple multiple growth parameters in a single model and allow for changes in the experiment.

1.2.3 Kinetic Monte Carlo

Kinetic Monte Carlo (KMC) is a statistical process used in modeling that is from the conventional Monte Carlo (MC) family. Monte Carlo processing is a popular

modeling technique that takes its name from the town of Monte Carlo in Morocco where which made determining probabilities famous via gambling. KMC is an extension of MC that incorporates time. KMC will use Poisson process to help determine time probability densities that will evaluate the time associated with the actual simulation.

A second type of MC simulations are referred to density based Monte Carlo. These simulations use probability density functions to help increase processing time during simulation. This extension can cause an increase in error, but can be managed if implemented properly.

1.3 Dissertation Contributions

The research done for this dissertation has led to the development of a Kinetic Monte Carlo (KMC) software package that accounts for the surface reactions during growth. These affects have been modeled using growth equation developed form kinetic gas theory as well as surface reaction theory. The surface reactions have been coupled into the appropriate equation to give a final set of 6 governing equations; five for growth process and one for time. The research performed has increased the accuracy and flexibility of KMC simulations by adding and changing various portions.

One advantage of the developed KMC model is its ability to accept alloy deposition as well as homogenous deposition. This has increased the flexibility of the model dramatically. Alloy deposition is used in many applications in industry such as optics where Al is used to increase band gap size of devices. Upon including alloys the model will determine the film strain and transition the growth regime accordingly. The work presented in this study will show the results from SiGe/Si and how the transition to

island growth occurs as well as the ability to address the island changes based upon growth parameters.

The model was also created to allow for the incorporation of various crystal orientations. This allows for the user to see how changing the initial substrate orientation will change the overall surface morphology. This is an extension that will allow great insight into various aspects of growth, such as island formation due to strain relaxation. This will also allow for a change in material sets, by defining a new lattice structure in the surface library.

Work was also performed in the random processing of the KMC simulation. Pseudo RNG such as Sobol RNG were evaluated and implemented to reduce the cluster processing that can occur when uniform RNG are used. Sobol RNG has a memory, which helps create a uniform selection of surface sites to evaluate. Chapter 3 will show how clustering can be found if a uniform RNG is used in determining which site to evaluate on the surface.

One final advancement of the model was to include the ability to model surfaces that are non planar. This allows for nontraditional semiconductor systems such as MEMS and NEMS to be modeled as well as traditional systems such as via deposition. The ability to evaluate how via or step edge could the overall surface or device is becoming more important as the feature and device sizes shrink.

CHAPTER 2

GROWTH KINETICS AND THEORY

2.1 Introduction

The fabrication of semiconductor devices and structures can be accomplished using various deposition techniques; the following kinetic theory will be based upon the Molecular Beam Expitay (MBE) processing technology. MBE allows for a high degree of control during the deposition process, which allows for a more precise growth with fewer defects. Epitaxil growth is a process where impinging molecules that are deposited on a crystalline surface are influenced by the initial surface. The deposited material is compatible with the underlying material, and the temperature is high enough to allow fro reorganization of the deposited atom then the deposited film can be crystalline and be aligned with the underlying layer. This high precision comes from the low impurity count caused by the ultra high vacuum (UHV) in which the growth changer can achieve. MBE can deposit in chambers that have a vacuum as low as 1e-12 Torr. Chemical Vapor Deposition can usually achieve vacuums from 1e-5 to 1e-8 Torr while evaporation chambers can achieve levels of 1e-5 Torr. With the increased vacuums comes increased processing time, which greatly limits MBE use in industry.

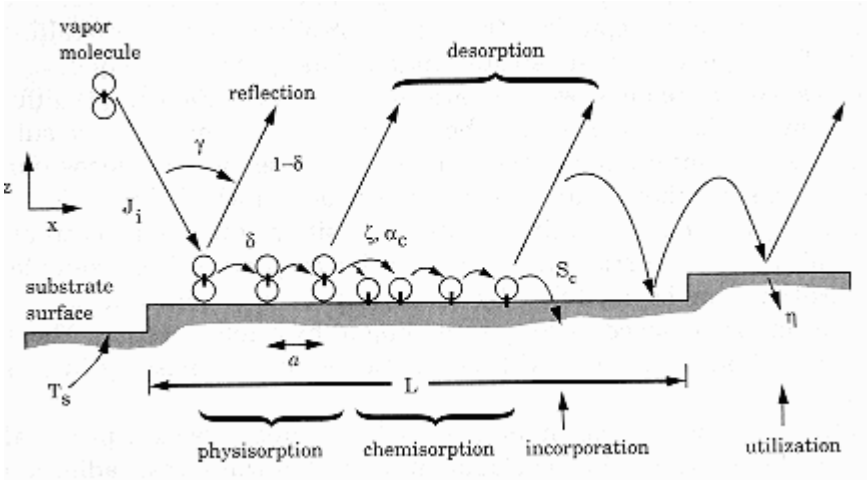


Figure 2.1 Deposition lifecycle for impinging atoms [5].

The chapter discusses various aspects of the surface interactions that take place during deposition. The process of impinging atoms being incorporated to the semiconductor surface happens over a few different steps. Each step has different probabilities for success or failure. Figure 2.1 shows the complete lifecycle of an impinging atom onto the surface of a substrate. The growth control equations used by the Kinetic Monte Carlo Simulation described in the following chapter will be derived from the foundation described in this chapter.

2.2 Growth Stages

To better understand the growth process, it will be broken into four stages (impact, physisorption/chemisorptions and incorporation). Each of these stages will be described in detail to allow a better understanding of the KMC model formulation developed later.

2.2.1 Impact Stage

During the impact stage a vapor molecule will strike the deposition surface. We will consider one of two possible outcomes for this stage, either the atom will stick on the surface and move to the following stage or will reflect from the surface. The probability for either event is determined based upon the energy of the incoming molecule as well as the cross section of adsorption.

The molecule will reflect from the surface if its energy or the surface energy is too great. This energy is related to growth kinetics through temperature. The temperature of the incoming atom derives its thermal energy. This energy level can then be equated with the atoms kinetic energy. This property can be calculated based upon gas theory using Maxwell's speed distribution. Using this distribution the root mean square velocity of an atom at vacuum can be calculated using Equation 2.1.

$$\upsilon_{rms} = \sqrt{\frac{3k_bT}{m}} \qquad \left[\frac{m}{s}\right] \qquad 2.1$$

Where:

k_b = Boltzmann's constant = 1.3806503E-23 $\left[\frac{m^2 kg}{s^2 K}\right]$

T = Temperature [K]

m = Mass of particle [kg]

The temperature of impinging molecules is if often uncontrollable. This temperature is often set based upon melting points. The temperature of the substrate on

the other hand can often be controlled by use of substrate heaters. Deposition chambers for process such as MBE and Chemical Vapor Deposition (CVD) often have substrate heaters that can be used to alter the surface energy level. Less precise deposition process, such as evaporation, will often not have any such means of altering the surface temperature.

Having a surface with a higher temperature allows for more motion on the surface (i.e. a higher energy level). While this can be advantageous to avoid growth defects, if this level becomes too high it can cause atoms to reflect as well as desorb from the surface during later stages. Proper temperature levels are often determined by using experimental data or other growth modeling software. Once the atom has attached to the surface the impact stage is complete and the Phsisoroption/Chemisorption stage begins.

2.2.2 Physisorption/Chemisorption Stage

When the impinging molecule strikes the surface there are two main adsorption states that normally occur. The first state is the physisorption state. Physisorption occurs when the impinging molecule sticks to the surface due to the weak van der wall forces. This weak form of adhesion to the surface may not last. Figure 2.1 shows the trapping probability (δ), which is the ratio of impinging molecules that physissorb to the molecules that are reflected from the surface. The ability of the surface to accept molecules for physisorption is dependent on the ability of the surface to absorb the kinetic energy of the impinging molecules and thereby making the energy of the van der wall forces greater than that of the remaining energy of the impinging molecule.

There is a residence time parameter that can be calculated to help determine how long a molecule might stay in the physisorption state before either desorbing or incorporating into the surface through the next adsorption state, Equation 2.2 shows the residence time equation that was determined trough experimental data obtained by Ruediger Held[4].

$$\tau = \tau_0 e^{\frac{-E_0}{kT}} \quad [s] \qquad 2.2$$

Where:

τ_o = pre exponential factor

E_o = Activation Energy in eV

The only parameter that may be controlled on the surface of the substrate during growth with the MBE system is the temperature. This is how the activation energy as well as residence time are controlled. A high energy level at the surface corresponds to a high temperature of the substrate. As temperature of the surface increases, this also means that the average random kinetic energy of the surface increases. This increase in kinetic energy can thereby be transferred to atoms in the physisorption state. The bonding force in the physisorption state is weak, therefore if too much energy is transferred to the physisorbed atom its kinetic energy can increase higher than the bonding force and thereby desorb from the surface.

If a molecule in the physisorption state becomes chemically bonded to the surface, it is then refereed to as chemisorption. Figure 2.2 shows a graph of energy versus distance from surface. This figure shows the different states that can occur during

deposition of a material onto a substrate.

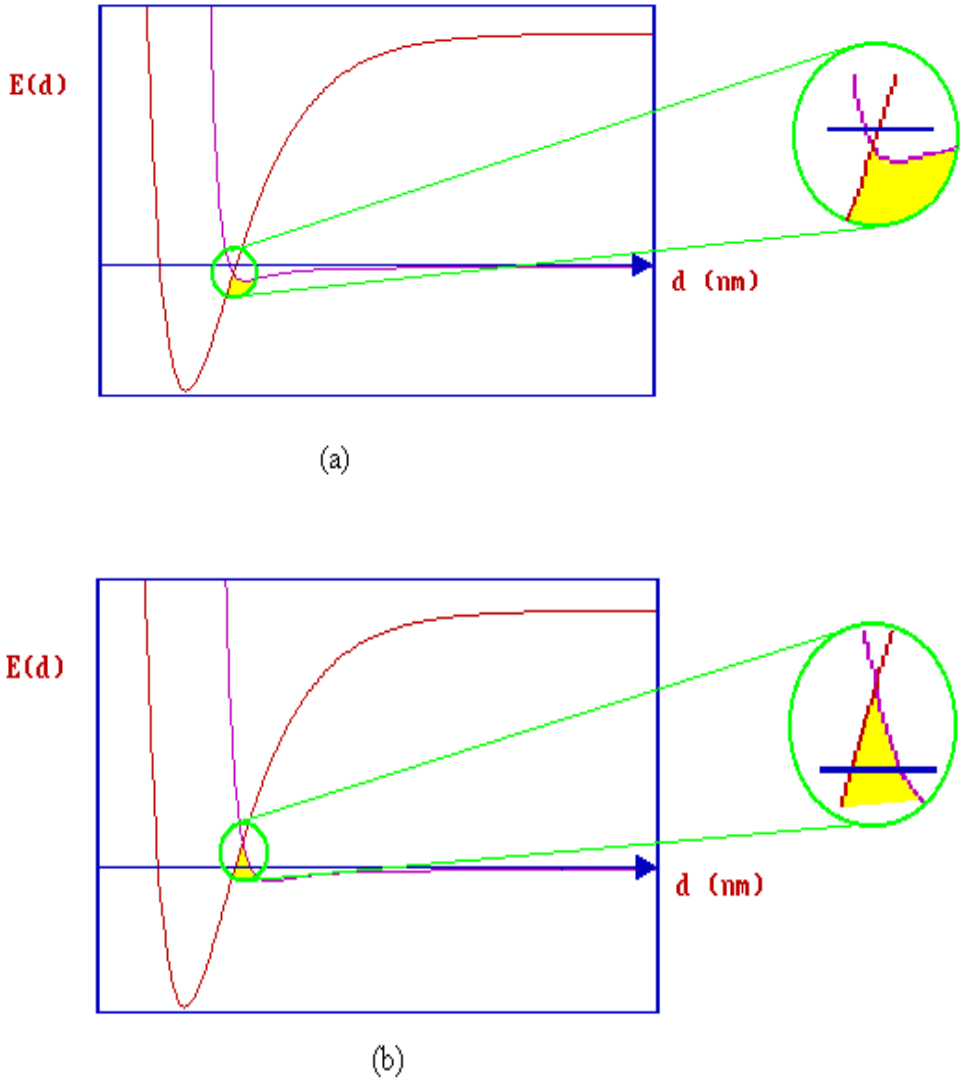


Figure 2.2 Chemisorption curve (a)non-activated (b) activated

The deep wells in Figure 2.2 show the chemisorption state while the smaller well show

the physisorption state (the deeper the well, the stronger the bond). As the molecule approaches the surface its energy will decrease until it reaches the physisorption state. Once physisorption has been reached the atomic lattice structure will need to stretch to allow for chemisorption to occur. For this reasons their will be an increase in energy required allowing for chemisorption. Once the bonds of the neighboring atoms have stretched there will be a sharp drop off of the energy to enter the chemisorbed state. The level of this increase in energy will determine the rate at which chemisorption will occur at the surface of the substrate. In Figure 2.2(a) the zoomed portion shows how the energy required to enter the chemisorption state is lower than the zero level energy of the surrounding area. This type of energy diagram shows a nonactivated chemisorption state. Figure 2.2(b) shows the activation energy to be above the zero level energy of the surrounding area and therefore is called an activated chemisorption state. If a reaction is found to have a nonactivated chemisorption state, incorporation of the molecules to the surface should occur quickly since there is no energy required to go from physisorption to chemisorption.

2.2.3 Incorporation Stage

Once chemisorption and physisorption states can been understood and mathematically modeled the sticking coefficient can be determined for the surface. The sticking coefficient is a ratio of the impinging flux to the number of molecules that incorporate into the surface. Once a molecule has reached incorporation it will not be deorbed. At this stage the molecule has actually become the new growth surface and a second impinging molecules could be deposited as a neighbor. Equation 2.3 shows the relationship for the sticking coefficient[5].

$$S_c = \frac{R_r}{J_i} \qquad 2.3$$

R_r = Chemisorption Rate

J_i = Impinging Flux of Molecules

If the temperature is kept in the proper window described above then the chemisorption can be assumed to only occur in one direction (i.e. not atoms desorb from the chemisorption state). If we assume that the impinging flux is small then the sticking coefficient is approximately equal to chemisorption reaction probability (ζ). This basic assumption comes from the fact that all molecules that undergo chemisorption will be incorporated into the surface. These assumptions lead to the chemisorption rate equation shown below[31].

$$R_r = J_i\left[\frac{\delta}{1+\frac{\upsilon_{0_d}}{\upsilon_{0_r}} e^{\frac{-(E_r-E_d)}{RT_s}}}\right] = \zeta J_i \qquad \left[\frac{1}{s}\right] \qquad 2.4$$

Where:

δ = Trapping Probability

T_s = Temperature at Substrate

υ_{o_k} = Frequency Factor, Pre-Exponential

E_k = Activation Energy

2.3 Growth Process Rates

With the foundations laid in Section 2.1 focus can now be turned to growth process rates. Several possible surface morphologies can occur during a given deposition, based upon the growth parameters (Temperature, Pressure, Material Set, etc.). Growth methods can range from common vertical growth to nucleation layer that lead to horizontal selective growth.

The deposition of a material on a substrate through MBE, or any process, is a very complex problem to study analytically. One common approach is to use growth isotherms that help describe the reactions on the surface of the substrate when an impinging species is present. Two different isotherms will be described here[29].

2.3.1 Growth Isotherms

The first growth isotherm is the Langmuir isotherm which has three assumptions that limit its effectiveness for describing growth on the surface of the substrate[29]. The first assumption is that the adsorption of the impingement flux will not be greater than one monolayer. This is not necessarily the case. During deposition the energy of the surface along with the adsorption states could allow for incorporation of the molecule below one monolayer. The time required to incorporate the molecule into the surface will greatly depend on the time it is located at surface and the probability of desorption from the surface of the substrate. Using the Langmuir description a molecule will only have time to adsorb through one monolayer before the next layer is being deposited on top.

Another assumption of the Langmuir isotherm is that all sites are equivalent and

the surface is uniform. This is one of the most limiting assumptions. During deposition the material may not grow as a perfectly smooth surface on an atomic scale. One common growth practice is a nucleation of the substrate to help induce growth. Nucleation layers act as a seed layer for the rest of the material to grow from. One such depostion process is called a selective lateral growth technique. In this process a small seed layer of molecules are deposited on the substrate, which form the small islands from which the impinging molecules will then grow laterally across the substrate. Figure 2.3 shows a graphical representation of the lateral growth.

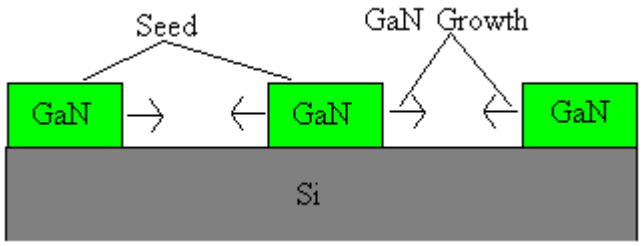


Figure 2.3 Graphical representation of lateral growth of GaN on Si substrate

This assumption also requires all sites to be equivalent which will not be the case if grown laterally. With lateral growth some sites will be occupied while others will be unoccupied.

One final assumption of the Langmuir isotherm is that the probability for a molecule to adsorb at a given location will have no dependence of the vacancies of neighboring adsorption sites. This will also not be the case once deposition has started. As the surface morphology changes during deposition the probability for adsorption at any site will change continuously. This assumption is one that has to be incorporated due the random kinetics involved in the growth. The adsorption statistics change

continuously during growth and therefore do not allow for a mathematical representation that can accurately predict the adsorption at sites on the substrate surface. For this reason a superposition approach is taken that allows for each site to act independent of all others and thereby decoupling the mathematical model for each adsorption site at the substrate surface.

To use the Langmuir isotherm the rates for adsorption and for desorption must be determined. The values are usually found by experimental means. The rate constants will follow the Arrhenius equation show in Equation 2.5[3].

$$k_{a,d} = \upsilon_{0_k} e^{\frac{-E_k}{RT}} \quad \left[\frac{1}{s}\right] \quad 2.5$$

Where:

$k_{a,d}$ = Rate of adsorption /desorption

υ_{o_k} = Frequency factor, pre-exponential

E_k = Reaction activation energy

The second isotherm is the BET isotherm. This isotherm is named after the three men who developed it (Stephen Brunauer, Paul Emmett, and Edward Teller). The BET isotherm is one of the most complex isotherms for multiplayer adsorption. The BET isotherm takes into account the material surface changing on which adsorption occurs, therefore allow for more accurate model of multiplayer deposition processes. There are many curve fitting parameters needed to able to properly apply this isotherm for a growth model. These parameters are usually extracted from empirical data obtained through

experiments. Due to the empirical values needed for curve fitting parameters a different approach was taken for the development of the adsorption rate equation used in this study.

2.3.2 Adsorption Rate

For the development of the adsorption rate a derivation from the kinetic theory of gases is used. The adsorption rate can be determined by comparing the arrival rate of impinging molecules with the molecules that either deflect from the surface or desorb from the surface. This sticking coefficient (S_c) is the gives the percentage of molecules that strike the surface that enter the chemisorption stage. Therefore by combing arrival rate with the sticking coefficient an adsorption model can be defined.

The arrival rate of molecules from a gas can be expressed in the form shown in equation 2.6.

$$r = \frac{1}{4} n v \qquad 2.6$$

Where:

n = Gas number density per unit volume

v = Average velocity

Equating the kinetic energy of the particle mass m with a root mean square velocity v_{rms} to their thermal energy determined by the absolute temperature T and Boltzmann's constant (k_b) gave equation 2.1. Using the relationship between the two velocities gives equation 4.7

$$v = \left(\frac{8}{3\pi}\right)^{1/2} v_{rms} \qquad 2.7$$

Using the relationship of pressure shown in equation 2.8 as well as equation 2.7, equation 2.6 can be rewritten in the form show in equation 2.9.

$$P = nk_B T \qquad 2.8$$

$$r = \frac{P}{\sqrt{2\pi k_B T m}}. \qquad 2.9$$

As stated earlier in this section combing the arrival rate with the sticking coefficient will give the adsorption rate shown in equation 2.10[21].

$$k_{ads} = S_c \frac{P}{\sqrt{2\pi k_B T m}}. \quad \left[\frac{1}{m^2 s}\right] 2.10$$

2.3.3 Desorption Rate

The desorption and diffusion rate calculation will be based around Arrhenius equation show in Equation 2.5 with nearest neighbor affect included. The desorption rate can be calculated by determining the strength of the molecules bond to the surface along with the energy being transferred to the molecule from the temperature of the surface. Using these characteristics the desorption rate equation will take the activation energy for desorption along with the energy added by bonded neighboring atoms and combine it with the effect of surface temperature. Equation 2.11[21] shows the form of desorption rate developed.

$$k_{de} = \frac{k_b T}{h} \exp\left(-\frac{E_{des,0} + i\Delta E}{k_b T}\right) \quad \left[\frac{1}{s}\right] \quad 2.11$$

Where:

$E_{des,0}$ = Desorption Activation Energy

ΔE = Energy Added by Bonded Neighbor

i = Number of Bonded Neighbors

The above equation uses the Maxwell's distribution since we are dealing with identical but distinguishable particles and the thermal frequency pre-exponential term. The nearest neighbor bonding method is the fundamental basis for desorption rate equation. This model does not account for account for surface anomalies such as step edge potential affects. This assumption will be valid in this model as long at temperature is sufficiently high enough.

2.3.4 Surface Diffusion Rate

The diffusion rate equation has the same form as equation 2.11 and is shown below.

$$k_{di} = \frac{k_b T}{h} \exp\left(-\frac{E_{dif,0} + i\Delta E}{k_b T} \right) \left[\frac{1}{s} \right] \quad 2.12$$

Where:

$E_{dif,0}$ = Diffusion Activation Energy

ΔE = Energy Added by Bonded Neighbor

i = Number of Bonded Neighbors

All of the constants and variable in equation 2.12 are identical to 2.11 except for the diffusion activation energy. During surface diffusion the molecule has enough energy to

break the surface bond, but not enough to desorb from the surface. This allows the atom to move around on the surface to find a lower energy state without thereby reducing defects on the surface. This is an imperative step in expitaxial growth. The surface temperature is the main via for energy exchange to the surface molecule. Often during the end of a deposition procedure an annealing phase will occur where the surface temperature is increased to allow for surface reconstruction through surface diffusion. This allows for surface atoms to rearrange in the lowest achievable energy level for the current conditions, which will often remove surface defects such as dislocations, slips and grain boundaries.

The nearest neighbor bonding method is the fundamental basis for diffusion rate equation. This model does not account for account for surface anomalies such as step edge potential affects. This assumption will be valid in this model as long at temperature is sufficiently high enough and flux rates are large enough to limit long range surface diffusion.

2.4 Growth Regimes

Three distinct growth regimes exist for thin film growth. This section will describe the Frank-van der Merwe (FM), Volmer-Weber (VW) and Stranski-Krastanov (SK) growth processes. The determination of which growth will occur on the surface depends on different energy calculations along with the lattice mismatch that causes strain in the depositing layers.

2.4.1 Frank-van der Merwe (FM)

The FM growth process describe a layer by layer surface morphology. In this process the underlying layer must be complete before the second layer begins developing on top. Figure 2.4 shows a graphical representation of the surface morphology for the FM regime.

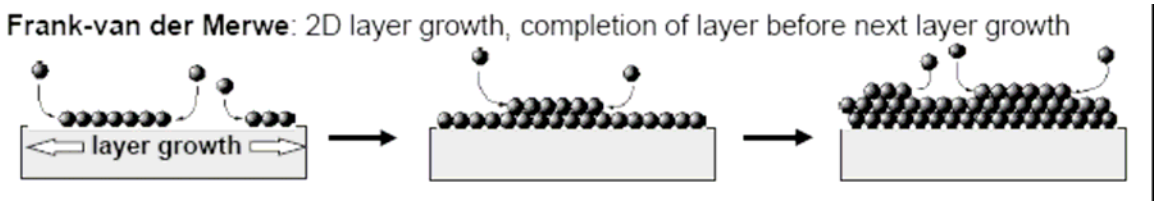


Figure 2.4 Graphical representation of the Frank-van der Merwe growth process

FM growth occurs when the interface energy and film energy have to be lower than the surface energy. Equation 2.13 shows the relationship that must be satisfied for FM growth to occur.

$$\gamma_I + \gamma_F \leq \gamma_S \qquad 2.13$$

Where:

γ_I = Interface Energy

γ_F = Film Energy

γ_S = Surface Energy

Defects at the surface or within the film during growth will cause a change in the interface energy which could cause a change in the growth mode. VM growth will

describe such a case.

2.4.2 Volmer-Weber (VW)

VM growth describes the formation of 3D islands on the surface from nucleation sites that occurring during deposition. This form of deposition is linked to defect sites on the surface such as kink, dislocations or step edges. This causes a change in the film and interface energy. Growth systems such as heteroepitaxy would cause such an increase in the interface energy that could lead to VM growth. Figure 2.5 shows a graphical representation of the VM growth process.

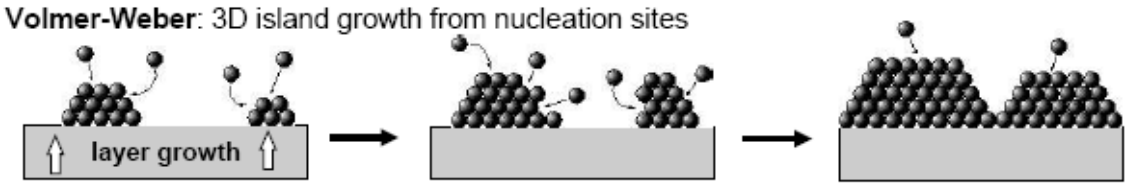


Figure 2.5 Graphical representation of the Volmer-Weber growth process

The condition that must exist for VM growth is described in Equation 2.14

$$\gamma_I + \gamma_F > \gamma_S \qquad 2.14$$

The last growth regime is described below which is a mix of FM and VW modes.

2.4.3 Stranski-Krastanov (SK)

SK growth is a mixed mode surface morphology that starts with thin film layer usually 1-2 ML thick called a wetting layer. This layer follows the FM growth regime and usually a coherently strained thin film. The stress in the film causes a need for equalization of the total energy of the system which causes the VW morphology to begin. The island formations allow for the surface to relax and thereby lower the overall energy of the system. This model is seen in alloy deposition techniques such as SiGe/Si. This is the same type of growth that occurs during the formation of Self Aligned Quantum Dots (SAQD) from SiGe/Si material system. Figure 2.6 shows a graphical representation of the SK growth regime.

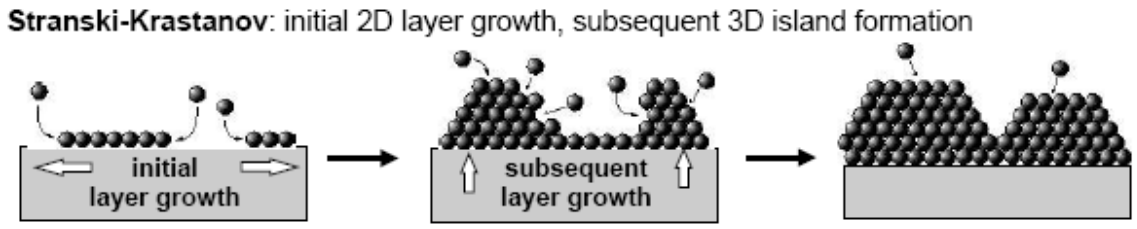


Figure 2.6 Graphical representation of the Stranski-Krastanov growth process

Equation 2.15 shows the relationship that shows when SK growth is mostly likely to occur.

$$\gamma_I + \gamma_F > \gamma_S \qquad 2.15$$

The model developed in this study uses SK growth to describe the SAQD that form during SiGe/Si growth. The formation and morphology of the islands are determined via growth parameters such as temperature and strain and surface energy calculations. Other affects such as Ge segregation and island composition, which have shown to have an affect on island morphology and formation, have not been included in this simulation.

CHAPTER 3

KINETIC MONTE CARLO MODEL SIMULATION THEORY AND DEVELOPMENT

3.1 Introduction

The development of a model that can accurately depict the overall surface of a deposition process is an extremely useful tool in development as well as manufacturing stages. Two main schools of thought are used in development of various growth models. The first is based upon molecular dynamics. These models couple closed form solutions of various fields of physics such as Newtonian, quantum and thermal. The models developed for these simulations can be very complex and require a large amount of computational power. Molecular dynamic models, if developed properly, can have a low error rate which gives outputs that have a high correlation to actual growth results. Usually such models are developed to study only a few of the growth parameters to help minimize the complex nature of the coupled growth equations.

The second school of thought follows a path, which provides results based upon statistical calculations that are of somewhat simpler nature than that of molecular dynamics simulations. These models are often referred to as Monte Carlo (MC) simulations. MC simulations are based around a set of governing equations along with statistical probabilities for different events. The coupling of the equations and

probabilities has shown to give a high correlation between simulated results and physical results.

Although the error produced by MC simulations is higher than that of a molecular dynamic simulation, it still proves useful in understanding and developing different growth phenomena and procedures. The computational requirements of a MC simulation are much less than its molecular dynamic counterpart, thereby allowing for quicker results with less overhead cost.

Within the MC simulation arena time is often neglected or removed from the simulation. Kinetic Monte Carlo (KMC) simulations on the other hand account for time through its governing equations. Simulating time in a random process in which several outcomes are possible, requires a more complex view of the experiment. The model described below takes into account the growth process discussed in Chapter 2 as well as processing time thereby giving a true KMC simulation.

3.2 Model Overview

The KMC simulation is broken into several different procedural steps. The simulation starts with definition and initialization of the growth surface under test (SUT). Once the surface is initialized the simulation begins by randomly selecting a growth site on the SUT via Sobol random number generator (RNG). The probabilities for each of the growth processes occurring at that site are then calculated by using the governing growth equations. One growth process is then selected through the use of a uniform RNG. Upon process selection the site is updated and process count is incremented, thereby allowing

for time calculation. At this point the KMC counter is incremented and the process continues until the surface has been completely analyzed for the current KMC step. At the end of a KMC step physical time step is calculated and the KMC counter is reset and thereby allowing the process to start again for a new KMC cycle. The process will continue until the KMC cycle reaches a user defined value or the physical time reaches a user defined value. Figure 3.1 shows the overall flow chart for the KMC simulation.

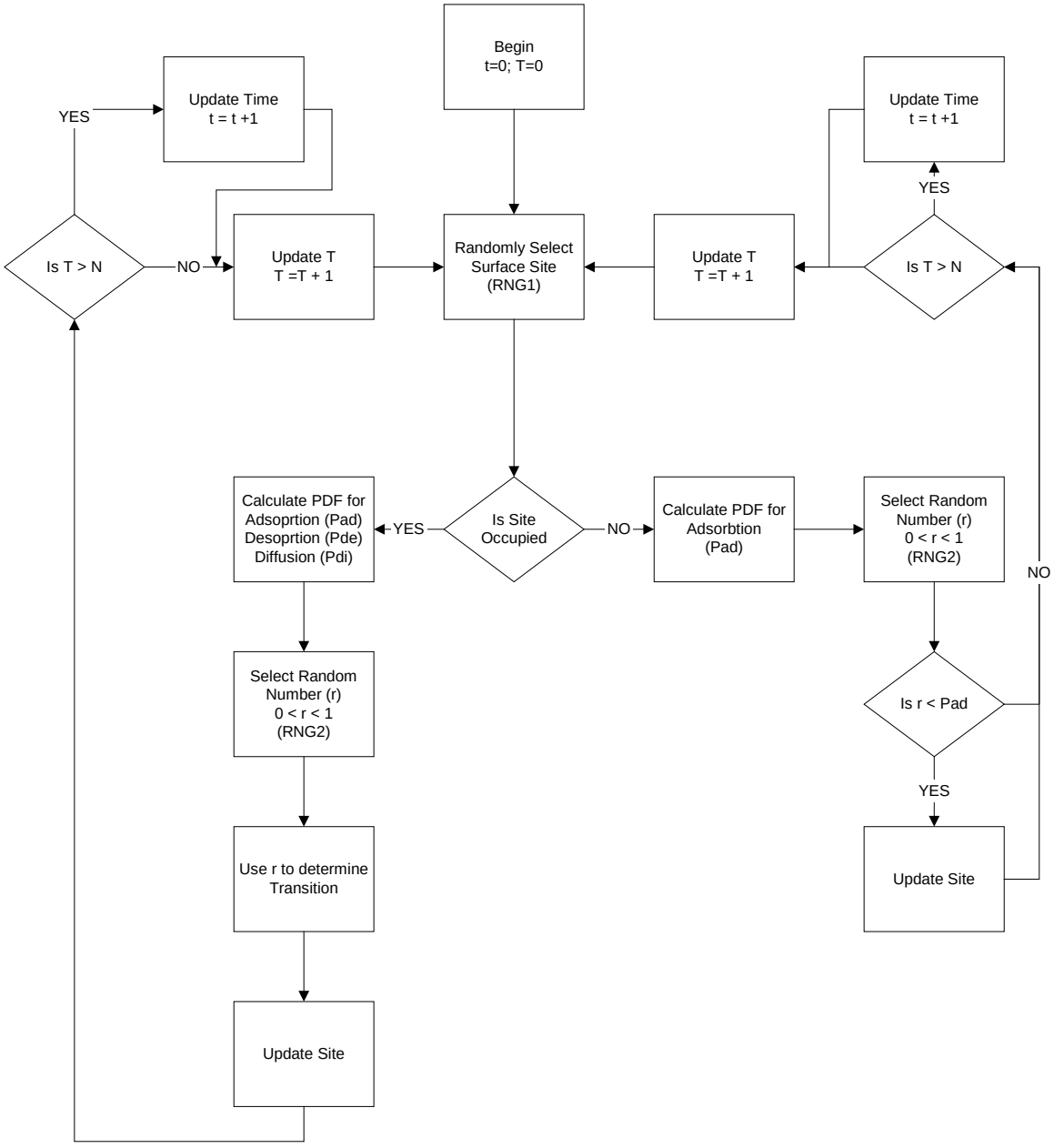


Figure 3.1 Flowchart for KMC simulation

3.3 Surface Initialization and Site Selection

The KMC simulation begins by obtaining user specified parameters. In that parameter set is the size definition of the SUT. The simulation then dynamically creates a three dimensional surface that corresponds to the crystal lattice structure of the SUT. In this simulation the SUT is always defined as silicon, therefore silicon's lattice constant is used for the surface definition (This can be altered for different material sets by changing global variables in the source code). The generated three-dimensional surface is created to show the physical bonding between neighboring atoms. The simulation places gaps on surface planes to allow for the physical representation of silicon's diamond lattice structure. Figure 3.2 shows a visualization of the silicon lattice.

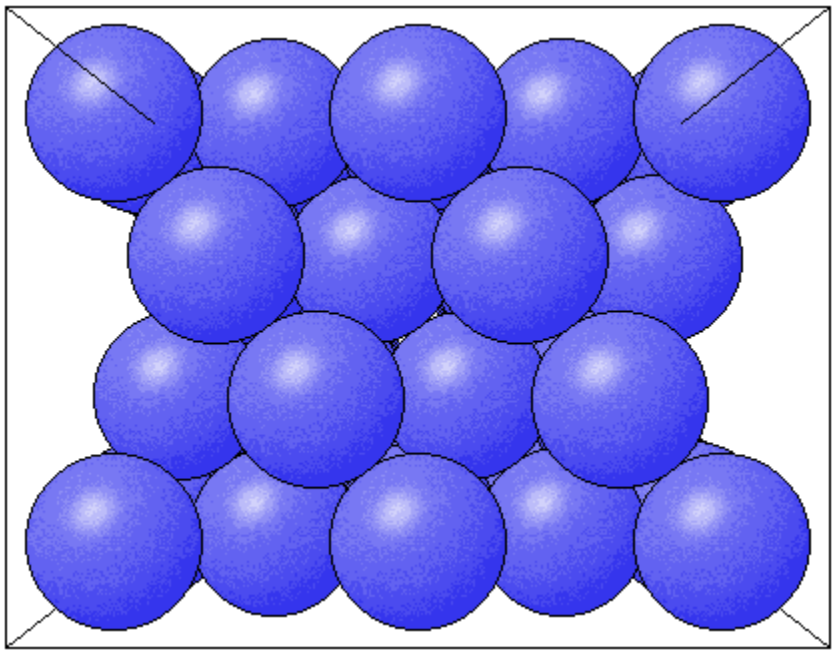
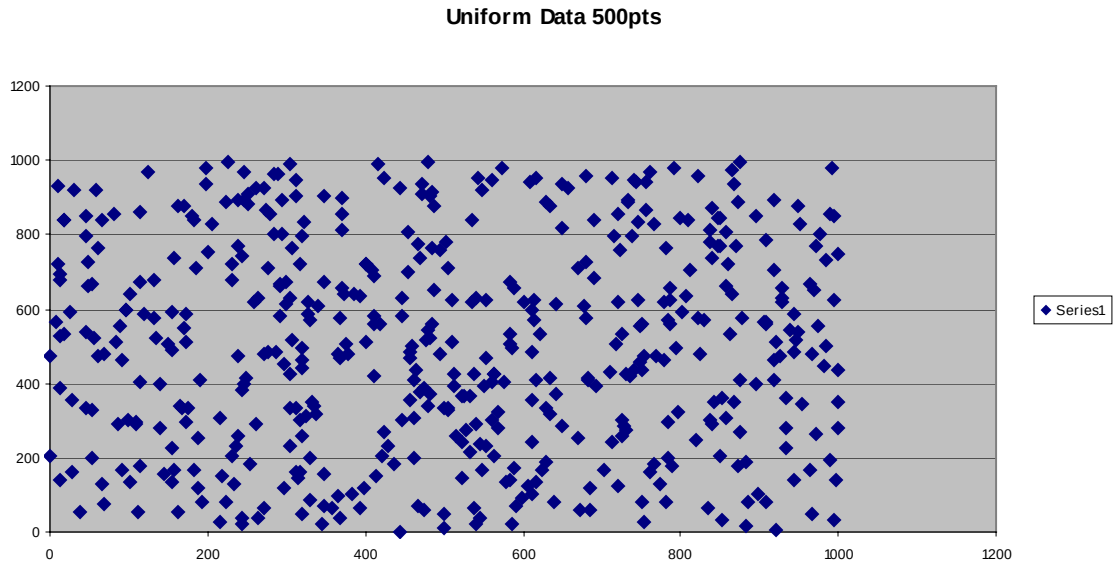


Figure 3.2 Visualization of the silicon crystal lattice structure created for simulation view from <100> direction

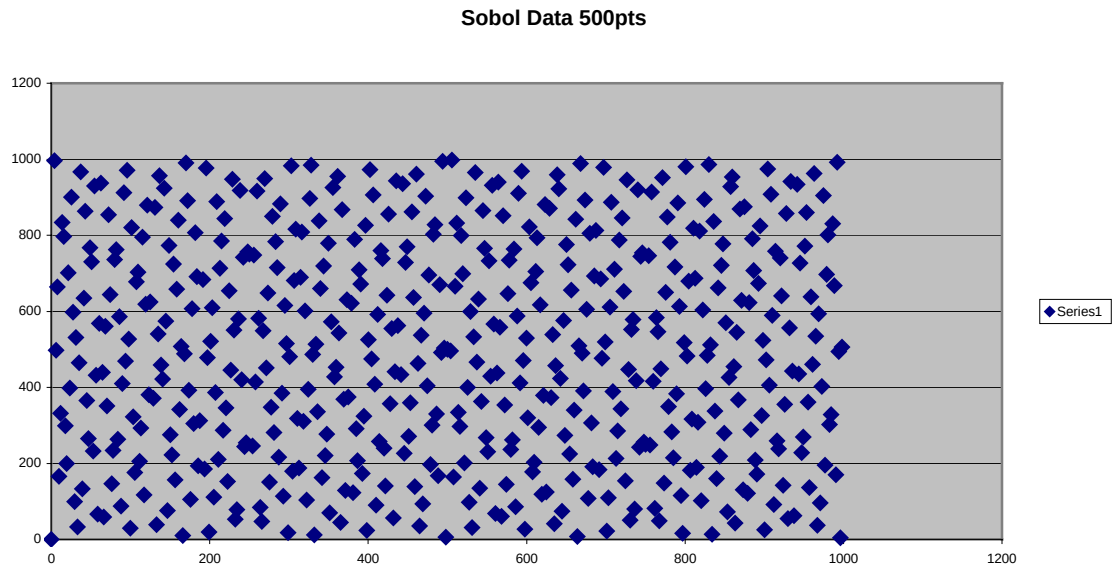
Once the SUT has been created it is initialized by voiding every atomic site in the three dimensional surface. This allows for the program to differentiate from sites that are occupied and sites that are vacant. This portion of the software can require a large amount of memory depending on the size of the SUT. Larger surfaces require more growth sites and therefore require a larger amount of memory.

Once the SUT is initialized the software begins the KMC simulation. The first step in the simulation is to select a valid growth site on the SUT. This site is selected by randomly selecting an x an y coordinate in the SUT. The z coordinate is found by locating the lowest unoccupied site for that coordinate pair. A Sobol RNG was chosen for this portion of the simulation.

Sobol RNG are considered to be a pseudo RNG due to the fact that it has a memory and is not completely random. The idea behind using a Sobol RNG was to allow for a uniform selection of growth sites and to eliminate cluster selections. Figure 3.3 and 3.4 show 500 randomly selected data points generated using a Uniform RNG and a Sobol RNG developed by Burkardt at Florida State Computational Science Department [22].



Graph 3.1 Plot of 500 randomly selected points using a uniform RNG



Graph 3.2 Plot of 500 randomly generated points using a Sobol RNG

The difference in coverage is easy to see and a uniform coverage was desired for site selection, therefore the Sobol RNG is used for this portion of the KMC simulation.

3.4 Growth Processes

The growth process equations developed in Chapter 2 will be the governing equations for the KMC simulation. The rate at which each process can occur will govern which process has the highest probability of occurring for a given set of conditions. Therefore the rate equations will be normalized to give the probability for a selected site to enter a second configuration (i.e. adsorption, desorption or surface diffusion) [21]. Upon each iteration only specific transitions will be analyzed. Figure 3.3 shows all of the possible transitions that can occur once an atom has adsorbed to the surface. The transitions depicted in Figure 3.3 follow those defined by Wolf-Villian MBE simulation techniques[53].

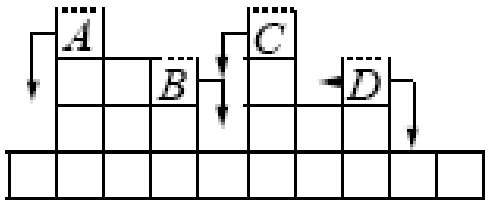


Figure 3.3 Shows the valid transitions that are possible for different adatoms

Each possible transition that can occur in the KMC simulation will be determined via the various equations described below. These equations are evaluated to determine the probability of an event occurring as well as the amount of time that occurred during the event transition. The probability for the different transitions will be determined via the rate equations developed in Chapter 2.

3.4.1 Adsorption

The adsorption rate equation developed in Chapter 2 will be used in the KMC simulation to determine probability and timing for an adsorption event. Equation 3.1 is a reproduction of Equation 2.10. Upon selection of a growth site, if it is determined by the KMC simulation that an atom will adsorb to the site the program will update the site to show it is now occupied.

$$k_{ads} = S_c \sqrt{\frac{1}{2\pi m k_b T}} \qquad 3.1$$

Where:

S_c = Sticking Coefficient

m = Mass of silicon

T = Temperature of Substrate

k_b = Boltzmann's constant

3.4.2 Desorption and Diffusion

The reaction rates calculated in Chapter 2 are the same used in the KMC simulation model. They are reproduced below for convenience, no changes have been made.

$$k_{de} = \frac{k_b T}{h} \exp\left(- \frac{E_{des,0} + i\Delta E}{k_b T} \right) \qquad 3.2$$

$$k_{di} = \frac{k_b T}{h} \exp\left(- \frac{E_{dif,0} + i\Delta E}{k_b T} \right) \qquad 3.3$$

Where:

$E_{dif,0}$ = Diffusion Activation Energy [3.02E-19J]

$E_{des,0}$ = Desorption Activation Energy [2.64E-18J]

ΔE = Energy Added by Bonded Neighbor [7.59E-20J]

i = Number of Bonded Neighbors

After a site has been selected the KMC simulation will determine the total number of bonded neighbors and calculate the diffusion and desorption rates. If the selected site will undergo desorption then the program will remove the atom from the SUT and reinitialize the spot to vacant. The diffusion and desorption rate equations only account for nearest neighbor affects. These neighbors must be located on the same monolayer as the current adatom to have an affect. This model will ignore the potential barriers associated with diffusing up a layer, since an upward diffusion is not allowed in this simulation per the Wolf-Villian model used as a foundation. This assumption is valid for cases where the temperature imparted to the system allows for the bonding energy of atom at step edge is sufficiently large enough to prevent the jump of diffusing atom to higher level. In this case the step edge potential can be neglected.

Another assumption made by the KMC simulation is that only one atom at a time is evaluated. This removes the possibility of dilmer surface diffusion. This should only introduce a small margin of error, since only overall surface features are being examined and not individual atomic movements.

If a selected site is chosen to undergo surface diffusion then the program will reinitialize the current site to vacant as well. The program then enters a subroutine to determine which site the diffused atom will transition too. The routine will determine how many neighbors, if any, are bonded to the site. The routine determines which of the neighbor sites are valid for surface diffusion. Once the valid neighbor diffusion sites have been determined a value provided by a uniform RNG is used to determine which of the neighboring sites the atom will transition too.

3.4.3 Strain Driven Island Formation

During alloy deposition a few adjustments are made to the model to incorporate a mixed specie material system. Once a site has been selected on the surface to undergo analysis via the Sobol RNG the KMC simulation must determine which atomic specie is impinging on the surface. This is determined through a uniform RNG and the atomic mixture variable x ($[Si_{1-x}Ge_x]$). For example using SiGe material set if the random number is less than x, Si will be selected otherwise Ge will be selected. Once the appropriate atomic material has been selected the possible growth process for that site will be determined. The same growth process and transitions that were described above

will be examined.

Alloy deposition causes increased complexity in surface morphology. The mixed alloy material causes a strain to become present in the surface film. This strain causes the growth regime to change either to VM or SK growth. This effect is caused by the increased energy departed to the system due to the strain introduced by the alloy material. The overall energy in the system is always trying to maintain the lowest possible level for the current configuration. This leads to the 3D island formation found at low temperature SiGe growth.

In SiGe alloy systems SK growth occurs due to the competition between anisotropy strain energy and anisotropy surface energy. During deposition island formations occur as a way to relax the strain energy created during deposition. As strain energy is relaxed through island formation, surface energy is increased due to the increased surface area of the system. These two processes conflict with each other and cause a balancing act during the growth morphology.

In the model used for alloy growth the total energy of the system is comprised of strain and surface energy together. The model minimizes the overall system energy during growth by evaluating both strain and surface energy contributions. For instance, the island formed on the surface will continue to grow in height until the reduction in strain energy caused by increased height is negated by the increase in surface energy caused by the increase number of dangling bonds.

Alloy composition and temperature and the modeling equations will account for both affect both strain and surface energies. As temperature increases, the strain energy

lowers due to the thermal expansion caused in the lattice. This expansion helps relax the strain caused by the alloy and thereby decreases the overall film strain. Surface energy has the opposite affect due to the increased thermal energy departed to the surface as surface temperature increases.

The affects of temperature and alloy composition will cause changes in the island formation shapes. At lower temperatures the island formations will have smaller diameters but taller heights, most commonly denoted as "huts". This is caused by the decrease in the antistrophic surface energy and increase in antistrophic strain energy at lower temperatures. The lower temperature requires a taller island for relaxation and the decreased surface energy allows for a higher surface area.

As temperature increases the island huts will transform to a shorter wider island commonly referred to as "domes". The height is lowered due to the decrease in strain energy caused by increased temperature. The widening occurs due to the increased surface energy caused by increased surface temperature.

The shape transitions are also influenced by interactions of neighboring islands. Coalescence of islands occurs during growth as the island widths begin to increase. This phenomena is not directly found in the simulation equations, rather it is an affect that occurs naturally through the competition of surface and strain energy.

The final surface of SiGe/Si system shows a bimodal distribution of islands, which are randomly distributed on the surface. The distribution is based upon the coalescence that occurs during growth along with the minimization of total energy of the film during growth. The surface transitions are governed by the mass transport of

adatoms via a response from the surface gradients in the chemical potential. Equation 3.4 shows a simple form for the chemical potential.

$$\mu = \gamma + \omega + w \qquad 3.4$$

Where:

μ: Chemical potential

γ: Surface energy contribution term

ω: Strain energy contribution term

w: Surface interaction between film and substrate term

As the film becomes thick the surface interaction between film and substrate can be neglected and the chemical potential will be determined directly from surface and strain energy terms. Surface interaction term (w) helps describe the transition between wetting and island growth. The approach taken in this model was to implement a boundary condition solution that shows while the surface is below the critical thickness for coherently strained thing films the wetting layer contribution is accounted for and thickness larger that the critical thickness this term is ignored. The gradients in the chemical potential caused by the anisotropy of the crystal will govern the mass transport properties such as surface diffusion.

The monotonically decreasing chemical potential shows that any distribution of islands will coarsen with larger islands growing at the expense of smaller islands. This is one of the driving forces behind the coalescence of surface islands can be seen in

experimental results.

3.4.3.1 Surface Energy Calculations

The surface energy describes the ability of the surface to accommodate incoming adatoms. Higher surface energies correlate to easier surface adhesion while lower surface energies are associated with more difficult adhesion. The surface energy follows the same basic laws of energy states found in nature, in that the lowest energy state for the given configuration is the most desirable. This one assumption is the key to surface morphology during coherently strained films found in SK and VW growth regime. The KMC simulation will correlate the surface energy directly with the number of dangling bonds available on the surface. As the number of dangling bonds increases, the surface is more susceptible to surface adhesion. When the number of dangling bonds decreases, the number of bonding sites decreases and therefore the surface energy also decreases.

The KMC model also accounts for temperature affects in surface energy calculations. As the surface temperature increases, thermal energy is departed to the surface and increases the overall surface energy. Equation 3.4 shows the equation that the KMC simulation uses to determine a relationship for surface energy. The equation incorporates the number of available bonding sites with the bonding energy of a surface site along with an exponential term that accounts for temperature affects.

$$\gamma = nE_{bond}e^{-\frac{E_{bond}}{k_bT}} \qquad 3.4$$

Where:

n = Number of dangling bonds on surface

E_{bond} = Energy of dangling bond on surface [J]

T = Temperature of Surface [K]

k_b = Boltzmann's constant

Equation 3.4 will be used along with the strain energy equation derived in the following section to determine the total energy at any given configuration.

3.4.3.2 Strain Energy Calculations

The determination of strain energy is developed by the use of Hooke's law shown in Equation 3.5 below.

$$\sigma_{ij} = \lambda_{ijkl}\varepsilon_{kl} \qquad 3.5$$

Where:

σ_{ij} = Stress Tensor

λ_{ijkl} = Stiffness Tensor (or Elastic Moduli)

ε_{kl} = Strain Tensor

Equation 3.5 can be expanded into the anisotropic matrix form shown below.

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{zz}\\ \sigma_{yz}\\ \sigma_{zx}\\ \sigma_{xy}\end{bmatrix}=\begin{bmatrix}C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16}\\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26}\\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36}\\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46}\\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56}\\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66}\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{zz}\\ \varepsilon_{yz}\\ \varepsilon_{zx}\\ \varepsilon_{xy}\end{bmatrix}$$

For cubic structures the above matrix representation can be reduce into Voigt notation which produces a matrix relationship in the following form.

$$\begin{pmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{12}\\ \sigma_{13}\\ \sigma_{23}\end{pmatrix}=\begin{pmatrix}C_{11} & C_{12} & C_{12} & 0 & 0 & 0\\ C_{12} & C_{11} & C_{12} & 0 & 0 & 0\\ C_{12} & C_{12} & C_{11} & 0 & 0 & 0\\ 0 & 0 & 0 & C_{44} & 0 & 0\\ 0 & 0 & 0 & 0 & C_{44} & 0\\ 0 & 0 & 0 & 0 & 0 & C_{44}\end{pmatrix}\begin{pmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ \varepsilon_{12}\\ \varepsilon_{13}\\ \varepsilon_{23}\end{pmatrix}$$

the Voigt notation allows Hooke's Law to be rewritten in the form shown in equation 3.6, which includes Lame' coefficients.

$$\sigma_{ij} = 2\mu\varepsilon_{ij} + \lambda\delta_{ij}\varepsilon_{nn} \qquad 3.6$$

Where:

λ = Lame' coefficient

μ = Lame' coefficient

δ = Deformation

Where the Lame' coefficients are:

$$\lambda = C_{12}$$
$$\mu = C_{44}$$
$$\lambda + 2\mu = C_{11}$$

Equation 3.6 allows for the calculation on the strain in any direction based upon material properties and deformation. The deformation is assumed to be a constant level as defined in Equation 3.7 below.

$$\delta = \frac{a_f - a_s}{a_s} \qquad 3.7$$

Where:

a_f = lattice constant of alloy

a_s = lattice constant of substrate

The amount of strain produced in the film is directly proportional to the concentration of the alloy material. As the concentration of alloy material increases a higher percentage of alloy atoms will occupy growth sites. Each alloy site induces strain caused by its size differentiation from the substrate material. This change in size is the direct cause for strain production in the film, therefore the more growth sites that are

occupied by alloy material the higher the strain in the film. For this reason Equation 3.6 will be multiplied by the alloy concentration percentage to account the appropriate amount of strain. This assumes a linear relationship between the amount of strain in the film and the alloy concentration. Equation 3.6 can be used to calculate the initial amount of strain in the film, but as the height of the film increases relaxation of the film occurs and therefore the strain reduces. Equation 3.8 describes the relaxation of the surface through the formation of islands based upon island thickness [85]. Equation 3.8 is implemented in a discrete fashion in which the strain is calculated at each growth site on the surface. The variables in Equation 3.8 are related to the specifics at the current site under test.

$$w = \frac{C_o}{h^3} e^{\frac{4E_{bond} - nE_{bond}}{k_b T}} x\sigma_{ij} \qquad \left[\frac{kg}{s^2 m}\right] \qquad 3.8$$

Where:

C_o = Material-dependent, stress- and temperature independent constant $\left[0.001 m^3\right]$ [85]

h = Height of test site

x =Alloy concentration

The model being used will assume no voids are produced in the crystal and a coherently strained thin film will be produced. These assumptions should still allow for usable results when analyzing the overall surface structure properties such as island density and surface roughness.

3.4.4 Process Selection

Every iteration of the KMC simulation a site is selected and the process transition rates are calculated with the above equations. Once rates have been calculated they are normalized so one process can be selected for the site to undergo. Equation 3.9 shows the normalization and selection equation used to determine process transition of a growth site.

$$\frac{\sum_{j=0}^{r-1} k_j}{\sum_{j=0}^{R} k_j} \leq rand_1 \leq \frac{\sum_{j=0}^{r} k_j}{\sum_{j=0}^{R} k_j} \qquad 3.9$$

Where:

R = The total number of processes

r = Process

The transition rates are normalized with respect to sum of the transition rates to determine probabilities. The three probabilities will be placed on a continuous scale and a uniform random number will then be selected that will choose a transition process. The process r that makes equation 3.9 true will be the transition that occurs. The site will then be updated with the choice and the Monte Carlo counter will be increased.

In the case of island formation surface transition is determined via energies rather than transition rates. The total system energy will be determined for each possible transition. These energies will be normalized and the transition with the lowest energy will be given the highest probability for occurring. This process will be done identically as that for FM growth via equation 3.9 above.

3.5 Time Incrementing

Once the Monte Carlo counter has reached the number of counts per step, the actual time, t, will be incremented. To keep the Monte Carlo simulation accurate a Poisson Process must be modeled using the following derivation. Let N be a random variable that counts the number of events that occur during a time interval t. The probability that n event will occur is given by the binomial distribution shown below in Equation 3.10

$$P(N=n)=\left(\frac{n_t}{n}\right)(r\delta)^n(1-r\delta)^{n_t-n} \qquad 3.10$$

Where:

n_t = Total number of sample taken in interval

δ = Length of time sampled

r = Probability rate of an event occurring

In the limit of the length of time sampled approaching 0, the binomial distribution shown in Equation 3.10 approaches that of a Poisson distribution shown in Equation 3.11.

$$P(N=n)=\frac{(rt)^n}{n!}e^{-rt} \qquad 3.11$$

Further analysis of Equation 3.11 will lead us to an expression for the probability density of times between event occurring. This relation ship is shown in 3.12 below

$$f(t_e)=re^{-rt} \qquad 3.12$$

Evaluation of 3.12 shows that the mean time between successive events is $1/r$, which then leads to the time incrementing equation shown in 3.13. Noting this, the time interval increment will not be a constant rather will be calculated based upon the transition rates there were calculated in each Monte Carlo step. The evolution in time will follow the following equation.

$$\Delta t=-\frac{\ln rand_2}{\sum_{j=0}^{R}k_j} \qquad [\text{s}] \qquad 3.13$$

Equation 3.13 is based from the Poisson process time for a single event to occur along with the average of all possible transitions that occurred within the Monte

Carlo step. The random variable produced in equation 3.13 is derived from a uniform RNG.

3.6 3D Surface Plotting Software (SPS)

The model results produced by the KMC simulation required a software package that could handle large data sets. The common software packages available would not load the entire data files created as an output, which in turn would not allow for a clear picture of what the model produced. The ability to view the results as well as having a stand alone package that would allow anyone to view the results led to the development of the 3D Surface Plotting Software (SPS).

The 3D SPS viewer allows for rotation of the surface via the mouse as well as zooming by using CNTRL and left mouse button simultaneously. Panning of the surface can be accomplished by using the Shift and left mouse button simultaneously. SPS can also have the ability to take snap shots of the growth surface during the simulation. This is accomplished by calling SPS through the command prompt during simulation. Upon a command prompt call SPS will load a data set take a snap shot of the surface and store as a jpeg using a user defined file name. These snap shots can later be strung together to show an animation of the surface growth. For large data sets, this will limit the speed at which the simulation runs, for upon each call SPS will load the data set, create a jpg and save to the hard drive. SPS has three views to analyze the surface with; bar chart, scatter chart and surface chart. The code for the graphical software is located in Appendix B.

Figure 3.4 and 3.5 show screen shots of the Surface Plotting Software (SPS).

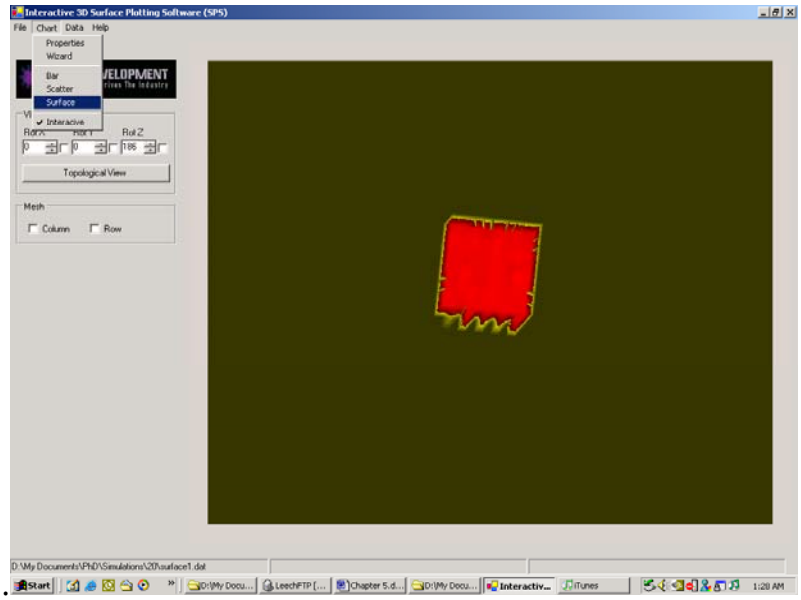


Figure 3.4 Example of SPS for a surface plot of a nano scale rectangular feature

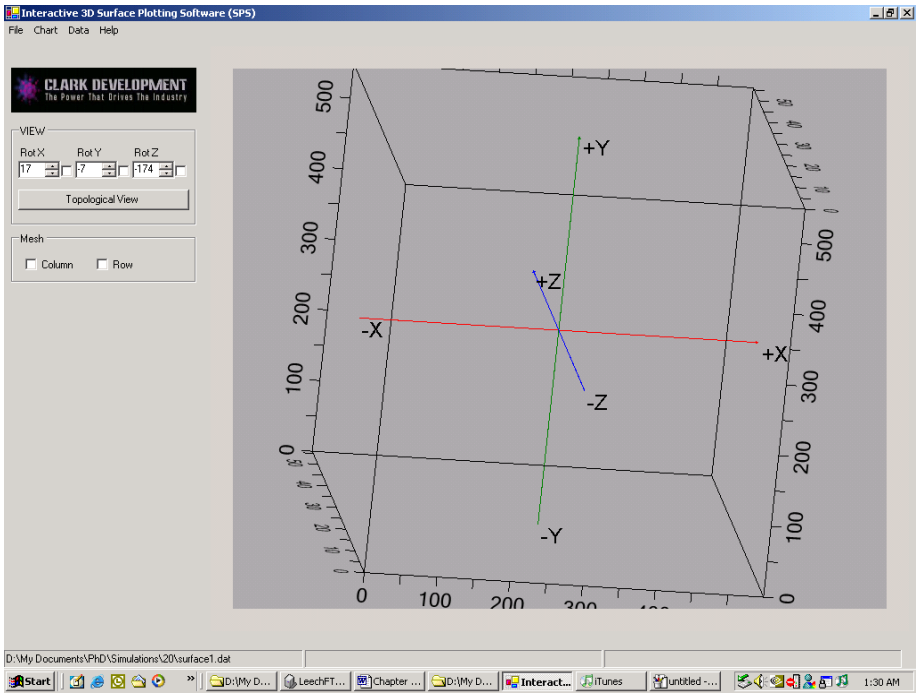


Figure 3.5 Example of SPS in interactive mode, rotating the surface via the mouse

3D SPS also has the ability to rotate the surface automatically as well as show the mesh used to calculate the primitives used to construct the 3D surface. Once a data set is loaded 3D SPS can also take snap shots of the current view and save in any of the conventional image formats(jpg, bmp, gif).

A second application was developed to help with the loading and conversion speed of SPS for large data sets. An application called conversion.exe was developed and can be called by 3D SPS to help eliminate unnecessary data in the surface. This application will measure the growth surface in its textual format and remove unoccupied space around the deposition surface. This helps eliminate a large amount of data that has no affect on the results. This application has increased processing time by 2-3 times.

3.6.1 Interpreting 3D SPS

Upon execution of 3D SPS the software will automatically look for a file called surface.dat and load it accordingly. If there is not surface.dat file then a default image will loaded. Once a data set has been loaded the default parameters will show the data in a topographical view (birds eye of the surface) that is rotating about the z axis. The rotation can be terminated by unchecking the box next to the z coordinate. The default graph will be a surface plot. Changing of colors will represent the surface heights. The color scheme will start with dark green and transition to lighter greens as the height increases. The color scheme will then jump to yellow and then red. Figure 3.6 show the default topographical view of a surface as well as the side view so the color variation can be visualized.

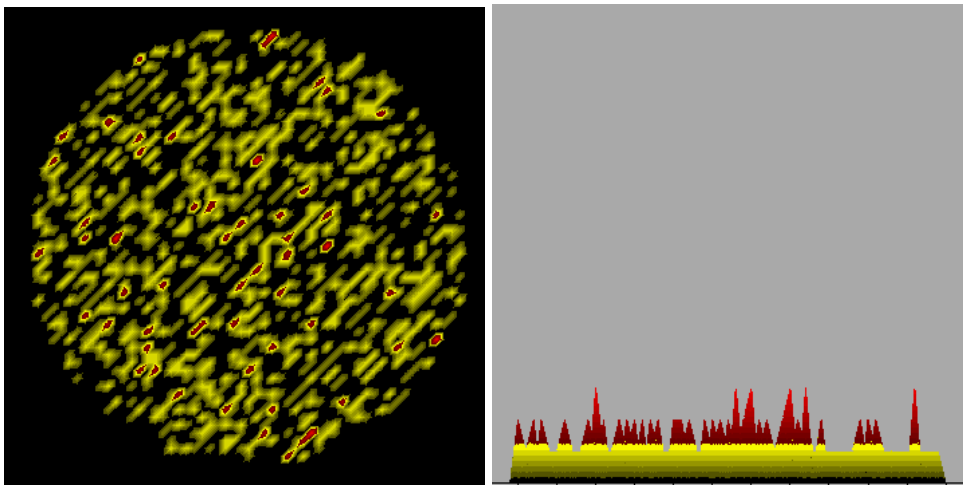


Figure 3.5 Topographical view and side view of data set loaded in 3D SPS

CHAPTER 4

KINETIC MONTE CARLO MODEL VALIDATION

4.1 Introduction

Model validation is an important step in simulation design. In the case of the work presented the validation process will be shown in several steps. The first step will be to show the ability of the current KMC model to produce results of a high correlation with a foundation model from which the overall execution and foundation physics were derived from. These results will be compared with laboratory results to show the ability of the simulations to model actual surface events.

Once the KMC simulation has shown to have agreement with the foundation model, a second set of results will be produced with the KMC tool that incorporates the model additions. The results from the expanded model will be evaluated to show the validity of the model with the added enhancements.

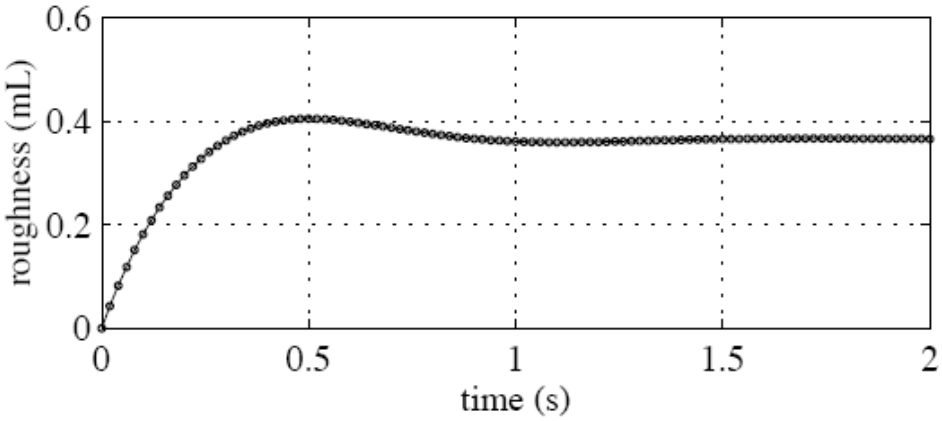
Finally, result from a non planar simulation will be analyzed. These results will show the ability for the KMC tool to handle surface morphology produced through step edge growth. These results will show how the tool can accurate analyze a non planar surface with multi levels of growth.

. To show the validation of the KMC model surface roughness was calculated via the KMC model developed for this study as well as the KMC model which was used as a foundation [21]. Surface roughness is a vital parameter in thin film deposition discussions. The ability to accurately describe the roughness of the surface during the different stages of surface morphology demonstrates the ability of a model to capture the complex nature of deposition process.

Surface roughness evolution during deposition shows the formation and completion of monolayers which leads to the ability for growth rate calculations . These results also show the models ability to describe the multiple reactions that are occurring on the surface throughout the deposition process.

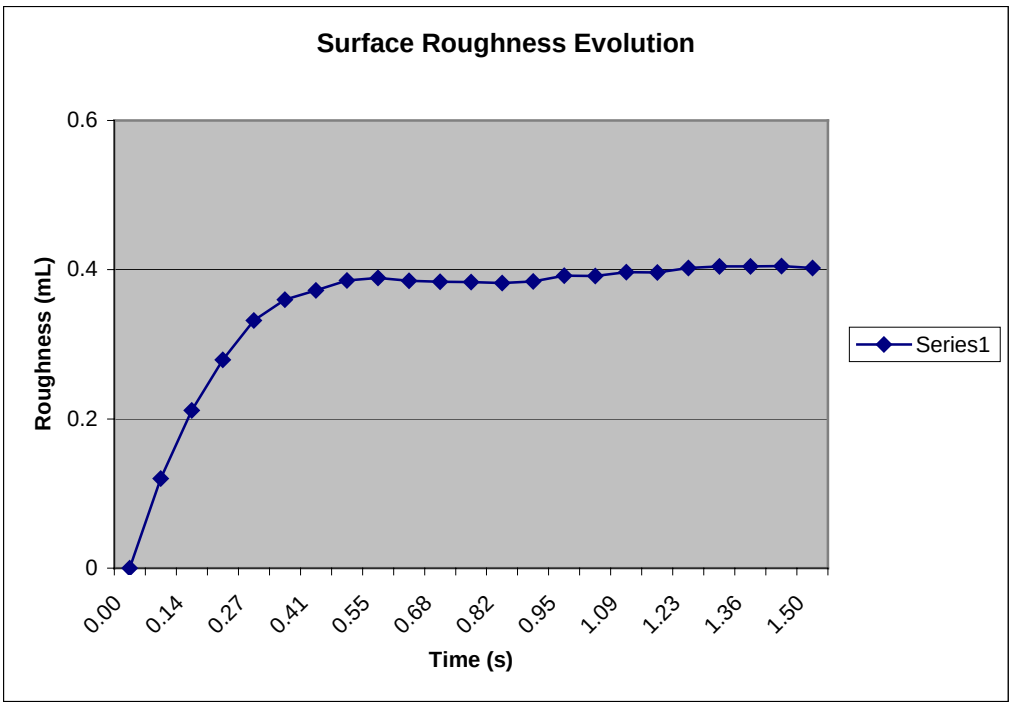
4.2 Foundation Model Results

The foundation model calculated the surface roughness as function of KMC cycles, there results are shown in Graph 4.1. The results show an initial kinetic roughening of surface with oscillations around a steady state roughness level. The oscillations represent the formation and completion of monolayers.



Graph 4.1 Depiction of surface roughness produced from foundation model at an equivalent growth condition of T = 650K and P = 1e-5Torr

Graph 4.1 shows a maximum surface roughness of approximately 0.4mL, which occurs approximately at 0.5 seconds. The surface roughness value then falls to a lower level with a slight oscillation. The oscillation is compressed in Graph 4.1, but if analyzed closely a local maximum approximately around 1.5 seconds can be seen. The surface roughness graph produced in Graph 4.1 describes the surface morphology seen during deposition The surface roughness increases through kinetic roughening and then reaches a steady state roughness value in which it oscillates around as growth continues. Graph 4.2 shows the results obtained via the KMC simulation developed in which no extensions have been added



Graph 4.2 Depiction of surface roughness from KMC simulation tool with foundation assumptions only at T = 650K and P = 1e-5Torr

Graph 4.2 shows an initial kinetic roughening that reaches a maximum value of just under 0.4mL at approximately 0.5 seconds. The oscillation then produced in Graph 4.2 has a maximum value around 1.42 seconds. The general shape shown in Graph 4.2 has the same general shape as that shown in Graph 4.1. There are slight discrepancies from Graph 4.1 and Graph 4.2 such as slight increase in around 1 second in Graph 4.2 that is not see in Graph 4.1. The nature of KMC simulation is that of random processing so the exact replication of a data set would be unlikely due to the random nature of the processing. Therefore the overall shape and general trends must be analyzed to show there correct depiction of surface evolution. Therefore the discrepancies shown from Graph 4.1 to Graph 4.2 are minor and therefore show strong correlation between each

other.

Graph 4.2 shows the surface roughness output produced by the KMC simulation tool developed in this study. The surface roughness was calculated using equation 4.1 below.

$$W = \sqrt{\frac{\sum_{1}^{N} \left(h - \bar{h}\right)^2}{N}} \qquad 4.1$$

Where:

h = Height of selected site

$\bar{h}$ = Averaged height of entire film

N = Number of sites selected for testing

Sites were selected via a uniform RNG. Initially the surface is sampled to find the average height of the film. This average value is saved and then used in the calculation of surface roughness. Once the average height has been determined the site is sampled a second time. Upon selection of a surface site, the height is determined and then the surface roughness is calculated using equation 4.1.

Snap shots during the growth of the simulation for Graph 4.2 were taken to show the overall transformation of the surface during the simulation. Screen captures of the surface during the deposition process is possible via the 3D SPS graphical software developed for this study. The screen shots are taken at the end of every MC cycle. This allows the user to watch the shape formation of the surface at every step. The snapshots

are taken as a topographical view. The color representation is the same as the one explained in section 3.6.1. When analyzing the screen shots from the 3D SPS one must remember that the KMC simulation is an overall model for the surface reactions and not a detailed MD simulation. Therefore the snapshots are only useful for an overall of view of what is happening at the surface, not a detailed description of individual atomic movements.

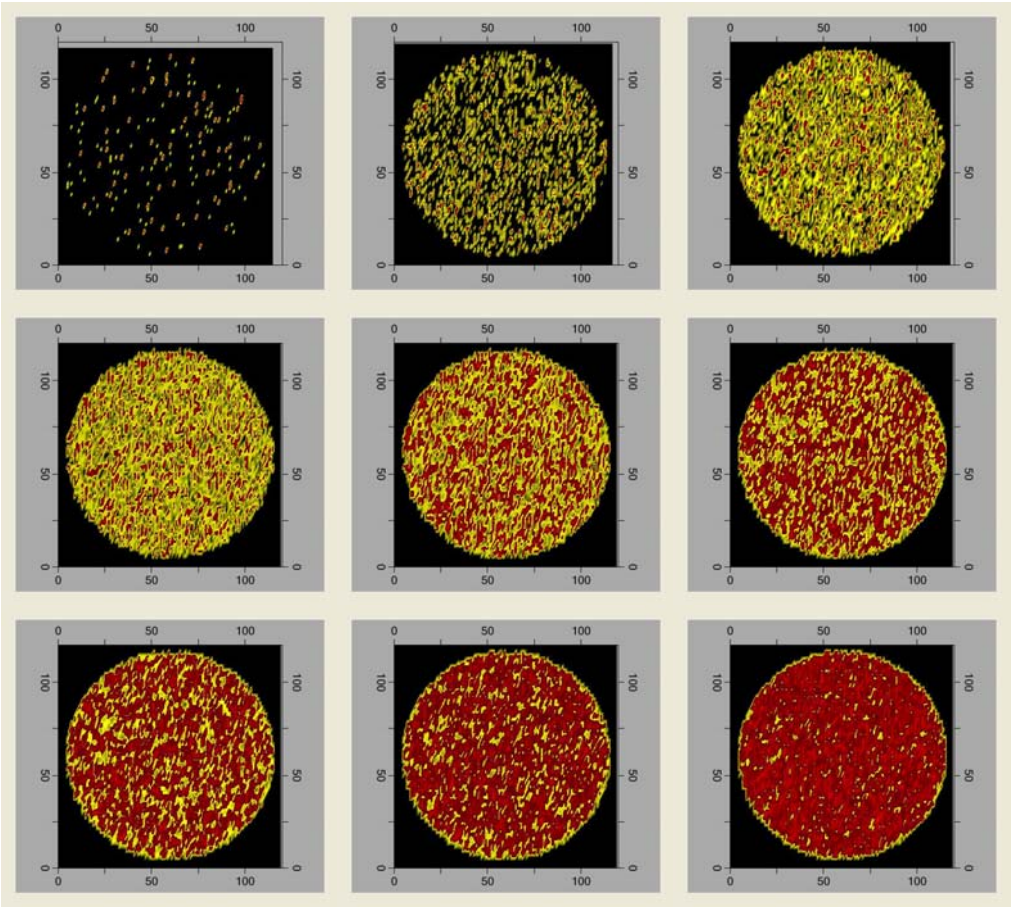
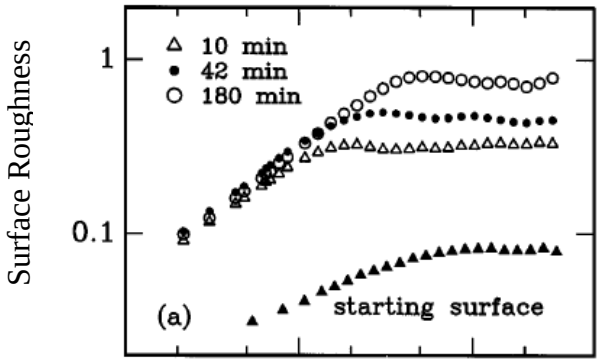


Figure 4.1 Snapshots from 3D SPS during the simulation preformed to produce the results shown in Graph 4.2

Graph 4.3 shows the published lab results for surface roughness [86]. These results depict the same trending shape as that for the foundation as well as the developed KMC model depicted in Graph 4.1 and 4.2 respectively. The results shown in Graph 4.3 were obtained via in situ STM.



Graph 4.3 Published laboratory results showing the surface roughness during growth obtained via in situ STM at various growth temperatures [86]

The results in Graph 4.3 show the overall trending of surface roughness during surface deposition processes. The overall trending effect is compared to that of Graph 4.1 and 4.2 which show a good correlation. The physical date for Graph 4.3 was not available so the only analysis that can be done is the comparison of the overall trending effect that is present during growth. This result agrees with the physics that describes the surface reactions occurring during surface growth. The initial kinetic roughening is caused by adatoms formation, which then leads to a step edge location from which growth will occur. The surface edge continues to propagate which causes the completion of a monolayer. This then leads to another layer being formed, which is the growth

described by the BET growth isotherm discussed in Chapter 2. Figure 4.1 shows an STM micrograph of the surface described in Graph 4.3. This figure shows the formation of multiple layers, which is described by the BET growth isotherm.

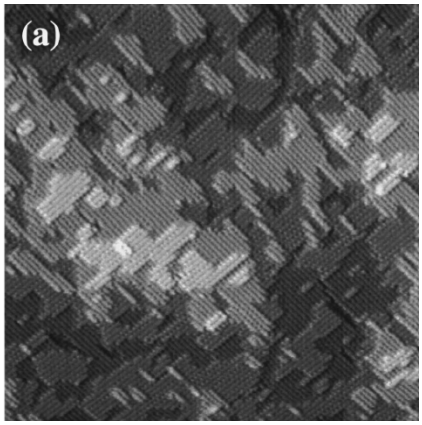


Figure 4.2 STM image of surface topology during mulitlayer growth using MBE

The conclusion of the comparison between foundation model results shown in Graph 4.2 and KMC simulation results shown Graph 4.2 show a close resemblance in the overall trend, values of surface roughness, timing scale and location of maximum values. This correlation shows the ability of the KMC simulation tool to reproduce the foundation model results, which have been shown to have a true likeness to laboratory results.

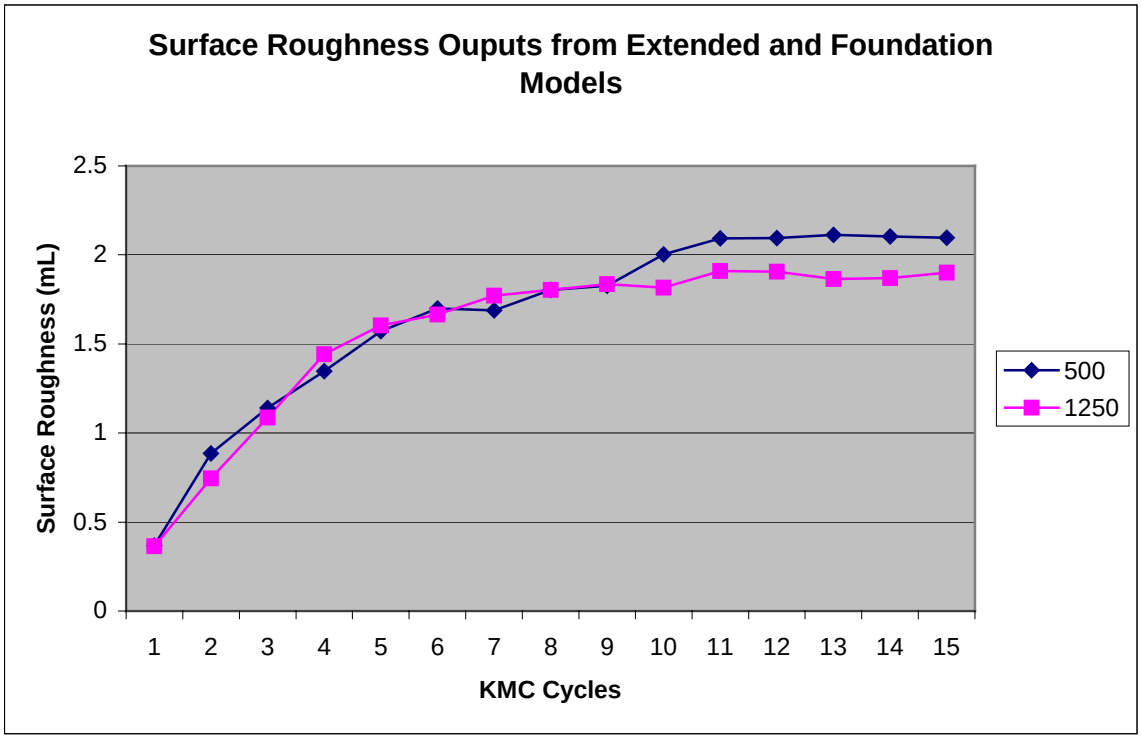
4.3 Model Extension

Several extensions were applied to the foundation model to enhance its capabilities in being applied in a wider range of applications as well as increasing accuracy in describing surface morphology. The ability to use different crystal orientations for the substrate was added. This allows for the evaluation of the affects crystal orientation has on the surface morphology during deposition. Enhancements were made in the site selection process for the overall KMC model simulation. This allowed fro a decrease in cluster processing. Cluster processing occurs when portions of the surface are over analyzed by repeatingly being selected from the RNG. The incorporation of a pseudo RNG such as a Sobol RNG decreases the cluster processing and gives a higher degree of uniformity across site selection. Another alteration was the use of probability functions to describe the incoming flux distribution. This allows another increase in expandability by describing irregular surface distributions that could be caused during aperture or via deposition. This allows for a view of the incoming flux that can be altered from uniform to other distributions such as Gaussian. This feature is not used in the current simulations preformed in this study. The ability to incorporate alloy growth systems was another extension applied to this model. The KMC simulation has the ability to accept multiple impinging atomic specie and describe the surface transformation. The last addition to the model was the calculation and incorporation of surface and strain energy, which was used to describe the strain driven Stranski-Krastanow growth that produced island during SiGe/Si deposition. This effect will be described in the model application in Chapter 5. The following section shows the comparison of the foundation model results produced

via the KMC tool to the results produced via the KMC tool with developed extensions

4.3.1 Foundation vs. Extended

The extension applied to the KMC model have not altered the foundation physical governing equations derived in Chapter 2 and used in section 4.2. Graph 4.4 shows a comparison of the foundation model results produced in Graph 4.2 with the KMC simulation with added extensions.



Graph 4.4 Comparison of Surface Roughness outputs from foundation KMC simulation with extended KMC simulation at T = 650K and P = 1e-5Torr

Graph 4.4 shows different effects caused by adding the extension to the KMC

simulation tool. The first and most noticeable change is the increased surface roughness for the extended model. This can be explained by the change in the surface structure used for simulating. The surface analyzed in the foundation model assumes a flat grid like surface for the initial deposition. This is apparent by the initial surface roughness value set to 0. The extended model accounts for various crystal orientations by initializing the substrate surface to account for the various bonding sites that are available in the specified orientation ([100] for this simulation). This causes an initial surface roughness, which is apparent from the initial value shown for the extended model. This also causes a larger variation in the average height of the film as well as a variation of the height for a selected site. The combination of these affects causes a higher surface roughness with larger oscillations for the given calculation methodology, but does not change the physics governing the surface reactions. Therefore the change in shown in the absolute scale does not correspond to a change in the actual film morphology; rather it shows the inclusion of the physical crystal surface compared to that of a flat grid.

A small change is also noticeable in maximum location but as discussed before this is due to the inherent randomness of the MC process. The exact locations of any given transformation can only be approximated with any MC process since it uses random simulation processes. To achieve higher accuracies on actual values the KMC simulation would have to be repeated several times and then all data would be averaged. Using the law of central tendency the averaged values of at least 30 or more KMC simulations will show a normal distribution whose mean approaches the value obtained from physical results.

The following figures show the screen shots taken during the extended KMC growth simulation shown in Graph 4.4. The screen shot shows how the surface evolves during deposition, which is in agreement with the previous discussion as well as the physical lab result presented in Graph 4.3.

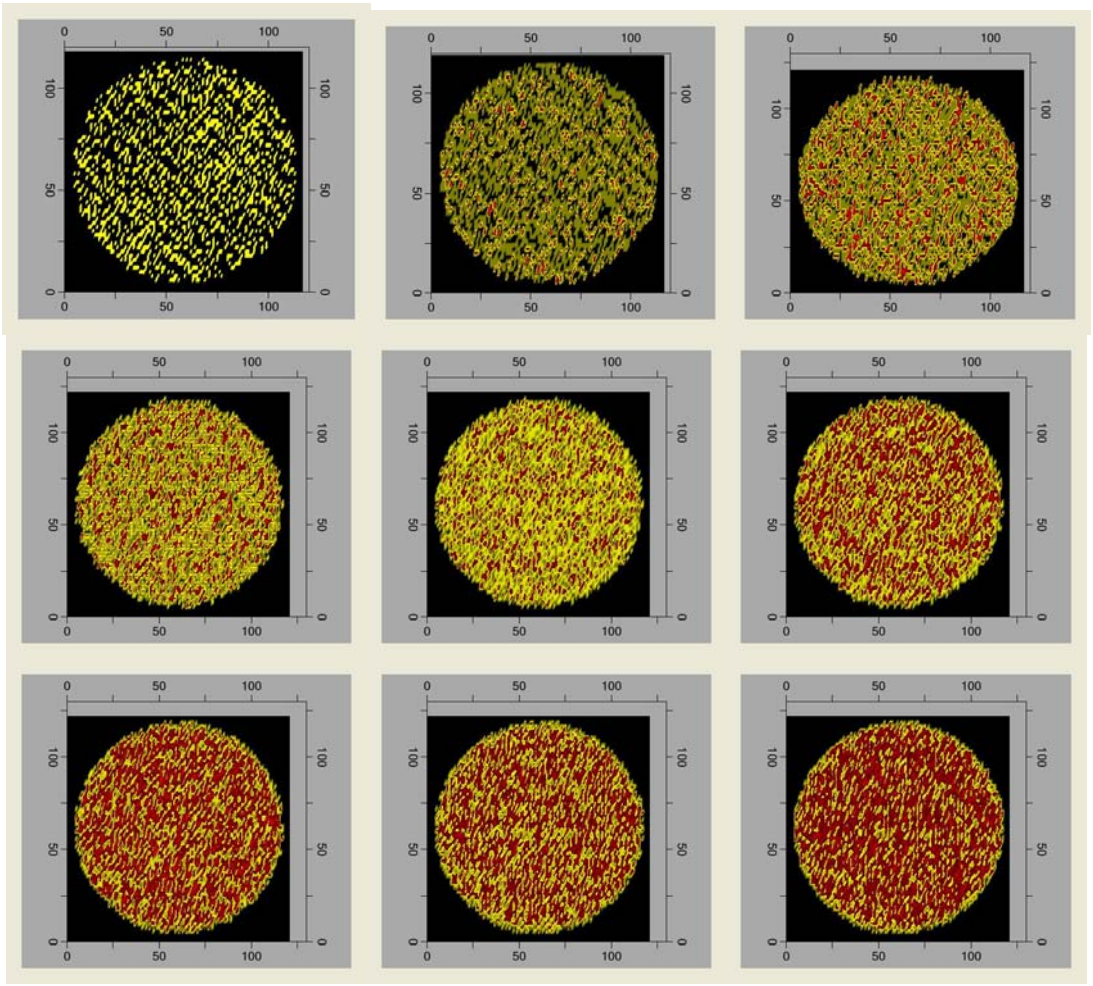


Figure 4.3 Snapshots taken during KMC simulation at T = 650K and P = 1e-5Torr,which depicts the change in surface during growth, which correlates to the surface roughens calculations presented.

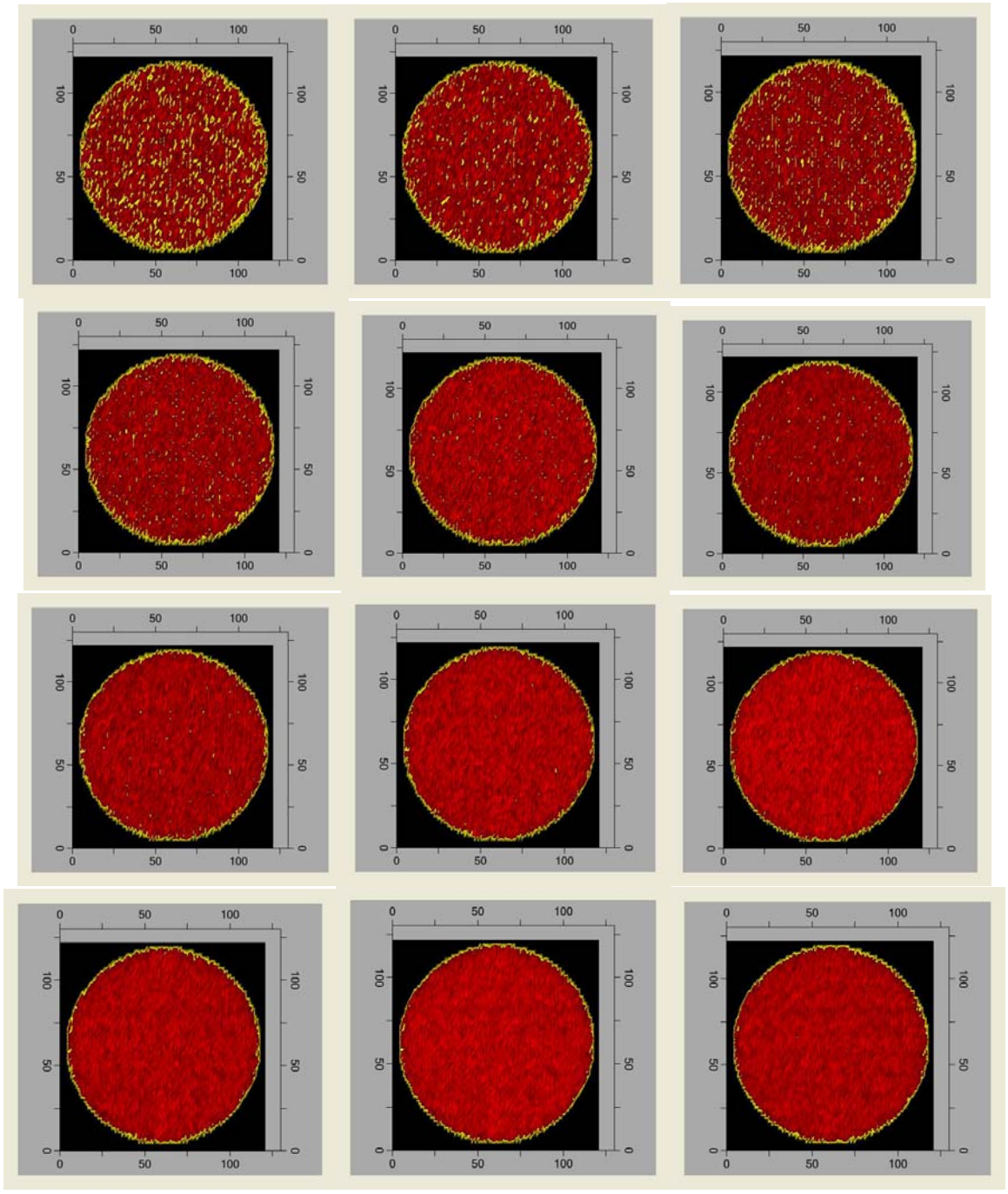
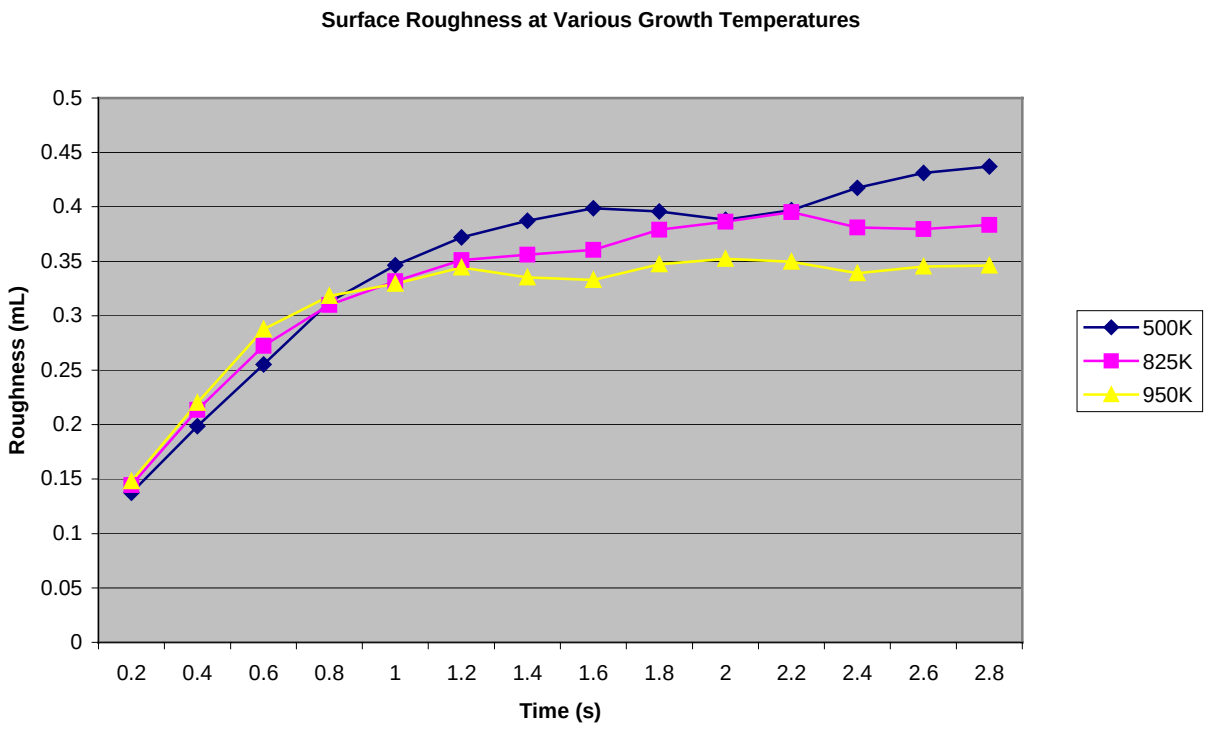


Figure 4.4 Continuation of snapshots taken during KMC simulation at T = 650K and P = 1e-5Torr,which depicts the change in surface during growth, which correlates to the surface roughens calculations presented.

Graph 4.5 is presented to show the correlation between surface roughness and

growth temperature.



Graph 4.5 Output of KMC simulation surface roughness at various growth temperatures

The data presented in Graph 4.5 shows how the surface will grow under smoother conditions as temperature rises. This is noted by the decrease in surface roughness at higher temperatures. This is due to the increased thermal energy departed to adatoms from the surface. This increased energy enhances the ability of surface diffusion and is captured in the diffusion model equation via the exponential term. The decrease in surface roughness shows that an increase in lateral diffusion has occurred which in

limiting the increase in height of the film. This result correlates with the physics that describe film morphology under varying temperatures.

The next section describes the addition of a pseudo RNG into the KMC simulation. This leads to higher degree of uniformity of site selection. This change, once again, has not altered the governing physics equations and showed no effect in the overall surface transition calculations.

4.3.2 Sobol Random Number Generator

One of the first step that is required in every KMC cycle is the ability to randomly select a site on the surface to test. This site should be random to eliminate any computational dependence that may arrive and the site selections should be uniform across the surface. In chapter 3 an example was shown that demonstrated the difference between a uniform RNG and a Sobol RNG. The uniform RNG did not sample evenly across the surface, in fact clustering was found which left some areas over sampled while other areas where never evaluated. This can lead to "cluster processing" on the surface. This can cause the results to be skewed and not correctly depict surface reactions. Figure 4.5 shows the result after a single KMC was executed on the surface under test.

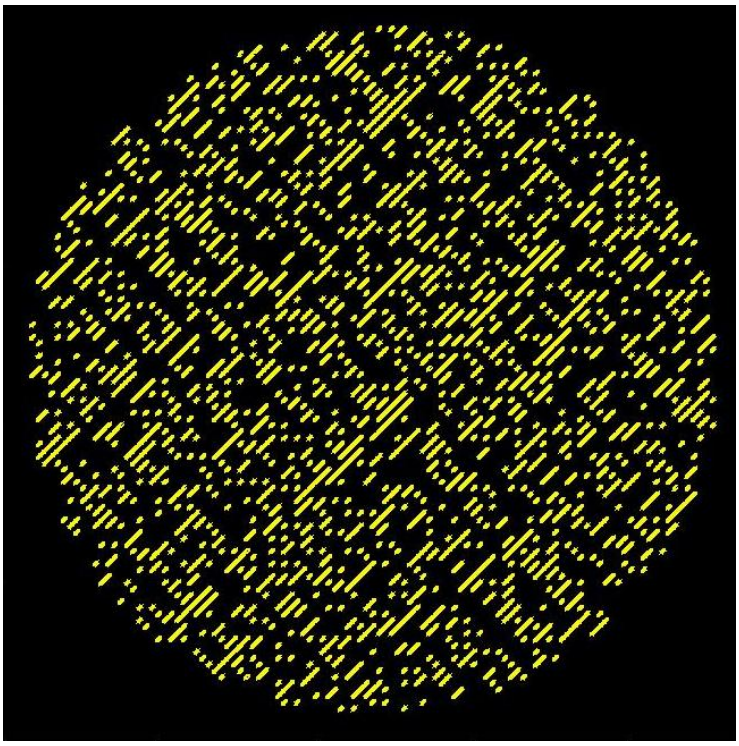


Figure 4.5 Depiction of surface after a single KMC iteration using Sobol RNG for site selection

The surface in Figure 4.5 shows an evenly distributed pattern on the surface. The cluster formations are minimized and the black spaced not analyzed on the surface has also been minimized. In contrast Figure 4.6 shows the surface after a single KMC iteration under the same growth conditions, but using a uniform RNG to select sites for evaluations.

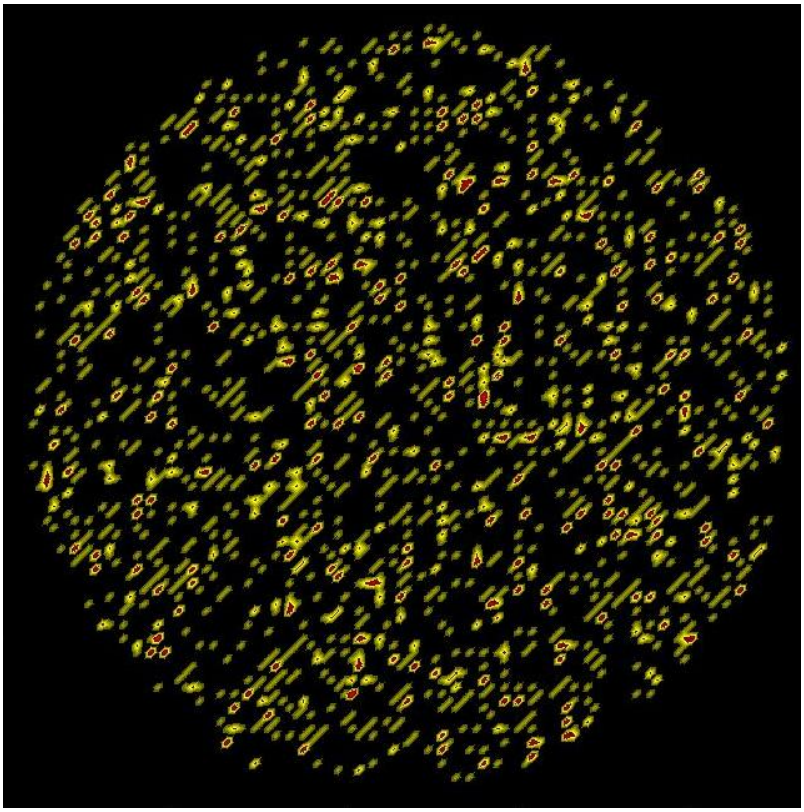


Figure 4.6 Depiction of surface after a single KMC iteration using a uniform RNG for site selection

Figure 4.6 shows a dramatically difference that is obtained by using a uniform RNG. The cluster is much more apparent, which can be seen by the red coloring in Figure 4.6. The red island in Figure 4.6 shows an increase in surface height. This means these areas have at minimum 2 atoms stacked on each other. This is a direct result of "cluster processing" associated with uniform RNGs. In addition to the increased in processing in some areas on the surface, areas can also be found on the surface where no processing has occurred. These areas are denoted by black. The black spacing found within the circle shows how portions of the surface can be neglected and thereby can

cause a skew in modeling results.

In comparison the Sobol RNG surface pattern seems to have a reduced amount of "randomness" than its uniform counterpart. This is due to its memory function, which helps eliminate selecting the same site more than once. The pattern produced by the pseudo RNG should have little to no affect on the overall performance of the KMC simulation. The only goal of the Sobol RNG was to randomly select sites on the surface uniformly, therefore a pattern on the surface will not affect the processing as long as the pattern is not formed in the same order in every KMC iteration.

In the case where a random number is selected to account for a random event or to select a random process, a pseudo RNG such as a Sobol RNG would not be the correct choice. In these applications a uniform RNG is required. These process require a number selection in which every number has the same probability of being selected, which is not the case in a RNG with a memory. Therefore, for every case where a random number is required for process or transition selection a uniform RNG will be used.

4.3.3 Simulation with Non Planar Surface

This section will analyze the growth kinetics involved with surface morphology with a non-planar initial growth surface. The developed KMC model will simulate an initial island on the surface and show the surface roughness over time as well as the surface morphology at high and low temperatures.

The island structure will cause a larger swing in the initial kinetic roughening of the surface, due the average height of the film will now include the island height. As the films grows and approaches the height of the island the surface roughness will fall. The surface roughness will eventually reach a steady state value that it will oscillate around as before once the island has been completely consumed by the film.

This simulation will show how the step edge will affect the growth of the film as well as show how the KMC simulation accurately depicts the overall surface morphology of the film. The surface is not analyzed on the atomic scale rather as the energetic average and probabilities of surface transitions. For this reason the periodic surface potentials seen by surface diffusion adatoms is not accounted for. The only possible transitions for the surface adatoms are those shown in Figure 3.3[53]. These transitions and assumptions will not allow for an understanding of individual atomic movement as a MD simulation would, rather it will show the overall characteristic of the surface and describe the overall film parameters such as surface roughness.

Stating that, the step edge will collect adatoms as they diffuse on the surface. The diffusion process will increase at higher temperatures due to the increase in thermal energy added to the adatoms. This increased energy will cause more surface diffusions and thereby cause the step edge to smooth over the growth at a higher temperature. Figure 4.7 shows the surface being tested after the first KMC cycle has been completed and Figure 4.8 shows a close up of the island that is located on the test surface.

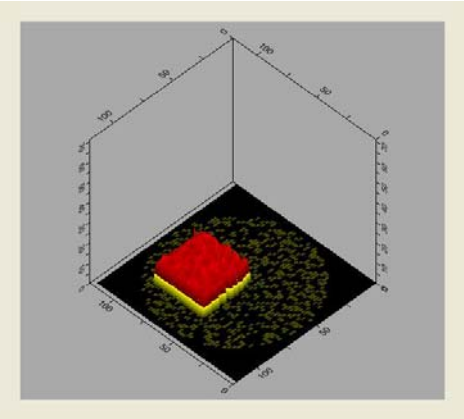


Figure 4.7 Depiction of the initial non planar surface used to analyze surface morphology with a step edge.

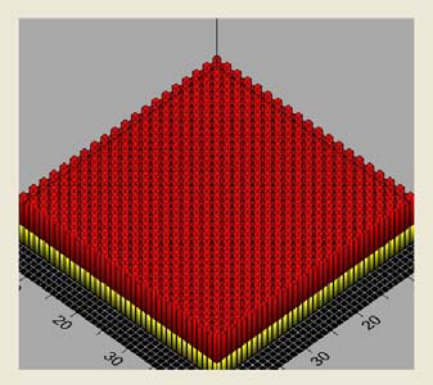
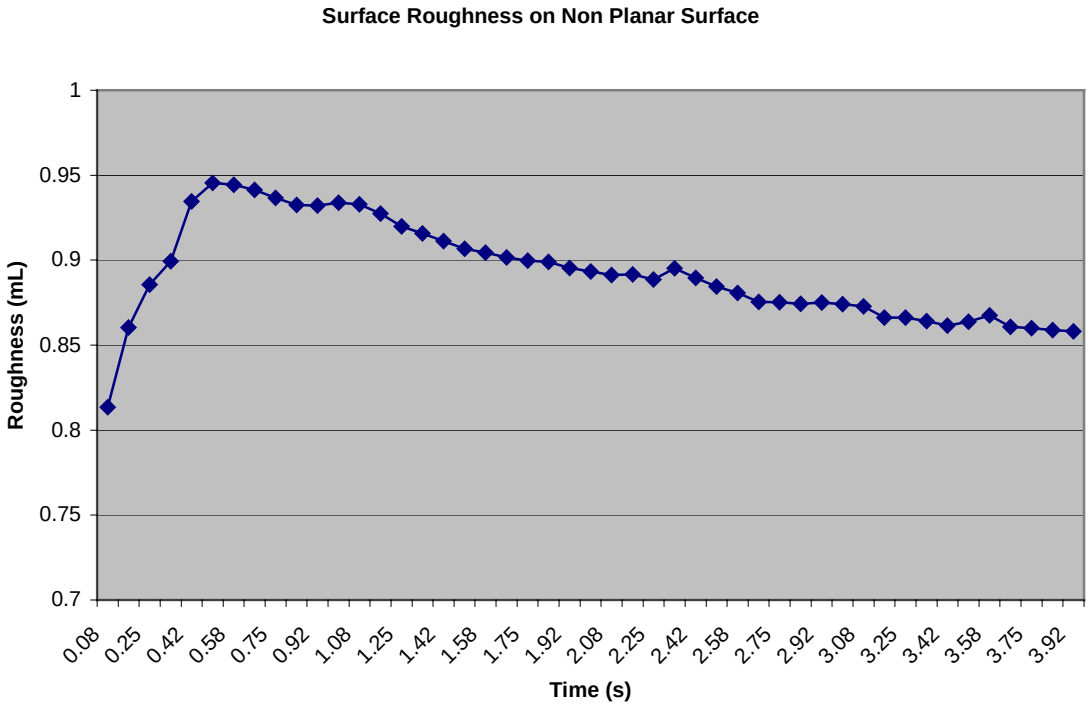


Figure 4.8 Depiction of the island located on test surface used to analyze surface morphology with step edge

Graph 4.6 shows the surface roughness plot from the KMC simulation.



Graph 4.6 Surface roughness output from KMC for non planar surface with island at T = 950K and P = 1e-5Torr

Graph 4.6 shows a higher initial surface roughness than before due to the height of the island. The kinetic roughening for the initial maximum occurs around 0.6 seconds. Oscillations can then be seen occurring periodically as the growth continues. This is caused by the same effect found in early results where the formation and completion of monolayers during growth. The initial kinetic roughening causes the increase in surface roughness, but due to surface diffusion and growth capture at the step edge the surface roughness decreases as the height of the deposited film approaches that of the initial island. Around 4 seconds the surface roughness begins to decrease at a slower rate, which is caused by the film height becoming closer to the height of the island. Figure 4.9

shows the screen shots associated with the step edge growth.

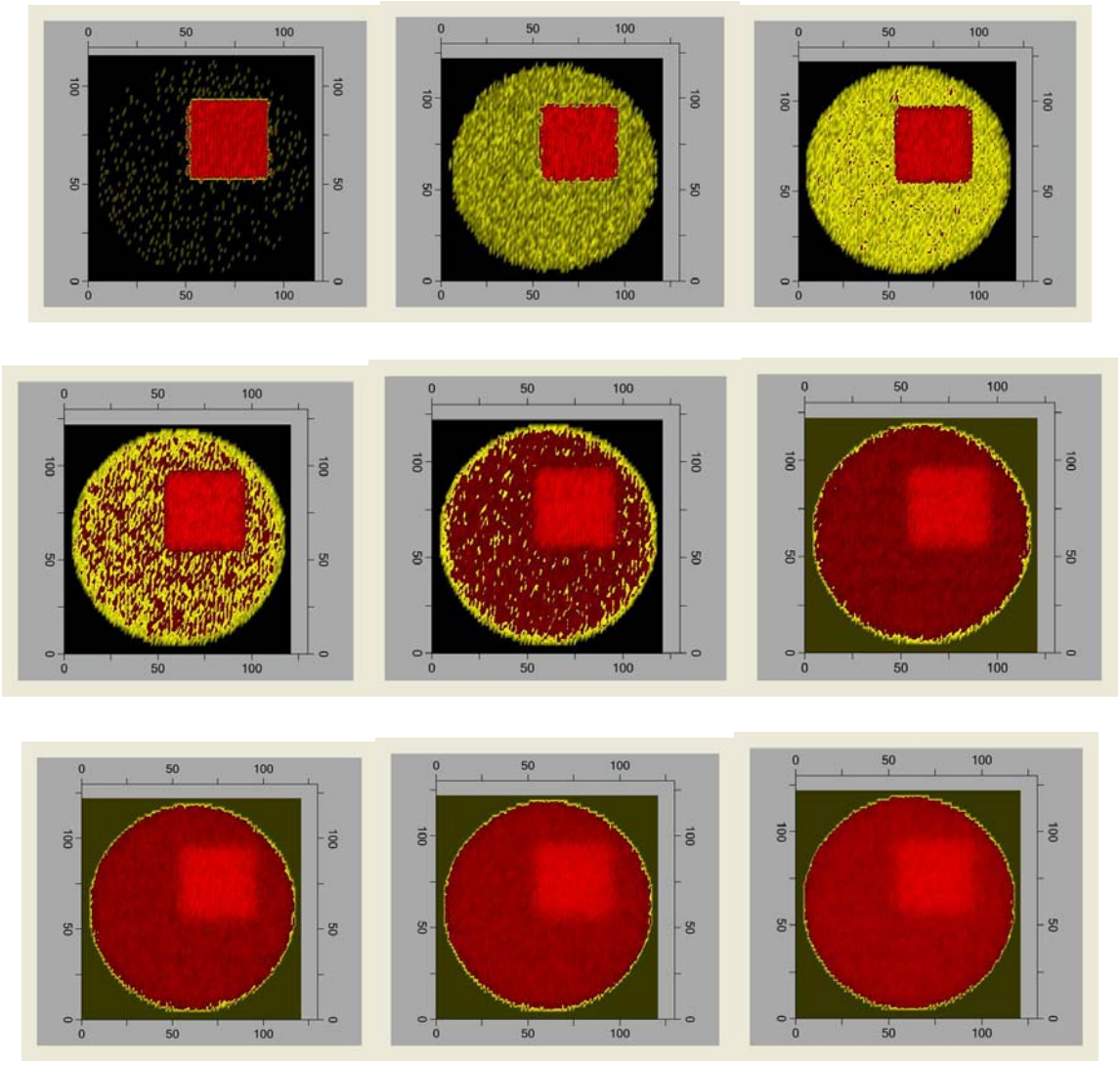


Figure 4.9 Snapshot of the surface morphology during the step edge growth analysis preformed with the KMC simulation at T = 950K and P = 1e-5Torr

The snapshots show the step edge propagating into the film due to the diffusion process of adatoms. This process is caused by the surface diffusion of adatoms. As the adatoms move around the surface they are captured by step edges. The increased bonds obtained by the step edge cause any further surface diffusion to become difficult. This in

turn causes the step edge to propagate which also acts as sink for surface diffused adatoms. The figures also show the same kinetic roughening and growth process that was previously shown in the planar surface results. Figure 4.10 shows a side profile of the growth morphology of the surface over time for the initial island defect surface high temperature.

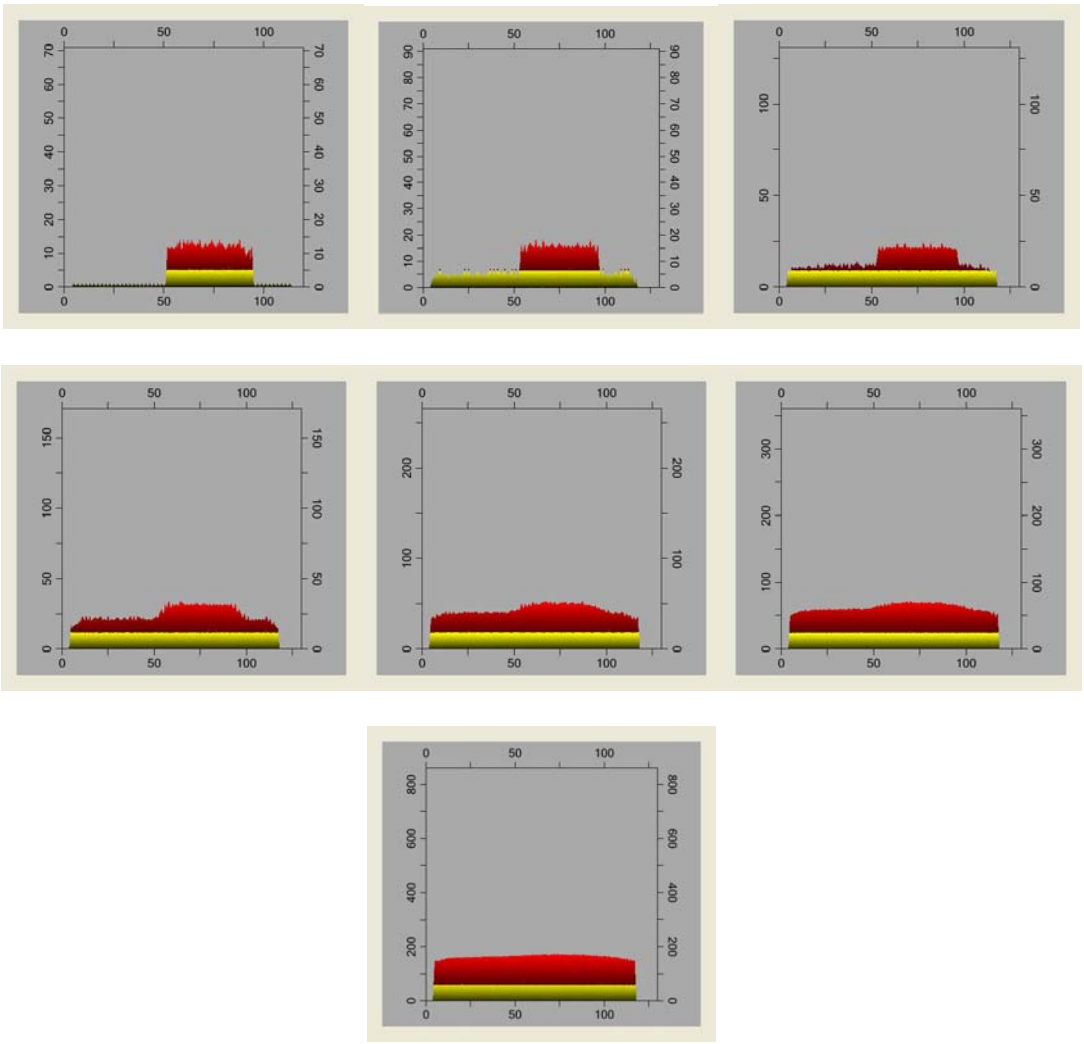


Figure 4.10 Side depiction of the surface morphology during the step edge growth analysis preformed with the KMC simulation at T = 950K and P = 1e-5Torr

Figure 4.10 shows how the added thermal energy allows for the step edge of the island to expand and smooth. This process agrees with the governing physics of the problem and shows a good correlation with experimental results.

CHAPTER 5
APPLICATION OF KMC GROWTH MODEL

5.1 Introduction

This chapter describes the application of the strain and surface energy extension added to the KMC model. The driving energy for the growth will be discussed and analyzed. Data will show the transition that occurs during growth from strain driven to surface energy driven morphologies. This transition will also cause a change in island topologies found on the surface. The topologies will be examined along with their dependence on growth parameters such as temperature and alloy composition. The size distribution of islands will be analyzed along with its dependence on temperature as well as the shape and placement of the island on the surface.

5.2 Alloy Deposition Model

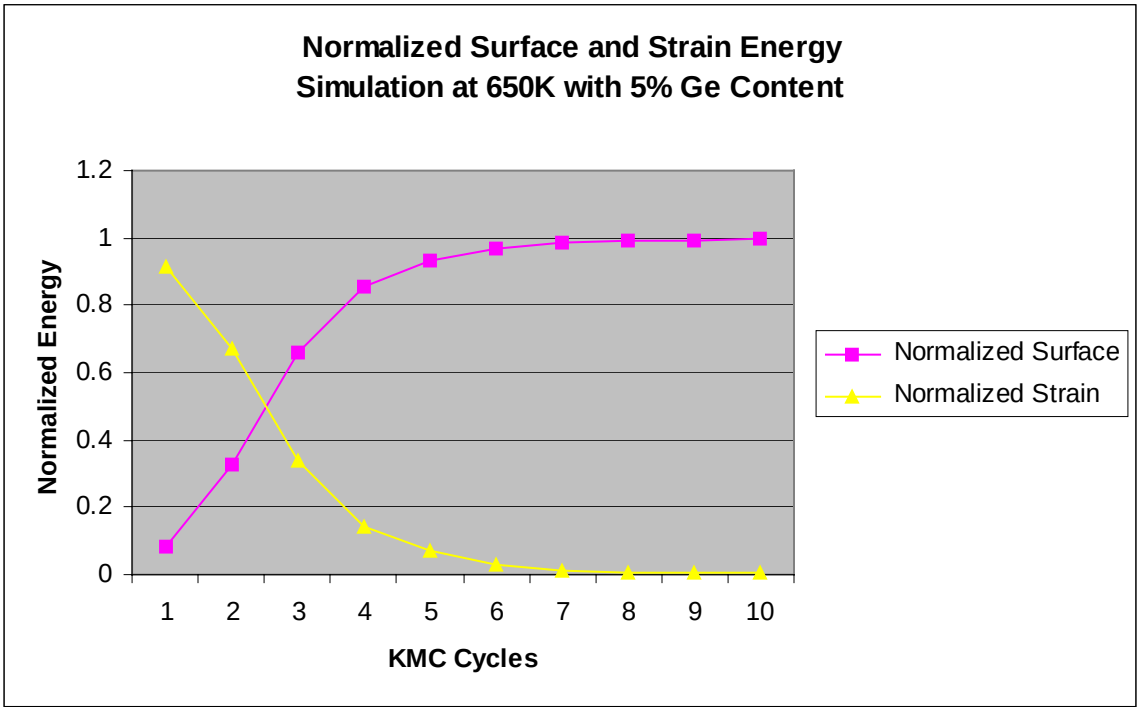
This section will cover the simulation results for alloy growth. The growth system used for analysis is SiGe/Si. This study analyzes the different energy consideration that must be accounted for during growth as well as island topologies and transformations. Analysis of island shape and placement is also discussed.

5.2.1 Energy Considerations

It has been shown that island formation and shape transitions in SiGe/Si growth is caused by a competition between anisotropy strain energy and anisotropy surface energy of the evolving surface via Stranski-Krastanow growth. Due to the strained film, relaxation occurs by roughening of the surface through island formation. As the island height grows and relaxation occurs the strain energy decreases but the surface energy increases. Due to the ability of the surface to obtain such a high level of relaxation, the island will widen once the strain relaxation no longer dominates the total energy of the system. At this point the system will begin coalescence of the island structures.

The coalescence reduces the surface energy of the system by combing the smaller islands into the larger islands. The transition from strain energy to surface energy driven morphology is also the transition between island topologies. Initial island topologies are considered "hut" shapes. The shapes are taller thinner islands. This is where strain energy is has the largest affect in the growth process. This is readily understandable from the increased height, which decreases the strain energy, while the decreased diameter increases the overall surface energy. Once strain relaxation has occurred the island topologies change to a "dome" formation. The dome formation is the attempt of the system to minimize the surface energy that was maximized during hut formation. The smoothing of the island, also the surface, causes a reduction in the surface energy as well as the overall system energy. Graph 5.1 shows the normalized surface and strain energy of the system. The strain and surface energies were normalized with respect to the total

energy of the system. This shows a definite transition from strain driven to surface driven growth.



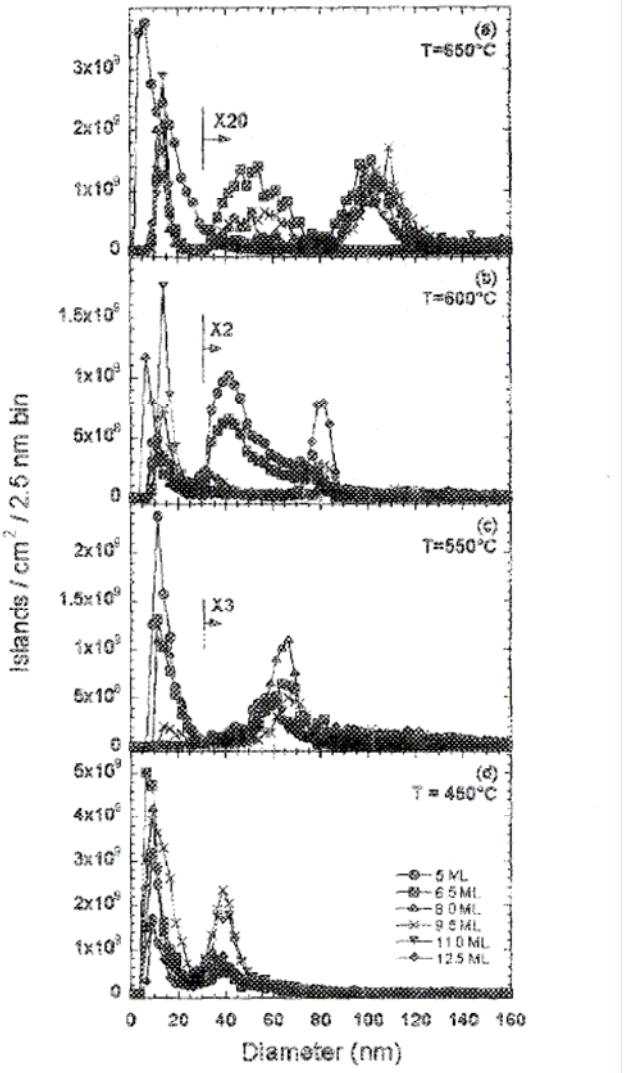
Graph 5.1 Depiction of normalized surface and strain energies which shows transition from strain drive to surface driven growth

Graph 5.6 shows a depiction of the normalized surface and strain energy vs KMC growth cycles. The graph agrees with the governing physics that describes relaxation through island formation. As growth continues the islands decrease the strain through relaxation, which in noted by the decrease in normalized strain energy in Graph 5.1. This process also causes a change in surface energy. Surface energy increases due to the increase in bonding sites, which is illustrated in Graph 5.1 as an increase in normalized surface energy. Between the second and third KMC cycle a transition can be seen

between the dominate energy of the system. This in turn changes the surface reactions that will occur as deposition continues. The overall energy of the system will continually try to be minimized. After this transition point occurs the surface morphology will transition from a vertical growth (increase in island height) to a horizontal spreading of existing island. This point causes the transition of island topologies found during deposition. This transition point can be effected by both temperature and alloy composition, both are described in detail below.

5.2.2 Island Topologies

The topology of the islands has a high dependence on temperature. At lower temperatures the island formations have a smaller diameter, 'skinny', and are huts (depicted as peak islands in 3D SPS). At high temperatures larger diameter, 'fat', domes (depicted as flat top islands in 3D SPS) are produced. The midrange temperature causes a mix in the island formations. The topology changes are due to the surface mobility caused by changes in temperatures as well as the balance between surface and strain energies. Graph 5.2 shows published laboratory results that depict the island populations by diameter for various growth temperatures.



Graph 5.2 Depiction of island densities by diameter fro various growth temperatures

Graph 5.2 shows the bimodality of island diameters that is produced during the growth of SiGe island structures. This bimodality is a cause of the initial roughening from seed islands that then grow into hut formations as well as the transformation into dome islands as the growth continues. The bimodality seems to always be present due to the seeding nature required to begin island formation. The change in island diameters as

temperature changes is caused by the change in the transition point of strain and stress energies during deposition. At higher temperatures the transition occurs earlier due to the increase ability for the film to relax thereby decreasing strain energy as well as the increased ability for adatoms to diffuse thereby decreasing surface energy. This effect therefore allows the islands to being transition to dome shapes earlier.

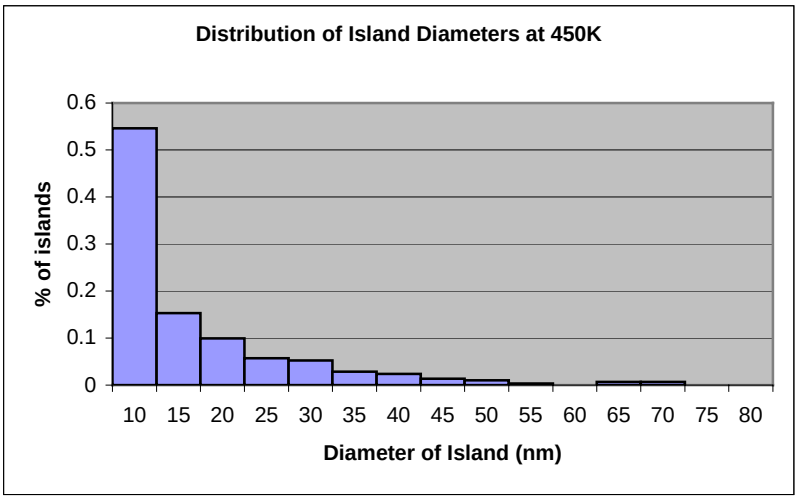
As temperature increases the islands begin to show a trimodal distribution. This is due to the increased energy supplied to the surface from thermal energy as well as coalescence.

The increased thermal energy allows adatoms to have more mobility and causes a larger decrease in the overall system energy by minimizing the dangling bonds on the surface. This causes a smoothing of the surface and the transition of island structures from hut to dome topologies. This transition is also the cause for coalescence mechanism.

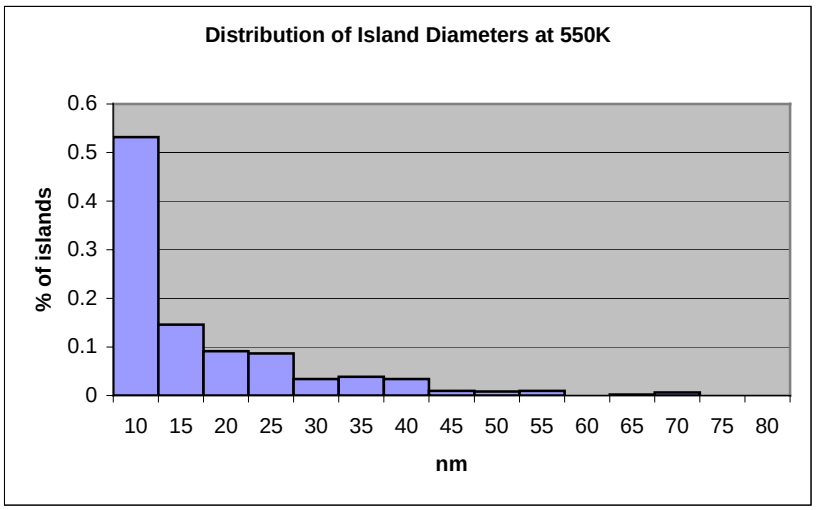
Coalescence occurs due to the spreading of the surface island structures. These islands diameter increase which in turns causes multiple islands to combine. This is also a leading reason for the decrease in the proportion of seeding islands at higher temperatures. The larger islands are covering more surface area therefore there is not as much area exposed where seeding will be required. This process approaches more of the selective growth process described in Chapter 2. The larger islands grow at the expense of the smaller islands, which causes a shift in the proportions of island dimension on the surface.

Graphs 5.3, 5.4 and 5.5 show the island distributions produced from the KMC

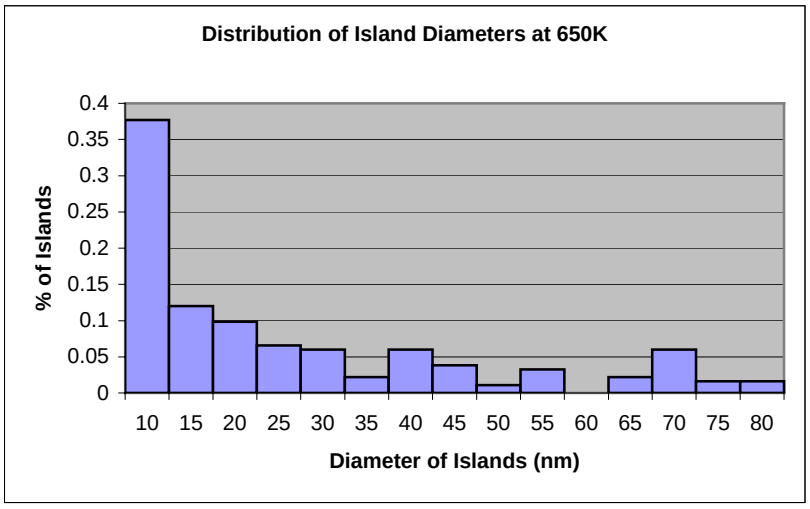
simulation at 450K, 550K and 650K growth temperatures respectively. All simulations preformed at various growth temperatures were done at a constant growth pressure of 1.5E-5 Torr. The graphs show how the population of island diameters changes from lower temperature to higher temperatures, which is the same as the physical results show in Graph 5.2.



Graph 5.3 Distribution of island diameters for KMC simulation with a growth temperature of 450K and pressure of 1.5E-5 Torr



Graph 5.4 Distribution of island diameters for KMC simulation with a growth temperature of 550K and pressure of 1.5E-5 Torr



Graph 5.5 Distribution of island diameters for KMC simulation with a growth temperature of 650K and pressure of 1.5E-5 Torr

Comparison between the Graph 5.3 at 450K and Graph 5.5 at 650K shows how the KMC simulation accounts for the transition of island formations base upon growth temperature. The physical results presented in Graph 5.2 show an island diameter of 10nm as the largest population at 450K. This is the same result shown in Graph 5.3.

Graph 5.2 shows a second distribution spike around 40nm, which is present in the simulation results shown in Graph 5.3, but it is not a local maximum. At 650K Graph 5.2 shows a trimodal distribution with local maximum approximately at 10,40 and 80nm. Graph 5.2 can also be misleading due to the magnification that occurs at various portion of the graph. The magnification in the 650K shows a magnification of 20 times. This was done to show the presence of the increased diameter islands, but the concentration of those islands is not as large as that of the 10nm islands. Graph 5.5 also shows a trimodal distribution with local maximum approximately at 10,40,70nm. The transition from bimodal to trimodal is evident between Graphs 5.3 and 5.5 as it is in the physical results presented in Graph 5.2. The approximate size distributions between Graphs 5.2, 5.3, and 5.5 also show a correlation. The variance between the simulation and physical results leads to the belief that other affects such as Ge solute effect at interface along with Ge segregation need to be accounted for to have a higher accuracy. The assumptions used ignore these effects but could be added in future work. Although these assumptions have shown a variance from physical results, the overall changes in topologies and size are within an acceptable error for simulation results.

Figures 5.1, 5.2 and 5.3 show sample island formations produced by the KMC model at 450-650K temperature ranges as rendered by 3D SPS. The figures show a correlation to the physical results obtained as well as the description presented above. The goal of figures 5.1-5.3 is to show how 3D SPS displays the simulation results so they can be further analyzed. The hut formations are depicted via a pointed island structure, while the dome islands have a flat surface. The 3D SPS package uses a triangle meshing

algorithm which causes the sharp transitions on the surface of the islands. This representation can look somewhat misleading, since the actual island structures do not correspond to pointed and flat top rather each has more of a smoother topology in the physical results. The sharpness of the structure depicted in figures 5.1-5.3 is an artifact of the 3D SPS rendering software.

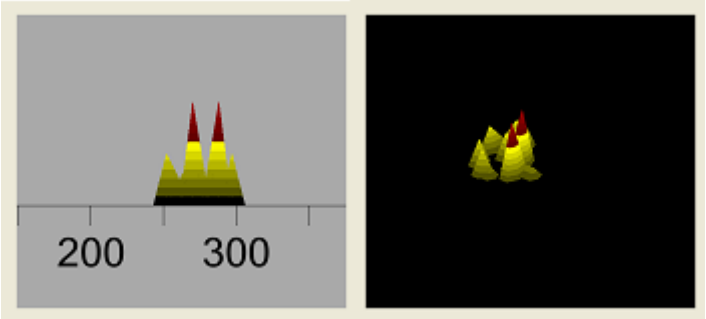


Figure 5.1 Depiction of island formation at 450K and pressure of 1.5E-5 Torr. The island formation are huts with smaller diameters

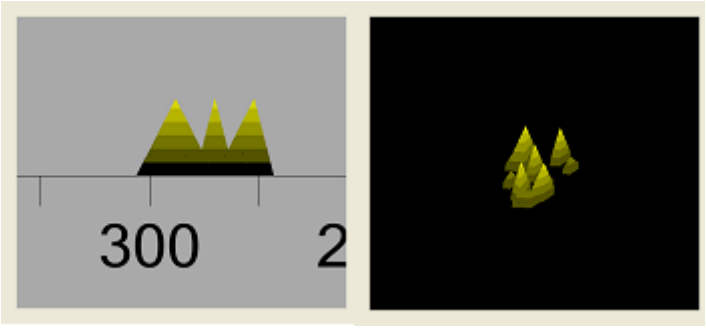


Figure 5.2 Depiction of island formation at 550K and pressure of 1.5E-5 Torr. The island formations have both skinny and fat islands

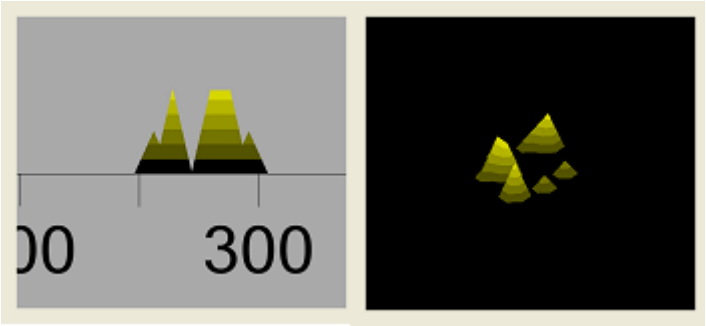


Figure 5.3 Depiction of island formation at 650K and pressure of 1.5E-5 Torr. The island formations are fatter with dome shaped top (flat top)

The results presented in figure 5.1-5.3 directly correlate to the temperature dependance discussed earlier. The lower temperature simulation results of Graph 5.1 show hut formations while figure 5.3 shows dome formations. The differences in the figure 5.1-5.3 are due to the shift caused in the transition point by the change in temperature described earlier.

Figure 5.5 and 5.5 show a topographical and side profile view at 650K and 450K respectively. The comparison of 5.5 and 5.5 show that the lower temperature deposition produced skinner and taller hut formations while the higher temperature produced thicker and shorter dome formations.

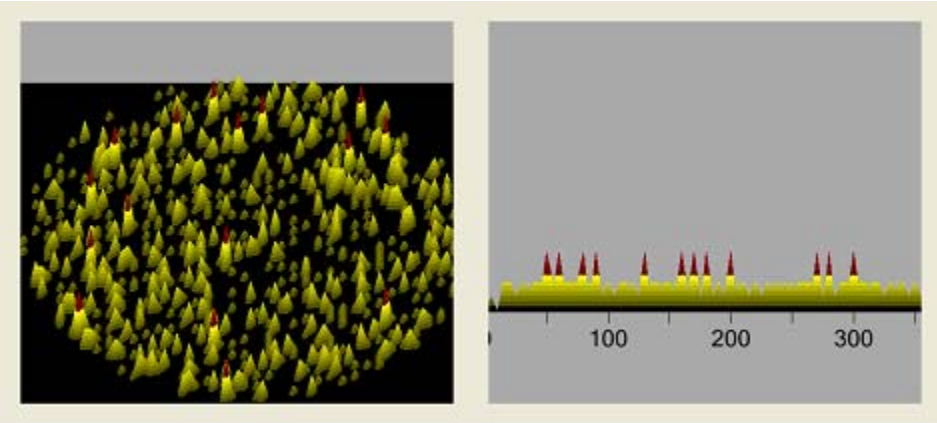


Figure 5.4 Depiction of island formation at 650K and pressure of 1.5E-5 Torr. The produced islands are fatter and shorter as well as the second tier island have the majority of dome shape (flat top)

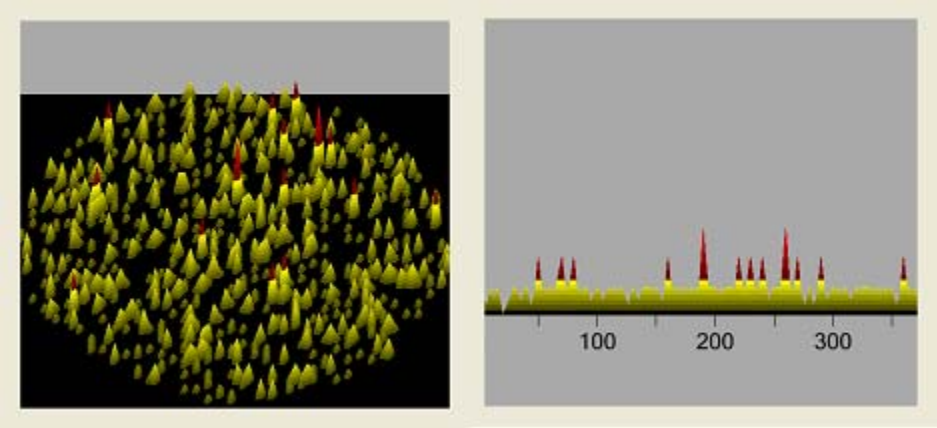
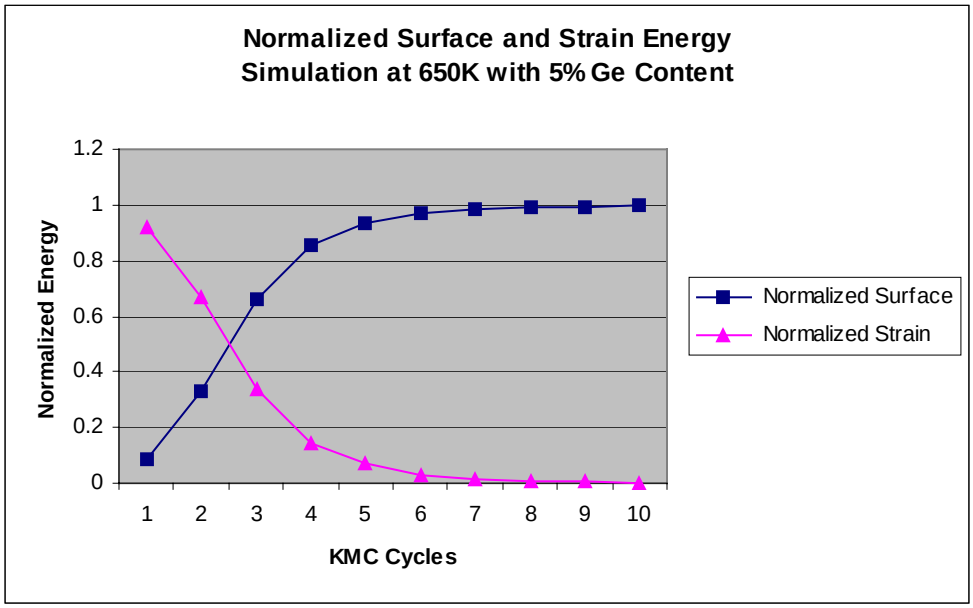


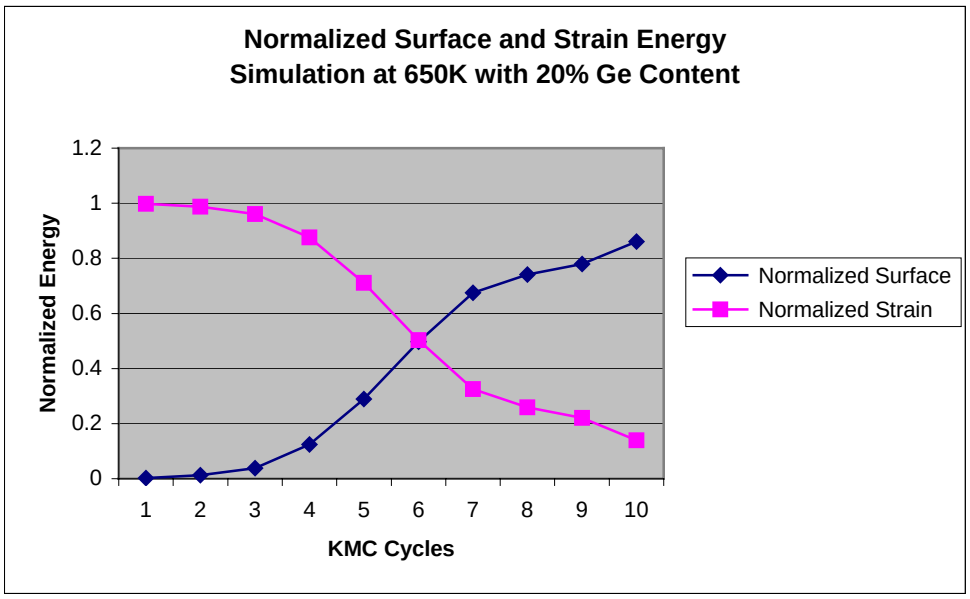
Figure 5.5 Depiction of island formation at 450K and pressure of 1.5E-5 Torr. The produced islands are skinner and taller as well as the second tier islands have the majority of hut shape (pointed top)

The transition from hut to dome shape can also be affected by the content of Ge in the alloy. This does cause a shift in transition point, but is not as dramatic as that is shown with changes in temperature. The change in the transition point is due to the increased stress added by the extra Ge atoms. The added atoms cause an increase in the overall strain in the island structure. This in turn requires taller island structures to

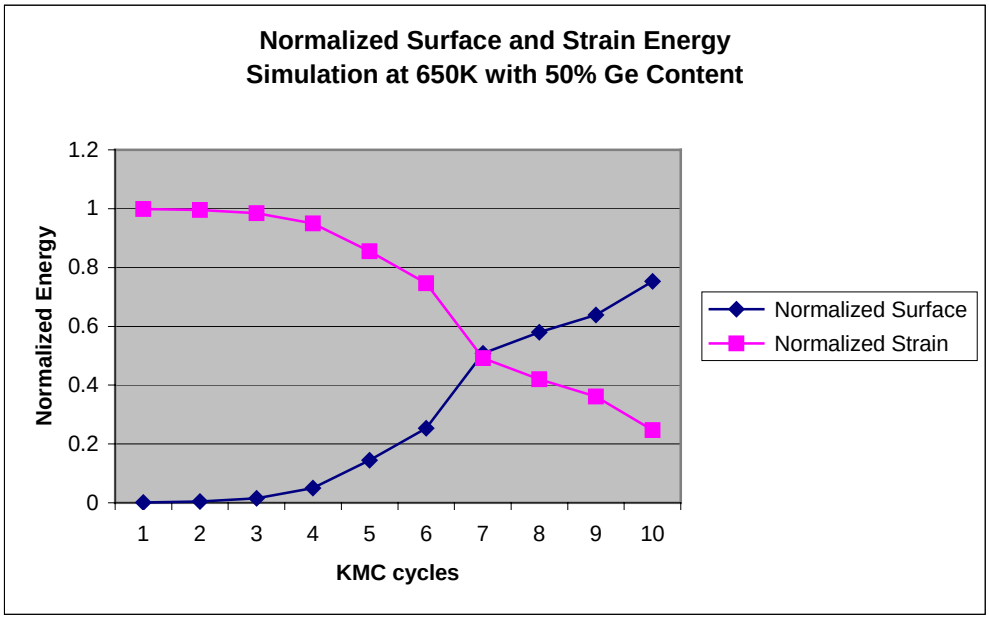
reduce the strain, which shifts the transition from hut to dome topologies. Graph 5.6, 5.7 and 5.8 show the normalized surface and strain energies vs. KMC cycles for 5%, 20% and 50% Ge concentrations.



Graph 5.6 Depiction of normalized surface and strain energies which shows transition from strain drive to surface driven growth with 5% Ge content and pressure of 1.5E-5 Torr



Graph 5.7 Depiction of normalized surface and strain energies which shows transition from strain drive to surface driven growth with 20% Ge content and pressure of 1.5E-5 Torr



Graph 5.8 Depiction of normalized surface and strain energies which shows transition from strain drive to surface driven growth with 50% Ge content and pressure of 1.5E-5 Torr

Evaluating the results presented in graphs 5.6-5.8 shows the transition points shifts from 2.5-7 KMC cycles. The increased amount of Ge atoms causes an increased amount of strain in the island to surface interface. This in turn requires the islands to grow taller in order to relax the strain. This is apparent by the shift in transition point from 2.5-7KMC cycles.

5.2.3 Island Locations and Shapes

The shape transition of an island from hut to dome can be modeled, yet the actual shape and location of the islands are completely random. This is due to random processing that takes place during deposition. This processing leads to a random placement of islands across the surface during the deposition process. Both experimental and simulation results agree with this finding. The ability to create self aligned quantum dots (SAQD) directly from deposition does not seem readily possible with the current growth conditions and setup. Alignment could be directed via pre placed surface defects that would increase the adsorption probability of various sites on the substrate surface. Annealing has also shown promise in the ability to align the island structures once the deposition process is complete. The current KMC simulation tool does not include an annealing routine, but has the flexibility for the addition of such a module.

The shape of the island structures is as random as their placement on the surface. Island formation show to have various shapes during the deposition process, which is

partially from the random processing as well as the coalescence of island during the growth process. This irregular shape also greatly limits the ability for SAQD during deposition. Once again annealing has shown promise in causing not only uniformly distributed islands on the surface but also uniformly shaped islands as well.

Figures 5.6 and 5.7 show the model outputs compared to published laboratory results, which shows the randomness in location and shape of the islands. 5.6 show the island formation from a topographical view. The shapes of the islands are irregular in shape due to coalescence and randomly distributed. Figure 5.7 shows an angled view from a 3D rendered AFM image as well as an angle view from the 3D SPS output of the KMC simulation of islands. Once again the random shape and location of islands can be seen as well as the coalescence of islands on the surface. The importance of figure 5.6 and 5.7 is to show the irregular shape and location distribution, it is not meant to be a side by side comparison of the same scale.

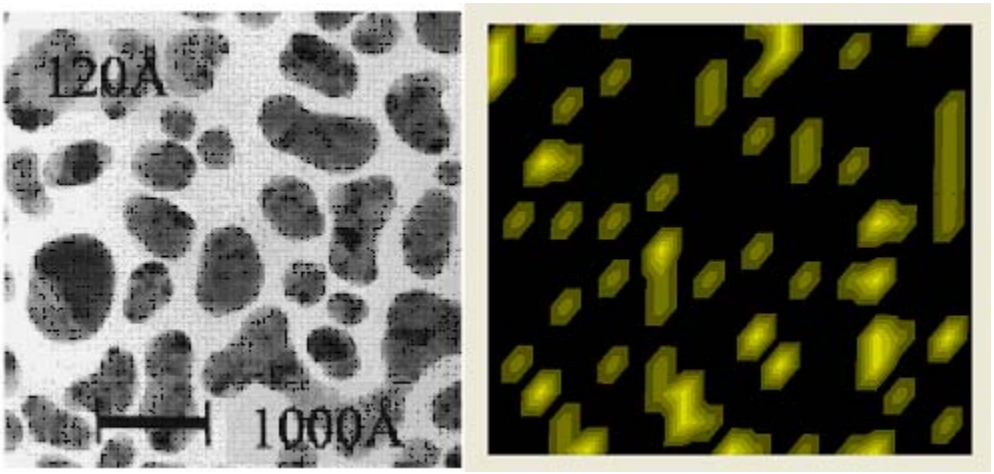


Figure 5.6 Shows topographical view of side by side comparison of physical and simulated results for island shape and placement on the surface during deposition at T=550K and P=1e-5Torr

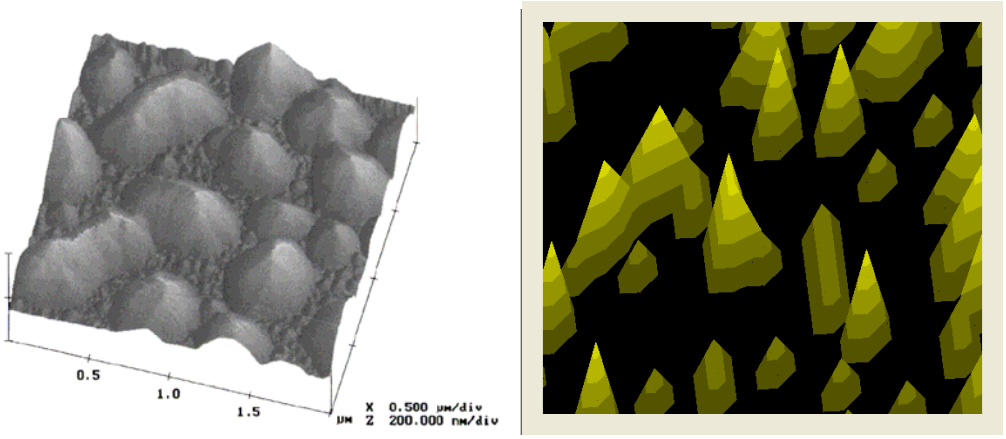


Figure 5.7 Shows angled view of side by side comparison of physical and simulated results for island shape, placement and coalescence on the surface during deposition T=550K and P=1e-5Torr

The KMC model expansion for alloy only assumed growth in the Si[100] direction and did not account for any defect formation or Ge segregation or Ge solute affect at SiGe/Si interface. Theses affects can causes changes in surface morphology and could be added at a later date to give a more accurate depiction of island formation.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The developed KMC allows for a flexible platform that can accurately describe surface morphology during various growth conditions. The models have expanded the capabilities of existing simulation tools to account for multiple affects in a single growth simulation tool. The model included pseudo RNG in the help of site selection and added the crystalline orientation to allow for an increase in growth simulation accuracy.

The crystal orientation has shown to have an affect on growth morphology during strained film growth such as that shown in SiGe/Si. The various crystal orientations cause a change in surface energy as well as strain energy. The altering of these parameters can dramatically cause variations in island formation, topology and transitions.

The strain energy alloy model shows the different affects various growth parameters have on SiGe/Si island formations. The model showed the affects Ge concentration and temperature had on overall island topology and morphology. The ability to create self-aligned quantum dots (SAQDs) during deposition was analyzed and determined to be improbable with current growth techniques. The work shows that an

annealing stage will need to present to align islands as well as convert to a uniform topology of uniform size and shape.

6.2 Future Work

The KMC simulation tool developed in this study has many advantages over existing modules. One of the most important advantages it the flexibility of the model. The KMC simulation tool was designed to allow for easy upgrades and additions of libraries and functions to increase the tools effectiveness in multiple applications. One such modification could be the addition of an annealing stage.

An annealing stage would allow further insight into using SiGe/Si deposition for the formation of self aligned quantum dots (SAQDs). The research results preformed show the ability to create SAQDs directly during deposition is not feasible with the current processing techniques being used. By adding an annealing stage further study could be preformed on analyzing the transition of the random shaped islands into uniformly shaped as well as uniformly distributed.

Further study is needed in the solute affect found at the SiGe/Si interface with Ge. Studies have shown that Ge can diffuse into the wetting layer and cause various affects during growth. Analysis needs to be preformed to account for the governing equations of this motion. These equations then need to be transformed into a probability density function that could then be adapted to fit the current KMC engine.

Currently the model does not allow for defect formation during the growth

process. More work should be done to account for defect formation such as grain boundaries, which can arise during strained growth. These affects will cause changes in device operations as well overall surface morphology. The current model is limited to coherently strained modules, by adding the ability to account for defect formation the models effectiveness could be dramatically increased.

Ge content inside of the island structures is also of interest. The Ge content in islands affects the materials optical and electrical properties, which in turn can alter device operations of semiconductor devices. Understanding the mixture and placement/configuration of Ge atoms inside island formations would lead to a better understanding of device operations as well as device limitations.

Interest has also been shown in coupling growth models with device models. This allows for a better understanding in how the growth parameters can directly affect the operation of semiconductor devices. This could also help in leading to real time analysis system that will determine device yield prior to completion of the fabrication process. This in turn can save time and money during the development stages of device development.

To increase processing time the programming topology could be converted to a object oriented multithreaded operation to take advantage of the Hyper Threading technology now available with AMD and Pentium processors. This would allow for multiple portions of the simulation to be performed simultaneously there by allow for a dramatic increase in performance time.

APPENDIX A

VB.NET CODE FOR SURFACE PLOTTING SOFTWARE

```
Imports System
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms
Imports System.Data
Imports System.Diagnostics
Imports System.Drawing.Printing
Imports System.IO

Imports C1.Win.C1Chart3D
'/ <summary>
'/ Summary description for Form1.
'/ </summary>
Public Class Form1
   Inherits System.Windows.Forms.Form
    WithEvents numericUpDown1 As
System.Windows.Forms.NumericUpDown
   WithEvents numericUpDown2 As
System.Windows.Forms.NumericUpDown
   WithEvents numericUpDown3 As
System.Windows.Forms.NumericUpDown
```

```
Private mainMenu1 As System.Windows.Forms.MainMenu
WithEvents menuItem1 As System.Windows.Forms.MenuItem
WithEvents menuItem2 As System.Windows.Forms.MenuItem
WithEvents menuItem3 As System.Windows.Forms.MenuItem
WithEvents menuItem4 As System.Windows.Forms.MenuItem
WithEvents menuItem5 As System.Windows.Forms.MenuItem
WithEvents menuItem6 As System.Windows.Forms.MenuItem
WithEvents menuItem7 As System.Windows.Forms.MenuItem
WithEvents timer1 As System.Windows.Forms.Timer
WithEvents checkBox1 As System.Windows.Forms.CheckBox
WithEvents checkBox2 As System.Windows.Forms.CheckBox
WithEvents checkBox3 As System.Windows.Forms.CheckBox
WithEvents menuItem8 As System.Windows.Forms.MenuItem
WithEvents menuItem9 As System.Windows.Forms.MenuItem
WithEvents menuItem10 As System.Windows.Forms.MenuItem
WithEvents menuItem11 As System.Windows.Forms.MenuItem
WithEvents menuItem12 As System.Windows.Forms.MenuItem
WithEvents menuItem13 As System.Windows.Forms.MenuItem
WithEvents menuItem14 As System.Windows.Forms.MenuItem
WithEvents menuItem15 As System.Windows.Forms.MenuItem
WithEvents menuItem16 As System.Windows.Forms.MenuItem
Private components As System.ComponentModel.IContainer
WithEvents menuItem17 As System.Windows.Forms.MenuItem
WithEvents menuItem18 As System.Windows.Forms.MenuItem
```

```
WithEvents menuItem19 As System.Windows.Forms.MenuItem
WithEvents menuItem20 As System.Windows.Forms.MenuItem
WithEvents menuItem21 As System.Windows.Forms.MenuItem
WithEvents menuItem22 As System.Windows.Forms.MenuItem
WithEvents menuItem23 As System.Windows.Forms.MenuItem
WithEvents menuItem24 As System.Windows.Forms.MenuItem
Private c1Chart3D1 As C1.Win.C1Chart3D.C1Chart3D
Private label1 As System.Windows.Forms.Label
Private label2 As System.Windows.Forms.Label
Private label3 As System.Windows.Forms.Label
 '
Private angleIncrement As Integer = 2
Private setPoint As Chart3DDataSetPoint
Private setGrid As Chart3DDataSetGrid
Private setIrGrid As Chart3DDataSetIrGrid
Private pointData(27) As Chart3DPoint
Private pointData1(27) As Chart3DPoint
Private contourData(,) As Double
WithEvents menuItem25 As System.Windows.Forms.MenuItem
WithEvents menuItem26 As System.Windows.Forms.MenuItem
WithEvents menuItem27 As System.Windows.Forms.MenuItem
WithEvents menuItem28 As System.Windows.Forms.MenuItem
WithEvents menuItem29 As System.Windows.Forms.MenuItem
```

```
    WithEvents menuHelpAbout As
System.Windows.Forms.MenuItem
    Private bUpdate As Boolean = True

    Public Sub New()
        InitializeComponent()

        '
        ' TODO: Add any constructor code after
InitializeComponent call
        '
        numericUpDown1.Minimum = - 360
        numericUpDown1.Maximum = 360
        numericUpDown1.Increment = CDec(angleIncrement)
        numericUpDown1.Value =
CDec(c1Chart3D1.ChartArea.View.RotationX)

        numericUpDown2.Maximum = 360
        numericUpDown2.Minimum = - 360
        numericUpDown2.Increment = CDec(angleIncrement)
        numericUpDown2.Value =
CDec(c1Chart3D1.ChartArea.View.RotationY)

        numericUpDown3.Maximum = 360
```

```
        numericUpDown3.Minimum = - 360
        numericUpDown3.Increment = CDec(angleIncrement)
        numericUpDown3.Value =
CDec(c1Chart3D1.ChartArea.View.RotationZ)
    End Sub 'New


    '/ <summary>
    '/ Clean up any resources being used.
    '/ </summary>
    Public Shadows Sub Dispose(disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub 'Dispose


#Region " Windows Form Designer generated code "
    '/ <summary>
    '/ Required method for Designer support - do not modify
    '/ the contents of this method with the code editor.
```

```
    '/ </summary>
    Friend WithEvents GroupBox1 As
System.Windows.Forms.GroupBox
    Friend WithEvents Button1 As
System.Windows.Forms.Button
    Friend WithEvents GroupBox2 As
System.Windows.Forms.GroupBox
    Friend WithEvents CheckBox4 As
System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox5 As
System.Windows.Forms.CheckBox
    Friend WithEvents StatusBar1 As
System.Windows.Forms.StatusBar
    Friend WithEvents StatusBarPanel1 As
System.Windows.Forms.StatusBarPanel
    Friend WithEvents StatusBarPanel2 As
System.Windows.Forms.StatusBarPanel
    Friend WithEvents StatusBarPanel3 As
System.Windows.Forms.StatusBarPanel
    Friend WithEvents PictureBox1 As
System.Windows.Forms.PictureBox
    Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container
```

```
        Dim resources As System.Resources.ResourceManager =
New System.Resources.ResourceManager(GetType(Form1))
        Me.c1Chart3D1 = New C1.Win.C1Chart3D.C1Chart3D
        Me.numericUpDown2 = New
System.Windows.Forms.NumericUpDown
        Me.numericUpDown3 = New
System.Windows.Forms.NumericUpDown
        Me.numericUpDown1 = New
System.Windows.Forms.NumericUpDown
        Me.mainMenu1 = New System.Windows.Forms.MainMenu
        Me.menuItem14 = New System.Windows.Forms.MenuItem
        Me.menuItem21 = New System.Windows.Forms.MenuItem
        Me.menuItem22 = New System.Windows.Forms.MenuItem
        Me.menuItem23 = New System.Windows.Forms.MenuItem
        Me.menuItem15 = New System.Windows.Forms.MenuItem
        Me.menuItem16 = New System.Windows.Forms.MenuItem
        Me.menuItem19 = New System.Windows.Forms.MenuItem
        Me.menuItem28 = New System.Windows.Forms.MenuItem
        Me.menuItem24 = New System.Windows.Forms.MenuItem
        Me.menuItem20 = New System.Windows.Forms.MenuItem
        Me.menuItem4 = New System.Windows.Forms.MenuItem
        Me.menuItem27 = New System.Windows.Forms.MenuItem
        Me.menuItem26 = New System.Windows.Forms.MenuItem
        Me.menuItem25 = New System.Windows.Forms.MenuItem
```

```
        Me.menuItem5 = New System.Windows.Forms.MenuItem
        Me.menuItem6 = New System.Windows.Forms.MenuItem
        Me.menuItem7 = New System.Windows.Forms.MenuItem
        Me.menuItem17 = New System.Windows.Forms.MenuItem
        Me.menuItem18 = New System.Windows.Forms.MenuItem
        Me.menuItem1 = New System.Windows.Forms.MenuItem
        Me.menuItem2 = New System.Windows.Forms.MenuItem
        Me.menuItem12 = New System.Windows.Forms.MenuItem
        Me.menuItem13 = New System.Windows.Forms.MenuItem
        Me.menuItem3 = New System.Windows.Forms.MenuItem
        Me.menuItem11 = New System.Windows.Forms.MenuItem
        Me.menuItem8 = New System.Windows.Forms.MenuItem
        Me.menuItem9 = New System.Windows.Forms.MenuItem
        Me.menuItem10 = New System.Windows.Forms.MenuItem
        Me.menuItem29 = New System.Windows.Forms.MenuItem
        Me.menuHelpAbout = New
System.Windows.Forms.MenuItem
        Me.timer1 = New
System.Windows.Forms.Timer(Me.components)
        Me.checkBox1 = New System.Windows.Forms.CheckBox
        Me.checkBox2 = New System.Windows.Forms.CheckBox
        Me.checkBox3 = New System.Windows.Forms.CheckBox
        Me.label1 = New System.Windows.Forms.Label
        Me.label2 = New System.Windows.Forms.Label
```

```
        Me.label3 = New System.Windows.Forms.Label
        Me.GroupBox1 = New System.Windows.Forms.GroupBox
        Me.Button1 = New System.Windows.Forms.Button
        Me.GroupBox2 = New System.Windows.Forms.GroupBox
        Me.CheckBox5 = New System.Windows.Forms.CheckBox
        Me.CheckBox4 = New System.Windows.Forms.CheckBox
        Me.StatusBar1 = New System.Windows.Forms.StatusBar
        Me.StatusBarPanel1 = New
System.Windows.Forms.StatusBarPanel
        Me.StatusBarPanel2 = New
System.Windows.Forms.StatusBarPanel
        Me.StatusBarPanel3 = New
System.Windows.Forms.StatusBarPanel
        Me.PictureBox1 = New
System.Windows.Forms.PictureBox
        CType(Me.c1Chart3D1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.numericUpDown2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.numericUpDown3,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.numericUpDown1,
System.ComponentModel.ISupportInitialize).BeginInit()
        Me.GroupBox1.SuspendLayout()
```

```
        Me.GroupBox2.SuspendLayout()
        CType(Me.StatusBarPanel1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.StatusBarPanel2,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.StatusBarPanel3,
System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'c1Chart3D1
        '
        Me.c1Chart3D1.Anchor =
System.Windows.Forms.AnchorStyles.None
        Me.c1Chart3D1.BackColor =
System.Drawing.SystemColors.Control
        Me.c1Chart3D1.Location = New
System.Drawing.Point(232, 8)
        Me.c1Chart3D1.Name = "c1Chart3D1"
        Me.c1Chart3D1.PropBag = "<?xml
version=""1.0""?><Chart3DPropBag Version=""""><View
/><ChartGroupsCollection><C" & _
        "hart3DGroup><ChartData><Set
type=""Chart3DDataSetGrid"" RowCount=""11"" ColumnCount=" &
_
```

```
        """11"" RowDelta=""1"" ColumnDelta=""1""
RowOrigin=""0"" ColumnOrigin=""0"" Hole=""3.4028234" & _
        "7E+38""><Data>4.5 3.6 2.89999986 2.39999986 2.1
1.99999988 2.1 2.39999986 2.89999" & _
        "986 3.6 4.5 8.1 7.2 6.5 6 5.7 5.6 5.7 6 6.5 7.2
8.1 10.9 10 9.3 8.8 8.5 8.4 8.5 " & _
        "8.8 9.3 10 10.9 12.9 12 11.3 10.8 10.5 10.4 10.5
10.8 11.3 12 12.9 14.1 13.2 12." & _
        "5 12 11.7 11.6 11.7 12 12.5 13.2 14.1 14.5 13.6
12.9 12.4 12.1 12 12.1 12.4 12.9" & _
        " 13.6 14.5 14.1 13.2 12.5 12 11.7 11.6 11.7 12
12.5 13.2 14.1 12.9 12 11.3 10.8 " & _
        "10.5 10.4 10.5 10.8 11.3 12 12.9 10.9 10 9.3 8.8
8.5 8.4 8.5 8.8 9.3 10 10.9 8.1" & _
        " 7.2 6.5 6 5.7 5.6 5.7 6 6.5 7.2 8.1 4.5 3.6
2.89999986 2.39999986 2.1 1.9999998" & _
        "8 2.1 2.39999986 2.89999986 3.6
4.5</Data></Set></ChartData></Chart3DGroup></Cha" & _
        "rtGroupsCollection><StyleCollection><NamedStyle
Name=""LabelStyleDefault"" ParentN" & _
        "ame=""Control""
StyleData=""BackColor=Transparent;"" /><NamedStyle
Name=""Header"" Par" & _
```

```
        "entName=""Control"" /><NamedStyle Name=""Legend""
ParentName=""Control"" StyleData=""Wr" & _
        "ap=False;AlignVert=Top;"" /><NamedStyle
Name=""Footer"" ParentName=""Control"" /><Nam" & _
        "edStyle Name=""Area"" ParentName=""Control""
StyleData=""BackColor=Control;AlignVert=" & _
        "Top;"" /><NamedStyle Name=""Control""
ParentName="""" StyleData=""ForeColor=ControlTex" & _
        "t;BackColor=Control;""
/></StyleCollection><LegendData Compass=""East""
/><FooterDa" & _
        "ta Compass=""South"" /><HeaderData
Compass=""North"" /></Chart3DPropBag>"
        Me.c1Chart3D1.Size = New System.Drawing.Size(552,
496)
        Me.c1Chart3D1.TabIndex = 7
        '
        'numericUpDown2
        '
        Me.numericUpDown2.Location = New
System.Drawing.Point(72, 40)
        Me.numericUpDown2.Name = "numericUpDown2"
        Me.numericUpDown2.Size = New
System.Drawing.Size(48, 20)
```

```
        Me.numericUpDown2.TabIndex = 2
        '
        'numericUpDown3
        '
        Me.numericUpDown3.Location = New
System.Drawing.Point(136, 40)
        Me.numericUpDown3.Name = "numericUpDown3"
        Me.numericUpDown3.Size = New
System.Drawing.Size(48, 20)
        Me.numericUpDown3.TabIndex = 2
        '
        'numericUpDown1
        '
        Me.numericUpDown1.Location = New
System.Drawing.Point(8, 40)
        Me.numericUpDown1.Name = "numericUpDown1"
        Me.numericUpDown1.Size = New
System.Drawing.Size(48, 20)
        Me.numericUpDown1.TabIndex = 2
        '
        'mainMenu1
        '
```

```
        Me.mainMenu1.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.menuItem14,
Me.menuItem4, Me.menuItem1, Me.menuItem29})
        '
        'menuItem14
        '
        Me.menuItem14.Index = 0
        Me.menuItem14.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.menuItem21,
Me.menuItem22, Me.menuItem23, Me.menuItem15, Me.menuItem16,
Me.menuItem19, Me.menuItem28, Me.menuItem24,
Me.menuItem20})
        Me.menuItem14.Text = "File"
        '
        'menuItem21
        '
        Me.menuItem21.Index = 0
        Me.menuItem21.Text = "Open Chart"
        '
        'menuItem22
        '
        Me.menuItem22.Index = 1
        Me.menuItem22.Text = "Save Chart"
        '
```

```
'menuItem23
'
Me.menuItem23.Index = 2
Me.menuItem23.Text = "-"
'
'menuItem15
'
Me.menuItem15.Index = 3
Me.menuItem15.Text = "Load Data"
'
'menuItem16
'
Me.menuItem16.Index = 4
Me.menuItem16.Text = "Save Data"
'
'menuItem19
'
Me.menuItem19.Index = 5
Me.menuItem19.Text = "-"
'
'menuItem28
'
Me.menuItem28.Index = 6
Me.menuItem28.Text = "Save Image"
```

```
        '
        'menuItem24
        '
        Me.menuItem24.Index = 7
        Me.menuItem24.Text = "Print Preview"
        '
        'menuItem20
        '
        Me.menuItem20.Index = 8
        Me.menuItem20.Text = "Exit"
        '
        'menuItem4
        '
        Me.menuItem4.Index = 1
        Me.menuItem4.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.menuItem27,
Me.menuItem26, Me.menuItem25, Me.menuItem5, Me.menuItem6,
Me.menuItem7, Me.menuItem17, Me.menuItem18})
        Me.menuItem4.Text = "Chart"
        '
        'menuItem27
        '
        Me.menuItem27.Index = 0
        Me.menuItem27.Text = "Properties"
```

```
'
'menuItem26
'
Me.menuItem26.Index = 1
Me.menuItem26.Text = "Wizard"
'
'menuItem25
'
Me.menuItem25.Index = 2
Me.menuItem25.Text = "-"
'
'menuItem5
'
Me.menuItem5.Index = 3
Me.menuItem5.Text = "Bar"
'
'menuItem6
'
Me.menuItem6.Index = 4
Me.menuItem6.Text = "Scatter"
'
'menuItem7
'
Me.menuItem7.Index = 5
```

```
        Me.menuItem7.Text = "Surface"
        '
        'menuItem17
        '
        Me.menuItem17.Index = 6
        Me.menuItem17.Text = "-"
        '
        'menuItem18
        '
        Me.menuItem18.Index = 7
        Me.menuItem18.Text = "Interacive"
        '
        'menuItem1
        '
        Me.menuItem1.Index = 2
        Me.menuItem1.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.menuItem2,
Me.menuItem3, Me.menuItem11, Me.menuItem8, Me.menuItem9,
Me.menuItem10})
        Me.menuItem1.Text = "Data"
        '
        'menuItem2
        '
        Me.menuItem2.Index = 0
```

```
        Me.menuItem2.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.menuItem12,
Me.menuItem13})
        Me.menuItem2.Text = "Points"
        '
        'menuItem12
        '
        Me.menuItem12.Index = 0
        Me.menuItem12.Text = "1 series"
        '
        'menuItem13
        '
        Me.menuItem13.Index = 1
        Me.menuItem13.Text = "2 series"
        '
        'menuItem3
        '
        Me.menuItem3.Index = 1
        Me.menuItem3.Text = "Grid"
        '
        'menuItem11
        '
        Me.menuItem11.Index = 2
        Me.menuItem11.Text = "IrregularGrid"
```

```
            '
            'menuItem8
            '
            Me.menuItem8.Index = 3
            Me.menuItem8.Text = "-"
            '
            'menuItem9
            '
            Me.menuItem9.Index = 4
            Me.menuItem9.Text = "Holes"
            '
            'menuItem10
            '
            Me.menuItem10.Index = 5
            Me.menuItem10.Text = "4D"
            '
            'menuItem29
            '
            Me.menuItem29.Index = 3
            Me.menuItem29.MenuItems.AddRange(New
System.Windows.Forms.MenuItem() {Me.menuHelpAbout})
            Me.menuItem29.Text = "&Help"
            '
            'menuHelpAbout
```

```
            '
            Me.menuHelpAbout.Index = 0
            Me.menuHelpAbout.Text = "&About"
            '
            'timer1
            '
            Me.timer1.Interval = 20
            '
            'checkBox1
            '
            Me.checkBox1.Location = New
System.Drawing.Point(56, 40)
            Me.checkBox1.Name = "checkBox1"
            Me.checkBox1.Size = New System.Drawing.Size(16, 24)
            Me.checkBox1.TabIndex = 3
            Me.checkBox1.Text = "checkBox1"
            '
            'checkBox2
            '
            Me.checkBox2.Location = New
System.Drawing.Point(120, 40)
            Me.checkBox2.Name = "checkBox2"
            Me.checkBox2.Size = New System.Drawing.Size(16, 24)
            Me.checkBox2.TabIndex = 4
```

```
        Me.checkBox2.Text = "checkBox2"
        '
        'checkBox3
        '
        Me.checkBox3.Location = New
System.Drawing.Point(184, 40)
        Me.checkBox3.Name = "checkBox3"
        Me.checkBox3.Size = New System.Drawing.Size(16, 24)
        Me.checkBox3.TabIndex = 5
        Me.checkBox3.Text = "checkBox3"
        '
        'label1
        '
        Me.label1.Location = New System.Drawing.Point(8,
24)
        Me.label1.Name = "label1"
        Me.label1.Size = New System.Drawing.Size(48, 16)
        Me.label1.TabIndex = 8
        Me.label1.Text = "Rot X"
        '
        'label2
        '
        Me.label2.Location = New System.Drawing.Point(72,
24)
```

```
        Me.label2.Name = "label2"
        Me.label2.Size = New System.Drawing.Size(48, 16)
        Me.label2.TabIndex = 9
        Me.label2.Text = "Rot Y"
        '
        'label3
        '
        Me.label3.Location = New System.Drawing.Point(136,
24)
        Me.label3.Name = "label3"
        Me.label3.Size = New System.Drawing.Size(48, 16)
        Me.label3.TabIndex = 10
        Me.label3.Text = "Rot Z"
        '
        'GroupBox1
        '
        Me.GroupBox1.Controls.Add(Me.Button1)
        Me.GroupBox1.Controls.Add(Me.label1)
        Me.GroupBox1.Controls.Add(Me.numericUpDown1)
        Me.GroupBox1.Controls.Add(Me.checkBox1)
        Me.GroupBox1.Controls.Add(Me.label2)
        Me.GroupBox1.Controls.Add(Me.numericUpDown2)
        Me.GroupBox1.Controls.Add(Me.checkBox2)
        Me.GroupBox1.Controls.Add(Me.numericUpDown3)
```

```
        Me.GroupBox1.Controls.Add(Me.label3)
        Me.GroupBox1.Controls.Add(Me.checkBox3)
        Me.GroupBox1.Location = New System.Drawing.Point(8,
96)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(208,
112)
        Me.GroupBox1.TabIndex = 12
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "VIEW"
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(8,
72)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(192, 23)
        Me.Button1.TabIndex = 11
        Me.Button1.Text = "Topological View"
        '
        'GroupBox2
        '
        Me.GroupBox2.Controls.Add(Me.CheckBox5)
        Me.GroupBox2.Controls.Add(Me.CheckBox4)
```

```
        Me.GroupBox2.Location = New System.Drawing.Point(8,
216)
        Me.GroupBox2.Name = "GroupBox2"
        Me.GroupBox2.Size = New System.Drawing.Size(208,
56)
        Me.GroupBox2.TabIndex = 13
        Me.GroupBox2.TabStop = False
        Me.GroupBox2.Text = "Mesh"
        '
        'CheckBox5
        '
        Me.CheckBox5.Location = New
System.Drawing.Point(96, 24)
        Me.CheckBox5.Name = "CheckBox5"
        Me.CheckBox5.Size = New System.Drawing.Size(64, 24)
        Me.CheckBox5.TabIndex = 2
        Me.CheckBox5.Text = "Row"
        '
        'CheckBox4
        '
        Me.CheckBox4.Location = New
System.Drawing.Point(16, 24)
        Me.CheckBox4.Name = "CheckBox4"
        Me.CheckBox4.Size = New System.Drawing.Size(64, 24)
```

```
        Me.CheckBox4.TabIndex = 1
        Me.CheckBox4.Text = "Column"
        '
        'StatusBar1
        '
        Me.StatusBar1.Location = New
System.Drawing.Point(0, 523)
        Me.StatusBar1.Name = "StatusBar1"
        Me.StatusBar1.Panels.AddRange(New
System.Windows.Forms.StatusBarPanel() {Me.StatusBarPanel1,
Me.StatusBarPanel2, Me.StatusBarPanel3})
        Me.StatusBar1.ShowPanels = True
        Me.StatusBar1.Size = New System.Drawing.Size(792,
22)
        Me.StatusBar1.TabIndex = 14
        '
        'StatusBarPanel1
        '
        Me.StatusBarPanel1.AutoSize =
System.Windows.Forms.StatusBarPanelAutoSize.Spring
        Me.StatusBarPanel1.Width = 258
        '
        'StatusBarPanel2
        '
```

```
        Me.StatusBarPanel2.AutoSize =
System.Windows.Forms.StatusBarPanelAutoSize.Spring
        Me.StatusBarPanel2.Width = 258
        '
        'StatusBarPanel3
        '
        Me.StatusBarPanel3.AutoSize =
System.Windows.Forms.StatusBarPanelAutoSize.Spring
        Me.StatusBarPanel3.Width = 258
        '
        'PictureBox1
        '
        Me.PictureBox1.Image =
CType(resources.GetObject("PictureBox1.Image"),
System.Drawing.Image)
        Me.PictureBox1.Location = New
System.Drawing.Point(8, 32)
        Me.PictureBox1.Name = "PictureBox1"
        Me.PictureBox1.Size = New System.Drawing.Size(208,
50)
        Me.PictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage
        Me.PictureBox1.TabIndex = 15
        Me.PictureBox1.TabStop = False
```

```
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5,
13)
        Me.BackColor = System.Drawing.SystemColors.Control
        Me.ClientSize = New System.Drawing.Size(792, 545)
        Me.Controls.Add(Me.PictureBox1)
        Me.Controls.Add(Me.StatusBar1)
        Me.Controls.Add(Me.GroupBox2)
        Me.Controls.Add(Me.GroupBox1)
        Me.Controls.Add(Me.c1Chart3D1)
        Me.Menu = Me.mainMenu1
        Me.Name = "Form1"
        Me.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen
        Me.Text = "Interactive 3D Surface Plotting Software
(SPS)"
        CType(Me.c1Chart3D1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.numericUpDown2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.numericUpDown3,
System.ComponentModel.ISupportInitialize).EndInit()
```

```
        CType(Me.numericUpDown1,
System.ComponentModel.ISupportInitialize).EndInit()
        Me.GroupBox1.ResumeLayout(False)
        Me.GroupBox2.ResumeLayout(False)
        CType(Me.StatusBarPanel1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.StatusBarPanel2,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.StatusBarPanel3,
System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)

    End Sub 'InitializeComponent

#End Region
   '/ <summary>
   '/ The main entry point for the application.
   '/ </summary>
   <STAThread()>  _
   Shared    Sub Main()
      Application.Run(New Form1())
   End Sub 'Main
```

```
   Private Sub OnValueChanged1(sender As Object, e As
System.EventArgs) Handles numericUpDown1.ValueChanged
      If bUpdate Then
         c1Chart3D1.ChartArea.View.RotationX =
CInt(numericUpDown1.Value)
      End If
   End Sub 'OnValueChanged1

   Private Sub OnValueChanged2(sender As Object, e As
System.EventArgs) Handles numericUpDown2.ValueChanged
      If bUpdate Then
         c1Chart3D1.ChartArea.View.RotationY =
CInt(numericUpDown2.Value)
      End If
   End Sub 'OnValueChanged2

   Private Sub OnValueChanged3(sender As Object, e As
System.EventArgs) Handles numericUpDown3.ValueChanged
      If bUpdate Then
         c1Chart3D1.ChartArea.View.RotationZ =
CInt(numericUpDown3.Value)
      End If
   End Sub 'OnValueChanged3
```

```
   Sub createTestData()
      Dim i, j As Integer
      Dim x, y As Single
      ' init point dataset
      setPoint = New Chart3DDataSetPoint()

      Dim ix As Integer
      For ix = 0 To 2
         Dim iy As Integer
         For iy = 0 To 2
            Dim iz As Integer
            For iz = 0 To 2
               pointData((ix + 3 * iy + 9 * iz)).X = ix
               pointData((ix + 3 * iy + 9 * iz)).Y = iy
               pointData((ix + 3 * iy + 9 * iz)).Z = iz

               pointData1((ix + 3 * iy + 9 * iz)).X = 3 + 2
* ix
               pointData1((ix + 3 * iy + 9 * iz)).Y = 2 *
iy
               pointData1((ix + 3 * iy + 9 * iz)).Z = 2 *
iz
            Next iz
         Next iy
```

```
        Next ix
        setPoint.AddSeries(pointData)
        setPoint.AddSeries(pointData1)

        'float[,] gridData = new float[ 11, 11];
        Dim gridData As Single(,) =
CType(Array.CreateInstance(GetType(Single), 11, 11),
Single(,))
        For i = 0 To (gridData.GetLength(0)) - 1
           For j = 0 To (gridData.GetLength(1)) - 1
              x = - 5 + i
              y = - 5 + j
              gridData(i, j) = 0.1F * x * x - 0.4F * y * y
           Next j
        Next i
        setGrid = New Chart3DDataSetGrid(- 5, - 5, 1, 1,
gridData)

        ' contour data for 4d chart
        contourData =
CType(Array.CreateInstance(GetType(Double),
setGrid.ColumnCount, setGrid.RowCount), Double(,))
        For i = 0 To setGrid.ColumnCount - 1
           For j = 0 To setGrid.RowCount - 1
```

```
            contourData(i, j) = i + j
         Next j
      Next i
      ' init irregular grid dataset
      Dim xdata As Single() =  {1, 2, 6, 8, 9, 13, 15}
      Dim ydata As Single() =  {1, 2, 4, 8, 11, 12, 15}
      Dim vals As Single(,) =
CType(Array.CreateInstance(GetType(Single), xdata.Length,
ydata.Length), Single(,))

      setIrGrid = New Chart3DDataSetIrGrid(xdata, ydata,
vals)

      If (True) Then
         x = CSng(setIrGrid.ColumnOrigin)
         For i = 0 To setIrGrid.ColumnCount - 1
            x += CSng(setIrGrid.ColumnDeltaArray(i))
            y = CSng(setIrGrid.RowOrigin)
            For j = 0 To setIrGrid.RowCount - 1
               y += CSng(setIrGrid.RowDeltaArray(j))
               Dim val As Single = 0.1F * x * x - 0.4F * y
* y
               setIrGrid.SetValue(i, j, val)
            Next j
```

```
            Next i
        End If
        ' starting dataset
        c1Chart3D1.ChartGroups(0).ChartData.Set = setGrid

        ' adding labels to chart
        Dim labelC As Chart3DLabel =
c1Chart3D1.ChartLabels.LabelsCollection.AddNewLabel()
        labelC.Text = "Label(Coordinate)"
        labelC.LabelCompass = LabelCompassEnum.SouthEast
        labelC.AttachMethodData.X = 20
        labelC.AttachMethodData.Y = 20
        labelC.Visible = True

        Dim labelD As Chart3DLabel =
c1Chart3D1.ChartLabels.LabelsCollection.AddNewLabel()
        labelD.Text = "Label(DataCoordinate)"
        labelD.LabelCompass = LabelCompassEnum.SouthEast
        labelD.Offset = 50
        labelD.Connected = True
        labelD.AttachMethod = AttachMethodEnum.DataCoordinate
        labelD.AttachMethodData.X = 0
        labelD.AttachMethodData.Y = 0
        labelD.AttachMethodData.Z = 0
```

```
        labelD.Visible = True

        Dim labelI As Chart3DLabel =
c1Chart3D1.ChartLabels.LabelsCollection.AddNewLabel()
        labelI.Text = "Label(DataIndex)"
        labelI.LabelCompass = LabelCompassEnum.SouthWest
        labelI.Offset = 25
        labelI.Connected = True
        labelI.AttachMethod = AttachMethodEnum.DataIndex
        labelI.AttachMethodData.SeriesIndex = 2
        labelI.AttachMethodData.PointIndex = 4
        labelI.Visible = True

        ' adding data labels
        c1Chart3D1.DefaultLabelStyle.BackColor =
Color.MistyRose
        c1Chart3D1.DefaultLabelStyle.Border.BorderStyle =
BorderStyleEnum.Solid
        For i = 0 To setGrid.RowCount - 1
            Dim lab As Chart3DDataLabel =
c1Chart3D1.ChartGroups.RowLabels.AddNewLabel()
            lab.Index = i
            lab.Text = "Row " + i.ToString()
        Next i
```

```
        For j = 0 To setGrid.ColumnCount - 1
            Dim lab As Chart3DDataLabel =
c1Chart3D1.ChartGroups.ColumnLabels.AddNewLabel()
            lab.Index = j
            lab.Text = "Col " + j.ToString()
        Next j

        ' adding labels to axis
        Dim axis As Chart3DAxis
        For Each axis In  c1Chart3D1.ChartArea.Axes
            Dim alab1 As Chart3DAxisLabel =
axis.ValueLabels.AddNewLabel()
            alab1.Value = - 1
            alab1.Text = "minus one"

            Dim alab2 As Chart3DAxisLabel =
axis.ValueLabels.AddNewLabel()
            alab2.Value = + 1
            alab2.Text = "plus one"
        Next axis

        ' set bar colors
        c1Chart3D1.ChartGroups(0).Bar.SetBarColor(1, 1,
Color.Red)
```

```
      c1Chart3D1.ChartGroups(0).Bar.SetBarColor(1, 2,
Color.Red)
      c1Chart3D1.ChartGroups(0).Bar.SetBarColor(- 1, 3,
Color.Blue)
      c1Chart3D1.ChartGroups(0).Bar.SetBarColor(3, - 1,
Color.Green)
   End Sub 'createTestData


    Private Sub FormLoad(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Try

            Dim ApplicationPath As String
            ApplicationPath =
CurDir(Application.ExecutablePath) + "\"

            Dim sfg As String = ""
            sfg = ApplicationPath & "surface.dat"


c1Chart3D1.ChartGroups(0).ChartData.LoadDataFromFile(sfg)
            Me.StatusBarPanel1.Text = "surface.dat"
```

```
            'createTestData()
            AddHandler
c1Chart3D1.ChartArea.View.AfterRotate, AddressOf
RotateHandler
            'Me.propertyGrid1.Anchor = AnchorStyles.Top Or
AnchorStyles.Bottom Or AnchorStyles.Left
            Me.c1Chart3D1.Anchor = AnchorStyles.Top Or
AnchorStyles.Bottom Or AnchorStyles.Left Or
AnchorStyles.Right

            ' Setup Form1

            c1Chart3D1.ChartArea.View.BackColor =
System.Drawing.Color.DarkGray
            c1Chart3D1.ChartArea.Style.BackColor =
System.Drawing.Color.DarkGray

            c1Chart3D1.ChartArea.View.View3D =
View3DEnum.XY_2D_Pos
            '           c1Chart3D1.ChartArea.View.RotationX
= 25
            '          c1Chart3D1.ChartArea.View.RotationY
= 1
```

```
c1Chart3D1.ChartGroups(0).Surface.IsColumnMeshShowing =
False

c1Chart3D1.ChartGroups(0).Surface.IsRowMeshShowing = False

            c1Chart3D1.ChartGroups(0).Contour.IsZoned =
True

            Dim ZMax As Integer
            Dim Stp As Integer
            Dim i As Integer

            ZMax =
c1Chart3D1.ChartGroups(0).ChartData.Set.MaxZ
            Stp = 200 / ZMax

            c1Chart3D1.ChartGroups(0).Contour.NumLevels =
ZMax * 3

            c1Chart3D1.ChartArea.AxisZ.Min = ZMax * 5
```

```
            '
c1Chart3D1.ChartGroups(0).Contour.Levels.Item(0).Style.Fill
Color = _
            '
System.Drawing.ColorTranslator.FromWin32(RGB(0, 0, 0))


c1Chart3D1.ChartGroups(0).Contour.Levels.Item(i).Style.Fill
Color = System.Drawing.Color.Black

            For i = 1 To ZMax Step 1


c1Chart3D1.ChartGroups(0).Contour.Levels.Item(i).Style.Fill
Color = _

System.Drawing.ColorTranslator.FromWin32(RGB(i * Stp + 50,
i * Stp + 50, 0))

            Next

            c1Chart3D1.Refresh()
            Me.Refresh()
```

```
            c1Chart3D1.ChartArea.View.IsInteractive = True
            Me.menuItem18.Checked = True

            'Check if a parameter was passed
            Dim passFilename As String =
Microsoft.VisualBasic.Command()
            If passFilename <> "" Then

                sfg = ApplicationPath & passFilename
                c1Chart3D1.SaveImage(sfg, ImageFormat.Jpeg)
                Application.Exit()

            End If


            Me.checkBox3.Checked = True




        Catch ex As Exception

        End Try
```

```
    End Sub 'FormLoad
    'c1Chart3D1.DumpModel();

    Private Sub menuItem3_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem3.Click
        c1Chart3D1.ChartGroups(0).ChartData.Set = setGrid
    End Sub 'menuItem3_Click


    Private Sub menuItem5_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem5.Click
        c1Chart3D1.ChartGroups(0).ChartType =
Chart3DTypeEnum.Bar
    End Sub 'menuItem5_Click


    Private Sub menuItem6_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem6.Click
        c1Chart3D1.ChartGroups(0).ChartType =
Chart3DTypeEnum.Scatter
    End Sub 'menuItem6_Click
```

```
    Private Sub menuItem7_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem7.Click
        c1Chart3D1.ChartGroups(0).ChartType =
Chart3DTypeEnum.Surface
    End Sub 'menuItem7_Click


    Private Sub timer1_Tick(ByVal sender As Object, ByVal e
As System.EventArgs) Handles timer1.Tick
        If checkBox1.Checked Then
            c1Chart3D1.ChartArea.View.RotationX +=
angleIncrement
            If c1Chart3D1.ChartArea.View.RotationX >= 360
Then
                c1Chart3D1.ChartArea.View.RotationX = 0
            End If
            numericUpDown1.Value =
CDec(c1Chart3D1.ChartArea.View.RotationX)
        End If
        If checkBox2.Checked Then
            c1Chart3D1.ChartArea.View.RotationY +=
angleIncrement
```

```
            If c1Chart3D1.ChartArea.View.RotationY >= 360
Then
                c1Chart3D1.ChartArea.View.RotationY = 0
            End If
            numericUpDown2.Value =
CDec(c1Chart3D1.ChartArea.View.RotationY)
        End If
        If checkBox3.Checked Then
            c1Chart3D1.ChartArea.View.RotationZ +=
angleIncrement
            If c1Chart3D1.ChartArea.View.RotationZ >= 360
Then
                c1Chart3D1.ChartArea.View.RotationZ = 0
            End If
            numericUpDown3.Value =
CDec(c1Chart3D1.ChartArea.View.RotationZ)
        End If
    End Sub 'timer1_Tick


    Private Sub checkBox1_CheckedChanged(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
checkBox1.CheckedChanged, checkBox2.CheckedChanged,
checkBox3.CheckedChanged
```

```
        timer1.Enabled = checkBox1.Checked OrElse
checkBox2.Checked OrElse checkBox3.Checked
    End Sub 'checkBox1_CheckedChanged


    Sub setHoles(ByVal grset As Chart3DDataSetGrid, ByVal
enable As Boolean)
        Dim i, j As Integer
        If enable Then
            For i = 0 To grset.ColumnCount - 1
                For j = 0 To grset.RowCount - 1
                    Dim x As Single
                    Dim y As Single

                    If TypeOf grset Is Chart3DDataSetIrGrid
Then
                        Dim s As Chart3DDataSetIrGrid =
CType(grset, Chart3DDataSetIrGrid)
                        x = CSng(s.GetColumnValue(i))
                        y = CSng(s.GetRowValue(j))
                    Else
                        x = CSng(grset.MinX + i *
grset.RowDelta)
```

```
                        y = CSng(grset.MinY + j *
grset.ColumnDelta)
                    End If

                    Dim val As Single = 0.1F * x * x - 0.4F
* y * y
                    If i = j OrElse i = grset.RowCount - j
Then
                        grset.SetValue(i, j, val)
                    End If
                Next j
            Next i
        Else
            For i = 0 To grset.ColumnCount - 1
                For j = 0 To grset.RowCount - 1
                    ' holes
                    If i = j OrElse i = grset.RowCount - j
Then
                        grset.SetValue(i, j, grset.Hole)
                    End If
                Next j
            Next i
        End If
    End Sub 'setHoles
```

```
    Private Sub menuItem9_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem9.Click
        setHoles(setGrid, menuItem9.Checked)
        setHoles(CType(setIrGrid, Chart3DDataSetGrid),
menuItem9.Checked)
        menuItem9.Checked = Not menuItem9.Checked
    End Sub 'menuItem9_Click


    Private Sub menuItem10_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem10.Click
        If c1Chart3D1.ChartGroups(0).ChartData.ContourData
Is Nothing Then
            c1Chart3D1.ChartGroups(0).ChartData.ContourData
= contourData
            c1Chart3D1.ChartGroups(0).Contour.IsZoned =
True
        Else
            c1Chart3D1.ChartGroups(0).ChartData.ContourData
= Nothing
            c1Chart3D1.ChartGroups(0).Contour.IsZoned =
False
        End If
```

```
        c1Chart3D1.Refresh()

    End Sub 'menuItem10_Click

    Private Sub menuItem11_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem11.Click
        c1Chart3D1.ChartGroups(0).ChartData.Set =
Me.setIrGrid
    End Sub 'menuItem11_Click


    Private Sub menuItem13_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem13.Click
        If setPoint.SeriesCollection.Count = 1 Then
            setPoint.AddSeries(pointData1)
        End If
        c1Chart3D1.ChartGroups(0).ChartData.Set = setPoint
        c1Chart3D1.ChartGroups(0).ChartType =
Chart3DTypeEnum.Scatter
    End Sub 'menuItem13_Click
```

```
    Private Sub menuItem12_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem12.Click
        If setPoint.SeriesCollection.Count = 2 Then
            setPoint.RemoveSeries(1)
        End If
        c1Chart3D1.ChartGroups(0).ChartData.Set = setPoint
        c1Chart3D1.ChartGroups(0).ChartType =
Chart3DTypeEnum.Scatter
    End Sub 'menuItem12_Click


    Private Sub c1Chart3D1_MouseMove(ByVal sender As
Object, ByVal e As System.Windows.Forms.MouseEventArgs)
    End Sub 'c1Chart3D1_MouseMove

    'float x=0, y=0, z=0;
    'c1Chart3D1.ChartGroups[0].ChartData.CoordToDataCoord(
e.X, e.Y, ref x, ref y, ref z);
    'c1Chart3D1.ChartLabels[0].Text = String.Format( " {0},
{1}, {2}", x, y, z );

    Private Sub RotateHandler(ByVal sender As Object, ByVal
e As RotateEventArgs)
        bUpdate = False
```

```
        numericUpDown1.Value = CDec(e.RotationX / Math.PI *
180)
        numericUpDown2.Value = CDec(e.RotationY / Math.PI *
180)
        numericUpDown3.Value = CDec(e.RotationZ / Math.PI *
180)
        bUpdate = True
    End Sub 'RotateHandler


    Private Sub menuItem15_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem15.Click
        Dim ofd As New OpenFileDialog
        ofd.DefaultExt = ".dat"
        ofd.FileName = "surface"
        ofd.Filter = "SPS data files(*.dat)|*.dat|All
files(*.*)|*.*"

        If ofd.ShowDialog() = DialogResult.OK Then

c1Chart3D1.ChartGroups(0).ChartData.LoadDataFromFile(ofd.Fi
leName)
            Me.StatusBarPanel1.Text = ofd.FileName.ToString
```

```
            c1Chart3D1.ChartArea.View.BackColor =
System.Drawing.Color.DarkGray
            c1Chart3D1.ChartArea.Style.BackColor =
System.Drawing.Color.DarkGray

            c1Chart3D1.ChartArea.View.View3D =
View3DEnum.XY_2D_Pos
            '           c1Chart3D1.ChartArea.View.RotationX
= 25
            '          c1Chart3D1.ChartArea.View.RotationY
= 1

c1Chart3D1.ChartGroups(0).Surface.IsColumnMeshShowing =
False

c1Chart3D1.ChartGroups(0).Surface.IsRowMeshShowing = False

            c1Chart3D1.ChartGroups(0).Contour.IsZoned =
True

            Dim ZMax As Integer
            Dim Stp As Integer
            Dim i As Integer
```

```
            ZMax =
c1Chart3D1.ChartGroups(0).ChartData.Set.MaxZ
            Stp = 200 / ZMax

            c1Chart3D1.ChartGroups(0).Contour.NumLevels =
ZMax * 3

            c1Chart3D1.ChartArea.AxisZ.Min = ZMax * 5

            '
c1Chart3D1.ChartGroups(0).Contour.Levels.Item(0).Style.Fill
Color = _
            '
System.Drawing.ColorTranslator.FromWin32(RGB(0, 0, 0))


c1Chart3D1.ChartGroups(0).Contour.Levels.Item(i).Style.Fill
Color = System.Drawing.Color.Black

            For i = 1 To ZMax Step 1


c1Chart3D1.ChartGroups(0).Contour.Levels.Item(i).Style.Fill
Color = _
```

```
System.Drawing.ColorTranslator.FromWin32(RGB(i * Stp + 50,
i * Stp + 50, 0))

            Next

            c1Chart3D1.Refresh()
            Me.Refresh()



            'Dim ZMax As Integer
            'Dim Stp As Integer
            'Dim i As Integer

            'ZMax =
c1Chart3D1.ChartGroups(0).ChartData.Set.MaxZ
            'Stp = 200 / ZMax

            'c1Chart3D1.ChartGroups(0).Contour.NumLevels =
ZMax

            'c1Chart3D1.ChartArea.AxisZ.Min = ZMax * 5
```

```
            'For i = 0 To ZMax - 1 Step 1

            '
c1Chart3D1.ChartGroups(0).Contour.Levels.Item(i).Style.Fill
Color = _
            '
System.Drawing.ColorTranslator.FromWin32(RGB(i * Stp + 50,
i * Stp + 50, 0))

            'Next

            c1Chart3D1.Repaint = True
            c1Chart3D1.Refresh()
            Me.Refresh()



            Me.Refresh()
        End If
    End Sub 'menuItem15_Click

    Private Sub menuItem16_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem16.Click
        Dim sfd As New SaveFileDialog
```

```
        sfd.DefaultExt = "dat3xml"
        sfd.FileName = "doc1"
        sfd.Filter = "Chart3D.Net xml data
files(*.dat3xml)|*.dat3xml|C1Chart7 data
files(*.dat)|*.dat"

        If sfd.ShowDialog() = DialogResult.OK Then

c1Chart3D1.ChartGroups(0).ChartData.SaveDataToFile(sfd.File
Name)
        End If
    End Sub 'menuItem16_Click

    Private Sub menuItem18_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem18.Click
        c1Chart3D1.ChartArea.View.IsInteractive = Not
c1Chart3D1.ChartArea.View.IsInteractive
        menuItem18.Checked =
c1Chart3D1.ChartArea.View.IsInteractive
    End Sub 'menuItem18_Click


    Private Sub menuItem20_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem20.Click
```

```
        Me.Close()
    End Sub 'menuItem20_Click


    Private Sub menuItem21_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem21.Click
        Dim ofd As New OpenFileDialog
        ofd.DefaultExt = "chart3dxml"
        ofd.FileName = "doc1"
        ofd.Filter = "Chart3D xml
files(*.chart3dxml)|*.chart3dxml|All files (*.*)|*.*"

        If ofd.ShowDialog() = DialogResult.OK Then
            c1Chart3D1.LoadChartFromFile(ofd.FileName)
            Me.StatusBarPanel1.Text = ofd.FileName.ToString

            Dim ZMax As Integer
            Dim Stp As Integer
            Dim i As Integer

            ZMax =
c1Chart3D1.ChartGroups(0).ChartData.Set.MaxZ
            Stp = 200 / ZMax
```

```
            c1Chart3D1.ChartGroups(0).Contour.NumLevels =
ZMax

            c1Chart3D1.ChartArea.AxisZ.Min = ZMax * 5


            For i = 0 To ZMax Step 1


c1Chart3D1.ChartGroups(0).Contour.Levels.Item(i).Style.Fill
Color = _

System.Drawing.ColorTranslator.FromWin32(RGB(i * Stp + 50,
12, 0))

            Next

            c1Chart3D1.Refresh()
            Me.Refresh()

            Me.Refresh()
        End If
    End Sub 'menuItem21_Click
```

```
    Private Sub menuItem22_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem22.Click
        Dim sfd As New SaveFileDialog
        sfd.DefaultExt = "chart3dxml"
        sfd.FileName = "doc1"
        sfd.Filter = "Chart3D xml
files(*.chart3dxml)|*.chart3dxml|All files (*.*)|*.*"

        If sfd.ShowDialog() = DialogResult.OK Then
            c1Chart3D1.SaveChartToFile(sfd.FileName)
        End If
    End Sub 'menuItem22_Click

    Private Sub menuItem24_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem24.Click
        Dim pd As New PrintDocument
        AddHandler pd.PrintPage, AddressOf Me.pd_PrintPage

        Dim preview As New PrintPreviewDialog
        preview.Document = pd
        preview.ShowDialog()
    End Sub 'menuItem24_Click
```

```
    Private Sub pd_PrintPage(ByVal sender As Object, ByVal
ev As PrintPageEventArgs)
        Me.c1Chart3D1.Draw(ev.Graphics, ev.MarginBounds)
    End Sub 'pd_PrintPage


    Private Sub menuItem26_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem26.Click
        c1Chart3D1.ShowWizard()
    End Sub 'menuItem26_Click


    Private Sub menuItem27_Click_1(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem27.Click
        c1Chart3D1.ShowProperties()
    End Sub 'menuItem27_Click_1


    Private Sub menuItem28_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuItem28.Click
        Dim sfg As New SaveFileDialog

        sfg.Filter = "Metafiles (*.emf)|*.emf|" + "Bmp
files (*.bmp)|*.bmp|" + "Gif files (*.gif)|*.gif|" + "Jpeg
```

```
files (*.jpg;*.jpeg)|*.jpg;*.jpeg|" + "Png files
(*.png)|*.png"
        sfg.DefaultExt = "jpg"
        sfg.FileName = "image1"
        sfg.OverwritePrompt = True
        sfg.CheckPathExists = True
        sfg.RestoreDirectory = False
        sfg.ValidateNames = True

        If sfg.ShowDialog() = DialogResult.OK Then
            Dim imgfmt As ImageFormat = Nothing
            Select Case Path.GetExtension(sfg.FileName)
                Case ".emf"
                    imgfmt = ImageFormat.Emf

                Case ".bmp"
                    imgfmt = ImageFormat.Bmp

                Case ".gif"
                    imgfmt = ImageFormat.Gif

                Case ".jpeg", ".jpg"
                    imgfmt = ImageFormat.Jpeg
```

```
                Case ".png"
                    imgfmt = ImageFormat.Png

                Case Else
                    Return
            End Select
            c1Chart3D1.SaveImage(sfg.FileName, imgfmt)
        End If
    End Sub 'menuItem28_Click


    Private Sub menuHelpAbout_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles menuHelpAbout.Click
        MessageBox.Show(Me, [Text], "About",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub 'menuHelpAbout_Click


    Private Sub GenerateZones()

    End Sub

    Private Sub Button1_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
```

```
    End Sub

    Private Sub CheckBox4_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CheckBox4.CheckedChanged
        If Me.CheckBox4.Checked = True Then

c1Chart3D1.ChartGroups(0).Surface.IsColumnMeshShowing =
True
        Else

c1Chart3D1.ChartGroups(0).Surface.IsColumnMeshShowing =
False
        End If
        c1Chart3D1.Refresh()
    End Sub

    Private Sub CheckBox5_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CheckBox5.CheckedChanged
        If Me.CheckBox4.Checked = True Then

c1Chart3D1.ChartGroups(0).Surface.IsRowMeshShowing = True
```

```
        Else

c1Chart3D1.ChartGroups(0).Surface.IsRowMeshShowing = False
        End If
        c1Chart3D1.Refresh()
    End Sub

    Private Sub Button1_Click_1(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Button1.Click

        c1Chart3D1.ChartArea.View.BackColor =
System.Drawing.Color.DarkGray
        c1Chart3D1.ChartArea.Style.BackColor =
System.Drawing.Color.DarkGray

        c1Chart3D1.ChartArea.View.View3D =
View3DEnum.XY_2D_Pos
        'c1Chart3D1.ChartArea.View.RotationX = 30
        'c1Chart3D1.ChartArea.View.RotationY = 1

c1Chart3D1.ChartGroups(0).Surface.IsColumnMeshShowing =
False
```

```
        c1Chart3D1.ChartGroups(0).Surface.IsRowMeshShowing
= False



        c1Chart3D1.Refresh()

    End Sub
```

APPENDIX B

CODE FOR KINETIC MONTE CARLO SIMULATION

```
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <ctime>
# include <stdio.h>
# include <stdlib.h>

# include "growth.h"
# include "sobol.h"

using namespace std;

int main(void)
{
            ofstream GrowthSite("GrowthSites.txt", ios::out);
            ofstream FinalReport("GrowthReport.txt", ios::out);
            ofstream W("Roughness.txt", ios::app);
            GrowthSite <<"x,y,z"<<endl;
      float lattice = 0.543;  // lattice constant for Silicon in
Nanometer
      float data[11];
      int **history;
      int leng,wid,hei; //length, width, of template window; hei is
time
      int x,y,z;
      int x_max, y_max, z_max;
      int layers;
```

```
	int inital_out = 1;		//Variable used to determine if
inialized array
					// is to outputed for review
	int plot_out = 0;
	int i,j,k,a,b,c;	//index variables

	int ***volume;			//3D crystal structure
	float pdf[3];			//probability density functions for
transition
	int coord[3];			//x, y coordinate on surface
	int index[3];			//size of x,y,z inicicies of SUT
	int nano[3];			//size of x,y,z indicies of nano
window
	static int sobol_i = 0; // index used to increament sobol
generator
	int select = -1;	//used to determine which process was
selected (Adsopr,Desorp,Diff)
	float KMC_percentage = 0; // percentage of surface that will be
sampled each period
	int KMC_period = 0;		// number of KMC trials in each
time step
	int time_period = 0;	// total time period to be modeled
	int valid_site = 0;
	double pressure;
	double temprature;
	double area;
	double mass = 28.0855;
```

```
	int adjacent = 0;
	int side[4];
	int layer_limit;		//total number of layers created in 3D
space(size of array)
	float deviations;
	int defect;
	double clock = 0;
	int transistions[3] = {0,0,0};
	double adsorp_trans = 0;
	double desorp_trans = 0;
	double diff_trans = 0;
	float temp;

	double *field;
	double params[4];
	float pattern[2];

	double rough;



/*************************************************************
Start up menu:
Ask user for simulation paramters:
	window dimensions
	KMC period
	Time period
	Temprature
```

```
**********************************************************/
      setup(data);
      optic_setup( params,pattern);

      field = Fresnel(pattern,params);

      KMC_percentage = data[5];
      time_period = data[6];
      deviations = data[9];

      // Conversion of length into array indecies of the nano window
      // takes int and stores into float for conversion
      leng = data[1];
      temp = data[1];
      x = ((temp)/(lattice/2));

      wid = data[0];
      temp = data[0];
      y = ((temp)/(lattice/2));

      hei = data[2];
      temp = data[2];
      z = temp;

      //Setting max indices for SUT
      x_max = x * 10;
```

```
        y_max = y * 10;
        z_max = data[8];

        inital_out = data[7];
        temprature = data[3];
        pressure = data[4];
        area = data[0] * data[1];

        layer_limit = data[8];

        sobol_i = get_seed();
        //Defines the dimensions of the Surface Under Test (SUT)
        index[0] = x_max;
        index[1] = y_max;
        index[2] = z_max;

        nano[0] = x;
        nano[1] = y;
        nano[2] = z;
timestamp();
cout << "*****************************************"<<endl;
cout <<"Initial indicies for nano window and SUT" << endl;
cout <<"nano window"<<endl;
cout << "x = "<<x <<", y = "<<y<<", z ="<<z<<endl;
cout <<"SUT"<<endl;
cout <<"x = "<<x_max<<", y = "<<y_max<<", z = "<<z_max<<endl;
```

```
KMC_period = (KMC_percentage) * (x/2) * (y/2);
/*********************************************************

Create and initalize an array to hold site selection history


*********************************************************/

                  history = array2Dcreate(KMC_period,2);
                  array2Dinitial(history,KMC_period,2);

/*********************************************************
Create 3D growth surface


*********************************************************/
      cout << "....Creating Growth Surface.....\n";

      volume = array3Dcreate(x_max,y_max,z_max);




/*********************************************************
Initalize Growth Surface:
site set to -1 if no growth is possible at site (not a bonding site)
site set to 0 if growth can occur
```

```
**********************************************************/

      cout << ".......Initalizing Growth Surface.....\n";
      defect = 0;
      cout <<"Do you want to add an island on inital surface (1:Yes,
0:No)";
      cin >> defect;
      cout<<endl;

      array3Dinitial(volume,x_max,y_max,z_max,defect);


/*****************************************************************
Outputs intialized growth surface for review, can be disabled


**************************************************************/

      if(inital_out == 1)
      {
            cout << ".......... Outputing Initaialized Growth Surface
......\n";
            smooth( volume, index);
            system("SPS.exe");
```

```
	}

/*************************************************************
Begin KMC simulation process

*************************************************************/
	cout << "...Starting Simulation ...\n";

	//total number of bonding sites is wid/2 * leng/2
	//volume[] potrays the actual crystal structure with spacing
associated
	//the fcc structure
	KMC_period = (KMC_percentage) * (x/2) * (y/2);

	cout << "\n\n... aproximatley "<<KMC_period<<" out of "<<(x/2) *
(y/2)<<" sites will be examined per KMC cycle\n\n";
	a = 0;
	for(int u = 0; u < z; u++)
	{
		cout << "Entering layer #"<< u << endl;
		GrowthSite << "Entering layer #"<< u << endl;
		array2Dinitial(history,KMC_period,2);

		for(i = 0; i < KMC_period;i++)
		{
```

```
                    //Calls ValidSite()which updates coord, with a new
site x, y coord on the SUT
                                //that falls within valid range crierion

      ValidSite(&sobol_i,x_max,y_max,z_max,nano,coord,volume,deviations
,history, KMC_period);




                    //cout<<"Valid Site found on KMC itteration #"<<i<<",
"<<volume[coord[0]][coord[1]][coord[2]]<<endl;
              //*****************************************
              //PDF caluclations based upon if site is occupied
              //vacant
              //
              //*****************************************


                    //Site is vacant
                    if((coord[2] == 0) ||(coord[2] == 1))
                    {     adjacent = -1;


      pdfAdsorp(pdf,coord,temprature,mass,pressure,area,index,deviation
s,adjacent,volume,field,pattern);
                          //cout<<"pdf is: "<<pdf[0]<<", "<<pdf[1]<<",
"<<pdf[2]<<endl;
                          pdfVacantSelect(pdf, &select,transistions);
```

```
                        //cout<<"pdf calculated for vacant on KMC
itteration #"<<i<<endl;

                  }

                  //Site is occupied
                  if(coord[2] > 1)
                  {
                        neighbors(volume,coord,index,&adjacent);
                        //cout << "Done with neighbors"<<endl;

      pdfAdsorp(pdf,coord,temprature,mass,pressure,area,index,deviation
s,adjacent,volume,field,pattern);
                        //cout << "Done with Adsorp" <<endl;
                        pdfDesorp(pdf,temprature,adjacent,area);
                        //cout << "Done with Desorp" <<endl;
                        pdfDiff(pdf,temprature,adjacent,area);
                        //cout << "Done with Diff" <<endl;
                        //cout<<"pdf is: "<<pdf[0]<<", "<<pdf[1]<<",
"<<pdf[2]<<endl;
                        pdfSelection(pdf,&select,transistions);
                        //cout<<"pdf calculated for occupied on KMC
itteration #"<<i<<endl;
                  }

                  GrowthSite
<<coord[0]<<"\t"<<coord[1]<<"\t"<<coord[2]<<"\t"<<select<<endl;
```

```
                history[i][0] = coord[0];
                history[i][1] = coord[1];

                //Updating selected site based upon pdf calculation
and RNGs
                siteUpdate(volume, coord, &select, index,adjacent);

                //cout<<"Site updated "<<coord[2]<< ", on KMC
itteration #"<<i<<" "<<endl;
            }
            adsorp_trans = transistions[0] + adsorp_trans;
            desorp_trans = transistions[1] + desorp_trans;
            diff_trans = transistions[2] + diff_trans;

            timeStep(&clock,pdf,&area, transistions);
            if(data[10] == 1)
            //snapshot( volume, index,u);
            transistions[0] = 0;
            transistions[1] = 0;
            transistions[2] = 0;
            rough = roughness(volume,index,nano,deviations);
            W << rough <<endl;
            //cout << "Roghness = " << rough <<endl;
      /*    if((u == 18) || (u == 32))
            {
                smooth( volume, index);
                system("SPS.exe");
```

```
            }*/
        }



        cout << "......Simulation Complete ..............\n";
        FinalReport <<"......Simulation Complete ..............\n";
      FinalReport <<"Growth Temp = " <<temprature<<" K\n"<<endl;
        FinalReport <<"Growth Pressure = "<<pressure<<" Torr\n"<<endl;
        FinalReport <<"Standard Deviations  = "<<deviations<<" \n"<<endl;
        FinalReport <<"---------------------------------"<<endl;
      FinalReport <<"Nano window diminsions were as follows"<<endl;
        FinalReport <<"X = "<<wid<<" nm, Y = "<<leng<<" nm, Z = "<<hei<<"
nm"<<endl;
        FinalReport <<"Window lies between " << (x_max -x)/2 <<" to " <<
((x_max - x)/2) + x <<endl;
        FinalReport <<"Window lies between " << (y_max -y)/2 <<" to " <<
((y_max - y)/2) + y <<endl;
        FinalReport << "\n\n... aproximatley "<<KMC_period<<" out of
"<<(x/2) * (y/2)<<" sites will be examined per KMC cycle\n\n"<<endl;


        FinalReport <<"----------------------------"<<endl;
        FinalReport <<"Growth took "<<clock <<" sec"<<endl;
        FinalReport <<"      or    "<<clock/60 <<" min\n"<<endl;


        FinalReport <<"----------------------------"<<endl;
```

```
	FinalReport <<adsorp_trans<<" Adsorption Transistions Occured,
"<< (adsorp_trans /(adsorp_trans + desorp_trans + diff_trans) ) * 100
<<"% of all Transistions"<<endl;
	FinalReport <<desorp_trans<<" Desorption Transistions Occured "<<
(desorp_trans /(adsorp_trans + desorp_trans + diff_trans) ) * 100 <<"%
of all Transistions"<<endl;
	FinalReport <<diff_trans<<" Surface Diffusion Transistions
Occured "<<(diff_trans /(adsorp_trans + desorp_trans + diff_trans) ) *
100 <<"% of all Transistions"<<endl;
	FinalReport.close();
	timestamp();
	//cout<<"Do you want to print output to a CSV file for plotting?
(1 = Yes, 0 = No): ";
	//cin>>plot_out;
	//if(plot_out == 1)
	//{
		cout<<"....Converting and saving into CSV file....."<<endl;
		//PlotCSV( volume, index);
		smooth( volume, index);
		free(volume);
		//array3Dout( volume, index[0], index[1], index[2]);
	//}

	//wait();
		if(data[10] == 1)
	//		system("Slideshow.exe");
		//else
```

```
            {
                  system("conversion.exe");
                  system("SPS.exe");
            }
      //**********Print out a growth
Summary**************************



return 0;
}

# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <fstream>
# include <ctime>
# include <cmath>
#include <math.h>

#define ACC 40.0        //Make larger to increase accuracy.
#define BIGNO 1.0e10
#define BIGNI 1.0e-10
#define Bessel_max_order 20
#define PI 3.1415926535897932
const float pi = 3.14159265;
```

```
using namespace std;
//
#define oops(s) { perror((s)); exit(EXIT_FAILURE); }
#define MALLOC(s,t) if(((s) = malloc(t)) == NULL) { oops("error:
malloc() "); }
#define INCREMENT 10
//

float bessj0(float );
float bessj1(float );
float bessj(int , float );
void nrerror(char *);
double GeoField(double, double, double);
double DifField(double, double);
void FieldNormal(double *,int);
double* Fresnel(float *, double*);
void array2Dinitialgeo( double **, int , int );
double getField(int*, int*, double *, float *);
double** array2DcreateDouble(int ,int );
double      roughness(int ***,int *, int*,int);
int topofstack(int ,int ,int , int ***);

void setup(double *,float *);
void optic_setup(double *);
void intArrayClear( int*, int);
void array3Dintial( int***, int, int, int);
void array2Dinitial( int **, int, int,int);
```

```
void array3Dout(int***, int,int,int);
void PlotCSV( int ***, int *);
void wait(void);
void pdfAdsorp(float *, int *,int, double , double , double, int *,
float,int,int***, double *, float *);
void pdfDesorp(float *,int,int,double);
void pdfDiff(float *,int,int,double);
void neighbors(int ***,int *,int *,int *);
void smooth(int ***,int *);
void snapshot(int ***,int *,int);


int*** array3Dcreate(int,int,int);
int** array2Dcreate(int ,int );

void optic_setup(double *data,float *pattern)
{

      cout << "\n";
      cout << "These parameters are required for Geometirc \n";
      cout << "   affect to be accounted for \n";
/*    cout << "         Please enter the following parameters: \n";
      cout << "         APERTRUE RADIUS:  ";
      cin >> data[0];
      cout << "\n";
      cout << "         DISTANCE FROM SOURCE TO APERTRUE:  ";
      cin >> data[1];
```

```
        cout << "\n";
        cout << "         DISTANCE FROM APERTRUE TO SUBSTRATE :  ";
        cin >> data[2];
        cout <<"\n";
        cout << "       WAVELENGTH : ";
        cin   >> data[3];
        cout <<"\n";
        cout << "         Length of pattern  ";
        cin >> pattern[0];
        cout << "\n";
        cout << "         pattern resolution:  ";
        cin >> pattern[1];
        cout << "\n";
        */
        data[0] = 10e-3;
        data[1] = 10e-2;
        data[2] = 200e-9;
        data[3] = 500e-6;
        pattern[0] = 20e-6;
        pattern[1] = 0.005e-6;

}
void setup(float *data)
{

        cout << "\nKinetic Monte Carlo Simulation for Epitaxial\n";
        cout << "Grown Nanostructures.\n";
```

```
      cout << "\n";
      cout << "Model will assume an fcc Si lattice structure for
simulation";
      cout << "   Please enter the following parameters: \n";
      cout << "         Length of Window (nm):  ";
      cin >> data[0];
      cout << "\n";
      cout << "         Width of Window (nm):  ";
      cin >> data[1];
      cout << "\n";
      cout << "         Monte Carlo Simulated Growth Cycles :  ";
      cin >> data[2];
      cout << "       Growth Temprature : ";
      cin   >> data[3];
      cout <<"\n";
      cout << "       Growth Pressure : ";
      cin >> data[4];
      cout <<"\n";
      cout << "       KMC Coverage Percentage : ";
      cin >> data[5];
      cout <<"\n";/*
      cout << "       Time Period : ";
      cin >> data[6];
      cout <<"\n";
      cout << "       Would you like to review initalized surface [1 =
yes, 0 = no] : ";
      cin >> data[7];*/
```

```
	cout <<"\n";
	cout << "	How many layers do you want to be created in 3D
space"<<endl;
	cout << "		(process limitation) : ";
	cin >> data[8];
	cout <<"\n";/*
	cout << "	How many standard deviations : ";
	cin >> data[9];
	cout <<"\n";
	cout << "	Do you want snap shots of the surface [1 = yes, 0
= no]: ";
	cin >> data[10];*/
/*
	data[0] = 20;
	data[1] = 20;
	data[2] = 35;
	data[3] = 500;
	data[4] = 7.5e-5;
	data[5] = 1;*/
	data[6] = 1;
	data[7] = 0;
	//data[8] = 300;
	data[9] = 9;
	data[10] =0;

}
```

```
void intArrayClear(int *ptr, int length)
{
	int i;
	for(i = 0; i<length; i++)
	{
		ptr[i] = 0;
		//cout <<" ptr[" << i << "] = " << ptr[i] << "\n";
	}
}


void array3Dinitial( int ***Vol, int x, int y, int z,int defect)
{

	int i,j,k;
	int i_mod, j_mod, k_mod;
	int barrier = 10;
	if(defect == 1)
	{
		cout << "What height do you want for island";
		cin >> defect;
		cout <<"\n"<<endl;
	}
	for(i = 0; i < x; i++)
	{
		for(j = 0; j < y; j++)
		{
```

```
                for(k = 0; k < z; k++)
                {
                        i_mod = i%2;
                        j_mod = j%2;
                        k_mod = k%2;

                        if( k_mod == 0)
                        {
                                if( (i_mod) == (j_mod)  )
                                {
                                        if (defect ==0)
                                                Vol[i][j][k] = 0;
                                        else
                                        {
                                                if((k <= barrier) && (i >544)
&& (i < 584)
                                                        && (j >544) && (j <
584))
                                                {
                                                        Vol[i][j][k] = 1;
                                                        //cout<<"got here on x
= "<<i<<endl;
                                                }
                                                else
                                                        Vol[i][j][k] = 0;
                                        }
```

```
                        }
                        else
                        {
                              Vol[i][j][k] = -1;


                        }
                  }
                  else
                  {
                        if( (i_mod) != (j_mod) )
                        {
                              if (defect ==0)
                                    Vol[i][j][k] = 0;
                              else
                              {
                                    if((k <= barrier) && (i >544)
&& (i < 584)
                                          && (j >544) && (j <
584))
                                    {
                                          Vol[i][j][k] = 1;
                                          //cout<<"got here on x
= "<<i<<endl;
                                    }
                                    else
                                          Vol[i][j][k] = 0;
                              }
```

```
                        }

                        else
                        {
                              Vol[i][j][k] = -1;



                        }
                  }
            }
      }
      if((i % 16) == 0)
      {
            //barrier++;
            //cout << "x = " << i << endl;
            //cout << "barrier = "<< barrier <<endl;
      }
   }

}
void array2Dinitial( int **Vol, int x, int y)
{

   for(int i = 0; i < x; i++)
   {
      for(int j = 0; j < y; j++)
      {
```

```
                    Vol[i][j] = 0;

            }
      }
}
void array3Dout( int ***Vol, int x, int y, int z)
{
      ofstream outData("3DReview.csv", ios::out);
      outData <<  "x , y , z ,Site " << endl;
      int i,j,k;

      for(i = 0; i < x; i++)
      {
            for(j = 0; j < y; j++)
            {
                  for(k = 0; k < z; k++)
                  {
                        outData << i << "," << j << "," << k << "," <<
Vol[i][j][k] << "\n";
                  }
            }
      }


}

void PlotCSV( int ***Vol, int *index)
```

```
{
	ofstream outData("3DPlot.dat", ios::out);
	//outData <<  "x , y , Site " << endl;
	int i,j,k;
	int x_max, y_max, z_max;
	int high = 0;

	x_max = index[0];
	y_max = index[1];
	z_max = index[2];
	outData <<"GRID\t"<<x_max<<"\t"<<y_max<<endl;
	outData <<"-5\t1\t1\t0\t0"<<endl;
	/*for(i = 0; i < y_max; i++)
	{
		outData << i << ",";
	}
		outData <<"\n";
*/
	for(i = 0; i < x_max; i++)
	{
		//outData << i;
		for(j = 0; j < y_max; j++)
		{

			for(k = 0; k < z_max; k++)
			{
				if(Vol[i][j][k] == 1)
```

```
                        {
                              high = k;
                              //cout<<"For
Vol["<<i<<"]["<<j<<"]["<<k<<"] k is: "<<k<<endl;
                        }
                        if(k == (z_max - 1))
                        {
                              outData <<  "\t" << high ;
                              high = 0;
                        }
                  }

            }
            outData <<endl;
      }



}
int*** array3Dcreate(int x,int y,int z)
{
            int i,j,k;

            int*** array3D = new int**[x];
            for (i = 0; i < x; i++)
            {
            array3D[i] = new int*[y];
                  for (j = 0; j < y; j++)
```

```
                {
                        array3D[i][j] = new int[z];
                }
        }
        return array3D;
}

int** array2Dcreate(int x,int y)
{
        int i, j;
        int ** array2D = new int*[x];

        for(i = 0; i < x; i++)
        {
                array2D[i] = new int[y];
        }

        return array2D;
}
double** array2DcreateDouble(int x,int y)
{
        int i, j;
        double ** array2D = new double*[x];

        for(i = 0; i < x; i++)
        {
                array2D[i] = new double[y];
```

```
	}

	return array2D;
}
/*************************************************************************
*****************
Purpose:  pdfAdsorp calculates the proablitiy of an adsorption at a
paticular site on the
		surface based upon growth conditions

Var:  pdf: the probability density function array that hold the pd for
each possible
		transition that could occur pdf[0] = adsorption
		temp:temprature of molecules, a measure of kinetic energy
(kelvin)
		mass: Molecular Weight of the impinging specie
		pressure: pressur of reactor in Torr
		area: area where impinging flux is subject to (should be
given in nm^2)
		stick:  Sticking coefficent
		stdev: standard deveiation associated with shape of flux
comming through template
		stoch: size of the guassian distribution at the selected
site
		flux:  Impinging flux in units of atoms(molecules)/nm^2/s
*************************************************************************
*****************/
```

```
void pdfAdsorp(float *pdf, int *site, int temp, double mass, double
pressure, double area, int *max, float standard,int adjacent,int
***volume, double *field, float *pattern)
{
	ofstream outData("PDF_Adsorption.txt", ios::app);
	outData << "**********************"<<endl;

	//outData <<  "x , y , Site " << endl;
	//pdfAdsorp(pdf,coord,temprature,mass,pressure,area,index,deviati
ons,adjacent);
	/*
		pdf		probability function
		coord =		site under test
		temp		temprature
		mass		mass of flux atom
		pressure	pressure of chamber
		area		area of nano window
		max		maximum indicies of area under test
(Substrate indicies)
		standard	STD
		adjacent	number of neighboring atoms of the site under
test
```

```
      */
      double diffraction;
      double cons = 2 * pi;
      double stick = 1;
      double flux;
      double stdev_x,stdev_y ;
      double mean_x, mean_y;
      double zscore_x, zscore_y;
      double exp_x, exp_y;
      double stoch = 1;
      int x,y,z;
      int x_max, y_max, z_max;

      int x_nano, y_nano;           //nano window dimension
      int offset_x, offset_y; //offsets used to center the nano window
on SUT
      int x_low, x_limit;           //limits for valid x coordinate
      int y_low, y_limit;           //limits for valid y coordinate



    x = site[0];
      y = site[1];
      z = site[2];

      x_max = max[0];
      y_max = max[1];
```

```
	z_max = max[2];

	x_nano = x_max / 10;
	y_nano = y_max / 10;

	mean_x = x_max / 2;
	mean_y = y_max / 2;

	stdev_x = ( (x_nano / 2) + mean_x ) / standard; // (2 * Num of
STDEV that the nano window extends too)
	stdev_y = ( (y_nano / 2) + mean_y ) / standard;
	/*
	offset_x = (x_max - (x_nano))/2;
	offset_y = (y_max - (y_nano))/2;

	x_low = offset_x;
	x_limit = x_max - offset_x;

	y_low = offset_y;
	y_limit = y_max - offset_y;
	*/
	zscore_x = (x - mean_x) / ( stdev_x);
	if(zscore_x < 0 )
		zscore_x = zscore_x * -1;

	zscore_y = (y - mean_y) / ( stdev_y);
	if(zscore_y < 0 )
```

```
            zscore_y = zscore_y * -1;

      exp_x = exp(-0.5 * zscore_x * zscore_x);
      exp_y = exp(-0.5 * zscore_y * zscore_y);

                        //(( 1 / (stdev_x * stdev_y *  cons)) *
                        /// (( 1 / (stdev_x * stdev_y * cons)))

      //stoch =  pow(2.718281828459,( -(0.5) * ( ( (x - mean_x) * (x -
mean_x) / ( stdev_x * stdev_x ) ) + ( ( y - mean_y ) * ( y - mean_y ) /
( stdev_y * stdev_y ) ) ) ) );
      //stoch =  exp( -(0.5) * ( pow( (x - mean_x) / ( stdev_x) , 2) +
( ( pow( (y - mean_y) / ( stdev_y) , 2) ) ) ));
      stoch = exp_x * exp_y;

      outData << "stoch  = " << stoch <<endl;
      outData << "zscore x = "<<zscore_x<<", zscore y =
"<<zscore_y<<endl;
      outData << "exp x = "<<exp_x<<", exp y = "<<exp_y<<endl;
      outData <<" std_x = "<<stdev_x<<", std y = "<<stdev_y<<endl;
      outData <<"x mean = "<<mean_x <<", y mean = "<< mean_y <<endl;
      outData <<" x = "<<x<<", y = "<<y<<endl;

      diffraction = getField(site,max, field, pattern);

      outData <<"Field = "<<diffraction<<endl;
      flux = 3.51e8 * pressure / (sqrt(mass * temp));
```

```
	stick = 1;
	pdf[0] = stick * flux ; //* area* stoch * diffraction;

	outData<<"flux = "<<flux<<endl;
	outData<<"area = "<<area<<endl;
	outData<<"pdfAdsorp = "<<pdf[0]<<endl;
	outData.close();

	if( z > 1)
	{
		site[2] = site[2] - 2;
		neighbors(volume,site,max,&adjacent);

		if(adjacent == 0)
		{
			pdf[0] = pdf[0] * 0.001;
		}

		site[2] = site[2] + 2;
	}

}

/*
Purpose:  pdfDesorp determines the probability of desorption at a
surface site based
		upon current growth parametes
```

```
Var:        pdf:  output, where the transition rate of desorption is
saved pdf[1]
                  temp:  current temprature of atoms, measure of
kinetic energy (kelvin)
                  atoms:  Number of side neighbors to site under review
                  area: area of surface being simulated (nm^2)
                  desorp_energy:  Energy of desorption/adsorption (in
Joules) depth of potential
                                          well
                  bond_evergy:  Energy required to break bond ( used in
side neighbor)
*/
void pdfDesorp(float *pdf,int temp,int atoms,double area)
{
      double desorp_energy = 2.72e-19;//2.64e-18;
      double bond_energy = 7.59e-20;//3.69e-19;
      pdf[1] = ((1.38062e-23)/ (6.6262e-34)) * temp * exp(-
(desorp_energy + (atoms * bond_energy) ) / (1.38062e-23 * temp) )  *
area * 1e-14;
      //cout<<"Desorp = "<<pdf[1]<<endl;
      //cout<<"temp: "<<temp<<", neighbors: "<<atoms<<", area:
"<<area<<endl;
}

/*
```

```
Purpose:  pdfDiff determines the probability of diffusion at a surface
site based
              upon current growth parametes

Var:        pdf:  output, where the transition rate of diffusion is
saved pdf[2]
                  temp:  current temprature of atoms, measure of
kinetic energy (kelvin)
                  atoms:  Number of side neighbors to site under review
                  area: area of surface being simulated (nm^2)
                  desorp_energy:  Energy of diffusion (in Joules) depth
of potential
                                        well
                  bond_evergy:  Energy required to break bond ( used in
side neighbor)
*/
void pdfDiff(float *pdf,int temp,int atoms, double area)
{
      double diff_energy = 2.72e-20;//3.02e-19;
      double bond_energy = 7.59e-20;//3.69e-19;
      pdf[2] = ((1.38062e-23)/ (6.6262e-34)) * temp * exp(-(diff_energy
+ (atoms * bond_energy) ) / (1.38062e-23 * temp) )  * area * 1e-14;
      //cout<<"Diff = "<<pdf[2]<<endl;
      //cout<<"temp: "<<temp<<", neighbors: "<<atoms<<", area:
"<<area<<endl;
}
```

```
void neighbors(int ***Vol,int *site,int *index,int *count)
{
	//neighbors(volume,coord,index,&adjacent);
	int x,y,z;
	int x_max, y_max, z_max;

	*count = 0;

	x = site[0];
	y = site[1];
	z = site[2];

	x_max = index[0];
	y_max = index[1];
	z_max = index[2];

		if(Vol[x][y + 2][z] == 1)
		{
			*count = *count + 1;
		}

		if(Vol[x][y - 2][z] == 1)
		{
			*count = *count + 1;
```

```
		}

		if(Vol[x + 2][y][z] == 1)
		{
			*count = *count + 1;
		}

		if(Vol[x - 2][y][z] == 1)
		{
			*count = *count + 1;
		}

}

void smooth(int ***Vol,int *index)
{
	/*
		Vol = 3d array
		site = current site being evaluated
		index = max x,y,z indicies

	*/
	ofstream outData("surface.dat", ios::out);

    int i,j,k;
	int high;
	int x,y,z;
```

```
int x_max, y_max, z_max;
int bonds = 0;
int site[3];

int **output;

int i_mod, j_mod;



x_max = index[0];
y_max = index[1];
z_max = index[2];

output = array2Dcreate(x_max,y_max);


for(i = 0; i < x_max; i++)
{
      for(j = 0; j < y_max; j++)
      {
            for(k = 0; k < z_max; k++)
            {
                  if(Vol[i][j][k] == 1)
                  {
                        high = k;
                  }
```

```
                        if(k == (z_max - 1))
                        {
                              output[i][j] = high;
                              high = 0;
                        }
                  }
            }
      }

      bonds = 0;
/*
      for(i = 2; i < (x_max - 1); i++)
      {
            for(j = 2; j < (y_max - 1); j++)
            {
                  i_mod = i%2;
                  j_mod = j%2;

                  if( (i_mod) == (j_mod) )
                  {
                        if( (output[i][j] - output[i][j+1]) > 0)
                        {
                              bonds++;
                        }
                        if( (output[i][j] - output[i][j-1]) > 0)
                        {
                              bonds++;
```

```
                        }
                        if( (output[i][j] - output[i + 1][j]) > 0)
                        {
                              bonds++;
                        }
                        if( (output[i][j] - output[i - 1][j]) > 0)
                        {
                              bonds++;
                        }
                        if(output[i][j] == 0)
                        {
                              output[i][j] = output[i][j+1];
                        }
                        else if(bonds > 1)
                        {
                              output[i][j]++;
                        }
                  }

                  bonds = 0;
            }
      }
*/
      outData <<"GRID\t"<<x_max<<"\t"<<y_max<<endl;
      outData <<"-5\t1\t1\t0\t0"<<endl;
output[0][0]=0;
      for(i = 0; i < x_max; i++)
```

```
	{
		for(j = 0; j < y_max; j++)
		{
			outData  << output[i][j] <<  "\t";
		}
		outData <<endl;
	}
	free(output);
}

void snapshot(int ***Vol,int *index,int layer)
{
	/*
		Vol = 3d array
		site = current site being evaluated
		index = max x,y,z indicies
		layer = current layer under evaluation

	*/
	char filename[23];
	char temp;

	filename[0] =  'S';
	filename[1] =  'P';
	filename[2] =  'S';
	filename[3] =  '.';
	filename[4] =  'e';
```

```
filename[5] =  'x';
filename[6] =  'e';
filename[7] =  ' ';


if(layer <= 9)
{
      temp = layer;
      filename[8] =  48;
      filename[9] =  48;
      filename[10] =  temp + 48;
      filename[11] =  'L';
      filename[12] =  'a';
      filename[13] =  'y';
      filename[14] =  'e';
      filename[15] =  'r';
      filename[16] =    '_';
      filename[17] =  '1';
      filename[18] =  '.';
      filename[19] =  'j';
      filename[20] =  'p';
      filename[21] =  'g';
      filename[22] =  0;
}
else if(layer <= 99)
{
      temp = layer;
```

```
        filename[8] =  48;
        filename[9] = (temp / 10) + 48;
        filename[10] = (temp % 10) + 48;
        filename[11] =  'L';
        filename[12] =  'a';
        filename[13] =  'y';
        filename[14] =  'e';
        filename[15] =  'r';
        filename[16] =    '_';
        filename[17] =  '1';
        filename[18] =  '.';
        filename[19] =  'j';
        filename[20] =  'p';
        filename[21] =  'g';
        filename[22] =  0;
}
else
{
        temp = layer / 100;
        filename[8] = temp + 48;

        temp = layer - temp*100;
        filename[9] = (temp / 10) + 48;
        filename[10] = (temp % 10) + 48;
        filename[11] =  'L';
        filename[12] =  'a';
        filename[13] =  'y';
```

```
        filename[14] =  'e';
        filename[15] =  'r';
        filename[16] =    '_';
        filename[17] =  '1';
        filename[18] =  '.';
        filename[19] =  'j';
        filename[20] =  'p';
        filename[21] =  'g';
        filename[22] =  0;
  }



  ofstream outData("surface.dat", ios::out);

int i,j,k;
  int high;
  int x,y,z;
  int x_max, y_max, z_max;
  int bonds = 0;
  int site[3];

  int **output;

  int i_mod, j_mod;
```

```
x_max = index[0];
y_max = index[1];
z_max = index[2];

output = array2Dcreate(x_max,y_max);



for(i = 0; i < x_max; i++)
{
	for(j = 0; j < y_max; j++)
	{
		for(k = 0; k < z_max; k++)
		{
			if(Vol[i][j][k] == 1)
			{
				high = k;
			}
			if(k == (z_max - 1))
			{
				output[i][j] = high;
				high = 0;
			}
		}
	}
}
```

```
bonds = 0;

for(i = 2; i < (x_max - 1); i++)
{
      for(j = 2; j < (y_max - 1); j++)
      {
            i_mod = i%2;
            j_mod = j%2;

            if( (i_mod) == (j_mod) )
            {
                  if( (output[i][j] - output[i][j+1]) > 0)
                  {
                        bonds++;
                  }
                  if( (output[i][j] - output[i][j-1]) > 0)
                  {
                        bonds++;
                  }
                  if( (output[i][j] - output[i + 1][j]) > 0)
                  {
                        bonds++;
                  }
                  if( (output[i][j] - output[i - 1][j]) > 0)
                  {
                        bonds++;
                  }
```

```
                        if(output[i][j] == 0)
                        {
                              output[i][j] = output[i][j+1];
                        }
                        else if(bonds > 1)
                        {
                              output[i][j]++;
                        }
                  }

                  bonds = 0;
            }
      }

      outData <<"GRID\t"<<x_max<<"\t"<<y_max<<endl;
      outData <<"-5\t1\t1\t0\t0"<<endl;

      for(i = 0; i < x_max; i++)
      {
            for(j = 0; j < y_max; j++)
            {
                  outData  << output[i][j] <<  "\t";
            }
            outData <<endl;
      }
      outData.close();
      system(filename);
```

```
	free(output);
}
void wait(void)
{
	while(1)
	{
	}
}
void nrerror(char error_text[])
{
	fprintf(stderr, "...Run-time error....\n");
	fprintf(stderr, "%s\n", error_text);
	fprintf(stderr, "... now exiting to system ... \n");
	exit(1);
}


//Returns the Bessel function J0(x) for any real x.
float bessj0(float x)
{
	float ax,z;
	double xx,y,ans,ans1,ans2;		//Accumulate
polynomials in double precision.
		if ((ax=fabs(x)) < 8.0)		//Direct rational
function fit.
		{
			y=x*x;
```

```
            ans1=57568490574.0+y*(-13362590354.0+y*(651619640.7
            +y*(-11214424.18+y*(77392.33017+y*(-184.9052456)))));

            ans2=57568490411.0+y*(1029532985.0+y*(9494680.718
            +y*(59272.64853+y*(267.8532712+y*1.0))));

            ans=ans1/ans2;
      }
      else
      {
            z=8.0/ax;
            y=z*z;
            xx=ax-0.785398164;

            ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4
                  +y*(-0.2073370639e-5+y*0.2093887211e-6)));

            ans2 = -0.1562499995e-1+y*(0.1430488765e-3
                  +y*(-0.6911147651e-5+y*(0.7621095161e-6
                  -y*0.934945152e-7)));

            ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-
z*sin(xx)*ans2);
      }

      return ans;
}
```

```
//Returns the Bessel function J1(x) for any real x.
float bessj1(float x)
{
	float ax,z;
	double xx,y,ans,ans1,ans2;		//Accumulate
polynomials in double precision.

	if ((ax=fabs(x)) < 8.0)			//Direct rational
approximation.
	{
		y=x*x;

		ans1=x*(72362614232.0+y*(-7895059235.0+y*(242396853.1
			+y*(-2972611.439+y*(15704.48260+y*(-
30.16036606))))));

		ans2=144725228442.0+y*(2300535178.0+y*(18583304.74
			+y*(99447.43394+y*(376.9991397+y*1.0))));

		ans=ans1/ans2;
	}
	else
	{
		z=8.0/ax;
		y=z*z;
```

```
            xx=ax-2.356194491;

            ans1=1.0+y*(0.183105e-2+y*(-0.3516396496e-4
                  +y*(0.2457520174e-5+y*(-0.240337019e-6))));

            ans2=0.04687499995+y*(-0.2002690873e-3
                  +y*(0.8449199096e-5+y*(-0.88228987e-6
                  +y*0.105787412e-6)));

            ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);

            if (x < 0.0)
            {
                  ans = -ans;
            }
      }
      return ans;
}




//Returns the Bessel function Jn(x) for any real x and n = 2.
float bessj(int n, float x)
{
      int j,jsum,m,itemp;
```

```
	double dtemp;
	float ax,bj,bjm,bjp,sum,tox,ans;

	if (n < 2)
	{
		printf("n = %d\n",n);
		nrerror("Index n less than 2 in bessj");

	}

	ax=fabs(x);

	if (ax == 0.0)
	{
		return 0.0;
	}
	else if (ax > (float) n)		//Upwards recurrence from J0
and J1.
	{
		tox=2.0/ax;
		bjm=bessj0(ax);
		bj=bessj1(ax);
		for (j=1;j<n;j++)
		{
			bjp=j*tox*bj-bjm;
			bjm=bj;
			bj=bjp;
```

```
            }

            ans=bj;

      }
      else                                    //Downwards recurrence
from an even m here computed.
      {
            tox=2.0/ax;
            m=2*((n+(int) sqrt(ACC*n))/2);
      /*
            jsum will alternate between 0 and 1; when it is
            1, we accumulate in sum the even terms in
            (5.5.16).
      */
            jsum=0;
            bjp=ans=sum=0.0;
            bj=1.0;

            for (j=m;j>0;j--) //The downward recurrence.
            {
                  bjm=j*tox*bj-bjp;
                  bjp=bj;
                  bj=bjm;

                  if (fabs(bj) > BIGNO) //Renormalize to prevent
overflows.
```

```
                {
                        bj *= BIGNI;
                        bjp *= BIGNI;
                        ans *= BIGNI;
                        sum *= BIGNI;
                }
                if (jsum)
                {
                        sum += bj;              //Accumulate the sum.
                }

                jsum=!jsum;       //Change 0 to 1 or vice versa.

                if (j == n)
                {
                        ans=bjp;    // Save the unnormalized answer.
                }
            }

            sum=2.0*sum-bj;         // Compute (5.5.16)and use it to
normalize the answer.
            ans /= sum;
      }

      return x < 0.0 && (n & 1) ? -ans : ans;

}
```

```
//Geometric Field (Unobstructed Field)
double GeoField(double r, double r0, double k)
{
	double field;
	double temp;

	temp = pow( (cos( k * (r + r0) )),2) + pow(( sin( k * (r + r0) )
),2) ;
	field =  sqrt( temp ) / pow(r0,2) + 2 * r0 * r + pow(r,2);

	return field;
}

//Field modification caused by diffraction
double DifField(double u, double v)
{
	int i,j;

	double M, L;
	double U1, U2;
	double V1, V0;
	double theta;

	double temp;
	double field;
```

```
	U1 = 0;
	U2 = 0;
	V1 = 0;
	V0 = 0;

	//Calculation of Bessel Functions

	j = 1;  //Used to alternate sign in Bessel expansion
	for(i = 1; i <= (Bessel_max_order * 2); i = i + 2)
	{
		if(i < 2)
		{
			temp = bessj1(v);
			//printf("Bessel1(v=%f) = %f\n",v,temp);
			U1 = U1 + pow((-1.0),(j+1)) * ( pow((u/v),i) * temp
);
		}
		else
		{
			temp = bessj(i,v) ;
			U1 = U1 + pow((-1.0),(j+1)) * ( pow((u/v),i) * temp);
			//printf("Bessel%d(v=%f) = %f\n",i,v,temp);
		}

		j++;
	}
```

```
	j = 1;
	for(i = 2; i <= (Bessel_max_order * 2); i = i + 2)
	{

		U2 = U2 + pow((-1.0),(j+1)) * ( pow((u/v),i) * bessj(i,v)
);
		//printf("Bessel%d(v=%f) = %f\n",i,v,temp);
		//printf("u = %e, pow(u/v,i) = %e\n",u,( pow((u/v),i)));
		j++;
	}
	for(i = 0; i <= (Bessel_max_order * 2); i = i + 2)
	{

		if(i < 2)
		{
			V0 =  bessj0(v);
		}
		else
		{
			V0 =  V0+ pow((-1.0),(j+1)) * ( pow((v/u),i) *
bessj(i,v) );
		}
		j++;
	}
	for(i = 1; i <= (Bessel_max_order * 2); i = i + 2)
	{
```

```
            if(i < 2)
            {
                  V1= (v/u) * bessj1(v);
            }
            else
            {
                  V1 =  V1+ pow((-1.0),(j)) * ( pow((v/u),i) *
bessj(i,v) );
            }
            j++;
      }

      theta = u / 2;
      if(v > u)
      {
            M = ( U1 * sin(theta) ) - ( U2 * cos(theta) );
            L = ( U1 * cos(theta) ) + ( U2 * cos(theta) );
      }
      else
      {
            M = cos((v * v)/(2 * u)) + V0*cos(theta) - V1*sin(theta);
            L = sin((v * v)/(2 * u)) + V0*sin(theta) - V1*cos(theta);
      }

      field = pow(M,2) + pow(L,2);
      //printf("field = %e\n",field);
```

```
	return field;

}

void plot(int points, double *data)
{
	int i,j,k;
	int y = 11;
	double **graph,temp;

	FILE *fp;
	fp = fopen("Fresnel.dat","a");
	//GRID  MAX X   MAX Y
	//fprintf(fp,"GRID\t%d\t%d\n",2*points,y);
	//HOLEVALUE XSTEP YSTEP XORIGIN YORIGIN
	//fprintf(fp,"-5\t1\t1\t0\t0\n");

/*
  MALLOC(plot, sizeof(double *) * points * 2);
//    plot = (double *) malloc(sizeof(double)*points*2);
  for (i = 0; i < points*2; i++)
  {
    MALLOC(plot[i], sizeof(double) * y);
  }
*/
	graph = array2DcreateDouble(2*points, y);
	array2Dinitialgeo(graph,points,y);
```

```
//    for(i=1;i < (points-1); i++)
      for(i=0;i < (points-1); i++)
      {
            for(j = 0; j < y; j++)
            {
                  //plot[i][j] = data[(points-1)-i];
                  graph[i][j] = data[(points-1)-i];
                  //printf("data point = %e\n",data[(points-1)-i]);
            }
      }
//for(i=points+2;i < points + points -1; i++)
      for(i=0 ;i < points-1; i++)
      {
            for(j = 0; j < y; j++)
            {
                  //plot[i][j] = data[i-points-0];
                  graph[(points-1)+i][j] = data[i+1];
            }
      }
//NORMALIZE BASED UPON LARGEST VALUE IN PLOT
      temp = 0;
      for(i = 0; i < points; i++)
      {
            if( data[i] > temp)
                  temp = data[i];
      }
```

```
	for(i = 0; i < (2*points); i++)
	{
		for(j = 0; j < y; j++)
		graph[i][j] = graph[i][j]/temp;
	}


	for(i = 0; i < 2*points -2; i++)
	{
		for(j=0; j < 1; j++)
		{
			fprintf(fp,"%e",graph[i][j]);
		}
		fprintf(fp,"\n");
	}
	fclose(fp);
}

void FieldNormal(double *field,int size)
{
	int i;
	double temp =0;
//NORMALIZE BASED UPON LARGEST VALUE IN PLOT

	for(i = 0; i < size; i++)
	{
		if( field[i] > temp)
```

```
                temp = field[i];
        }

        for(i = 0; i < size; i++)
        {
                field[i]= field[i]/temp ;
        }


}

void array2Dinitialgeo( double **Vol, int x, int y)
{
        int i,j,k;

        for(i = 0; i < 2*x -1; i++)
        {
                for(j = 0; j < y- 1; j++)
                {
                        Vol[i][j] = 0;

                }
        }
}

double *Fresnel(float *pattern, double *params)
{
```

```
int i;



//Prpoblem Geometry Parameters
float pattern_length ,pattern_resolution;

      pattern_length = pattern[0];
      pattern_resolution = pattern[1];

double ApRadius,a,dSoA,dASu,lambda,k;


//Substrate Geometry Parameters
double r,theta,phi,x,y,z;
double L,m,n;     //double angles

//Source Geometry Parameters
double r0,theta0,phi0,x0,y0,z0;
double L0,m0,n0; //double angles

//Fresnel intetegral parametrs
double c,u,v;

double *field;
```

```
	int data_points = pattern_length / pattern_resolution;

	field = (double *) malloc(data_points * sizeof(double));


	//DEFIND THE GEOMETRY AND CONSTANTS OF PROBLEM

//IRRADIANCE PATTERN LENGTH


//APERTRUE RADIUS
		ApRadius = params[0];
		a = ApRadius;
//DISTANCE FROM SOURCE TO APERTRUE
		dSoA   = params[1];

//DISTANCE FROM APERTRUE TO SUBSTRATE
		dASu   = params[2];

//WAVELENGTH OF WAVE
		lambda       = params[3];
		k = (2 * PI) / lambda;

//SOURCE LOCATION
		x0 = 0;
		y0 = 0;
		z0 = -dSoA;
```

```
//SUBSTRATE OBSERVATION POINT
            y = 0;
            z = dASu;
            //x is varied to determine diffraction pattern

//GEOMETRY CALCULATIONS


            //Source
            r0 = sqrt(pow(x0,2)+pow(y0,2)+pow(z0,2));
            theta0 = PI/2;
            phi0= PI;

            //theta0 = acos(z0 / r0);
            //phi0= atan(y0 / x0);

            L0 = -1 * sin(theta0);
            m0 = sin(phi0) * sin(theta0);
            n0 = cos(theta0);

//*************************** FIELD CALCULATIONS
**********************//

            x = 0; // Start at origin

            for(i = 0; i < data_points; i++)
```

```
        {
            //GEOMETRY CALCULATIONS
            //Substrate
            r = sqrt(pow(x,2)+pow(y,2)+pow(z,2));
            theta = acos(z / r);
            phi = atan(y / x);
            L = cos(phi) * sin(theta);
            m = sin(phi) * sin(theta);
            n = cos(theta);


            //INTERGRAL INPUT CALCULATIONS

            c = r * sqrt( pow((L - L0),2) + pow((m - m0),2) );

            u = (k * pow(a,2) * (r0 + r) ) / (r0 * r);

            v = (k * a * c) / r;

            field[i] = GeoField(r,r0,k) * DifField(u,v) *
cos(theta0);

            x = x + pattern_resolution;
        }

    FieldNormal(field,data_points);
    plot(data_points,field);
```

```
	return(field);

}

double getField(int *site,int *max, double *field, float *pattern)
{
		ofstream outData("getField.txt", ios::app);
	//outData <<  "x , y , Site " << endl;
	double r;
	float lattice = 0.543;  // lattice constant for Silicon in
Nanometer
	float x,y;
	int x_max, y_max;
	int x_index, y_index; //values ussing array index at SUT orgin

	int x_nano, y_nano;		//nano window dimension
	int x_low, x_limit;		//limits for valid x coordinate
	int y_low, y_limit;		//limits for valid y coordinate
	int x_site,y_site, r_int;		//array index using nano
window origin
	int x_origin, y_origin;		//origin of nanowindow in
array indicies
	float length_SUT, width_SUT;  //length of SUT in nm
	float x_length, y_length;		//lenght of site from SUT
origin
	float pattern_length, pattern_resolution;
```

```
    x_index = site[0];  //array index for x
      y_index = site[1];      //array index for y

      outData<<"*************************"<<endl;
      outData <<"index for site x = "<<x_index<<" ,y =
"<<y_index<<endl;
      x_max = max[0];
      y_max = max[1];

      x_nano = x_max / 10;    //size of nano window in array index
      y_nano = y_max / 10;    //

      outData<<"size of nano window x = "<<x_nano<<", y =
"<<y_nano<<endl;

      length_SUT = (x_nano * lattice) *5;  //total length of SUT in nm
      width_SUT  = (y_nano * lattice) *5;  //total width of SUT in nm

      outData<<"total length of SUT x = "<<length_SUT<<" ,y =
"<<width_SUT<<endl;

      x_length = x_index * (length_SUT/x_max);  //length of site from
SUT origin
      y_length = y_index * (width_SUT/ y_max);  //width of site from
SUT origin

      x_origin = x_max / 2; //Origin of nano window
```

```
      y_origin = y_max / 2;

      outData <<"Origin of nanowindow is x = "<<x_origin<<", y =
"<<y_origin<<endl;

      x_site = abs(x_index - x_origin);   //Distance of point from nano
      y_site = abs(y_index - y_origin);   //origin, center of apature.
as array index

      outData<<"Distance from nanowindow origin in indicies x =
"<<x_site<<", y = "<<y_site<<endl;

      x = x_site * (length_SUT/x_max);//Distance of point from
      y = y_site * (width_SUT /y_max);//nano origin, in nm

      outData<<"distance from nano origin in nm x = "<<x<<", y =
"<<y<<endl;

      pattern_length = pattern[0]*1e-2;
      pattern_resolution = pattern[1]*1e-1;

      outData << "pattern length = "<<pattern_length<<endl;
      outData << "pattern resolution = "<<pattern_resolution<<endl;

      //Calculates the corresponding r value of field
      r = (sqrt(x*x + y*y)) * pow(10.0,-9);     //Distance from center
of Apreture in meters
```

```
	outData<<"calculated r = "<<r<<endl;

	r_int = (r / pattern_resolution); // index in field array for
site

	outData<< "index of r = "<<r_int<<endl;

	outData.close();
	return(field[r_int]);

}

double roughness(int *** Volume, int *index, int*nano,int standard)
{
	ofstream summation("Summation.txt", ios::app);
	int x_max, y_max, z_max;
	int xs,ys,rs;
	int i,j,k=0;
	double sum;
	double avg_height = 0;
	double dif_sq;
	int stack;
	double rough;
	double sum_of_squares = 0;
	double sum_of_heights = 0;
	int height;
```

```
      int x_nano, y_nano,r_nano;          //nano window dimension
      int offset_x, offset_y,offset_r; //offsets used to center the
nano window on SUT
      int x_low, x_limit;           //limits for valid x coordinate
      int y_low, y_limit;           //limits for valid y coordinate
      int x_range, y_range;   //diffrence between low and limit
      int r_low, r_limit;
      int SUT_r,xr, SUT_rx, SUT_ry;

      x_max = index[0];
      y_max = index[1];
      z_max = index[2];

      xs = x_max;
      ys = y_max;

      x_nano = nano[0];
      y_nano = nano[1];

      offset_x = (xs - (x_nano * (10-standard)))/2;
      offset_y = (ys - (y_nano * (10-standard)))/2;

      r_nano =xs/2 - offset_x;

      xr = xs/2;

      offset_r = xr - r_nano ;
```

```
x_low = offset_x;
x_limit = xs - offset_x;
x_range = x_limit - x_low;

y_low = offset_y;
y_limit = ys - offset_y;
y_range = y_limit -  y_low;

r_low =offset_r;
r_limit = xr - offset_r;

sum_of_heights = 0;
sum_of_squares = 0;
avg_height = 0;
sum = 0;
dif_sq =0;



for(i = x_low; i < x_limit; i++)
{
      for(j = y_low; j < y_limit; j++)
      {
            SUT_rx = abs(i - xr);
            SUT_ry = abs(j - xr);
            SUT_r = sqrt(double (SUT_rx*SUT_rx + SUT_ry*SUT_ry));
            if(SUT_r <= ( r_limit * 0.9) )
```

```
                {
                        stack = topofstack(i,j,z_max,Volume);
                        sum = sum + stack;
                        sum_of_heights = sum_of_heights + stack;
                        sum_of_squares = sum_of_squares + (stack *
stack);
                        k++;
                        //cout <<"SUM = " <<sum<<", Top of stack:
"<<stack<<endl;
                }
            }
      }

      avg_height = sum / (k );//(r_limit * r_limit);
      //cout << "Average height = " << avg_height <<", SUM:
"<<sum<<endl;
      k = 0;
      for(i = x_low; i < x_limit; i++)
      {
            for(j = y_low; j < y_limit; j++)
            {
                  SUT_rx = abs(i - xr);
                  SUT_ry = abs(j - xr);
                  SUT_r = sqrt(double (SUT_rx*SUT_rx + SUT_ry*SUT_ry));
                  if(SUT_r <= ( r_limit *0.9))
                  {
                        height = topofstack(i,j,z_max,Volume);
```

```
                        //cout <<"Height : "<<height<<" AVG height :
"<<avg_height<<endl;
                        dif_sq = dif_sq + ((height - avg_height) *
(height - avg_height));
                        summation << height <<"\t"<< avg_height << "\t"
<< dif_sq << endl;
                        k++;
                  }
            }
      }
      //cout << "Diffrence Square : " << dif_sq <<", K :"<<k<<endl;
      //cout << "R Limit : " << r_limit <<endl;
      rough =  sqrt(  dif_sq);
      rough = rough / k;
      rough = sqrt( ((k * sum_of_squares) - (sum_of_heights *
sum_of_heights)) / (k * k));
      //cout << "Rough : "<< rough << endl;
      //rough = sqrt( (4 / (r_limit * r_limit * pi) ) * dif_sq );
      return(rough);

}

int topofstack(int x,int y,int max, int ***Vol)
{
      int i,j,k;
      int height;
      height = 0;
```

```
	i = 0;


	while(i < max)
	{
		if( Vol[x][y][i] == 1)
				height = i;
		i++;
	}
//cout<<"height - "<<height<<endl;
	return(height);


}
#include <math.h>

#define ACC 40.0		//Make larger to increase accuracy.
#define BIGNO 1.0e10
#define BIGNI 1.0e-10
#define Bessel_max_order 20
#define PI 3.1415926535897932

//
#define oops(s) { perror((s)); exit(EXIT_FAILURE); }
#define MALLOC(s,t) if(((s) = malloc(t)) == NULL) { oops("error:
malloc() "); }
#define INCREMENT 10
//
```

```
float bessj0(float );
float bessj1(float );
float bessj(int , float );
void nrerror(char *);
double GeoField(double, double, double);
double DifField(double, double);
void FieldNormal(double *,int);
*double Fresnel(float *, double*);
double getField(int, int, double *, float *);

//int** array2Dcreate(int ,int );
void array2Dinitialgeo( int **, int , int );



void nrerror(char error_text[])
{
      fprintf(stderr, "...Run-time error....\n");
      fprintf(stderr, "%s\n", error_text);
      fprintf(stderr, "... now exiting to system ... \n");
      exit(1);
}


//Returns the Bessel function J0(x) for any real x.
float bessj0(float x)
{
```

```
	float ax,z;
	double xx,y,ans,ans1,ans2;		//Accumulate
polynomials in double precision.
		if ((ax=fabs(x)) < 8.0)		//Direct rational
function fit.
		{
			y=x*x;
			ans1=57568490574.0+y*(-13362590354.0+y*(651619640.7
			+y*(-11214424.18+y*(77392.33017+y*(-184.9052456)))));

			ans2=57568490411.0+y*(1029532985.0+y*(9494680.718
			+y*(59272.64853+y*(267.8532712+y*1.0))));

			ans=ans1/ans2;
		}
		else
		{
			z=8.0/ax;
			y=z*z;
			xx=ax-0.785398164;

			ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4
				+y*(-0.2073370639e-5+y*0.2093887211e-6)));

			ans2 = -0.1562499995e-1+y*(0.1430488765e-3
				+y*(-0.6911147651e-5+y*(0.7621095161e-6
				-y*0.934945152e-7)));
```

```
                ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-
z*sin(xx)*ans2);
            }

      return ans;
}


//Returns the Bessel function J1(x) for any real x.
float bessj1(float x)
{
      float ax,z;
      double xx,y,ans,ans1,ans2;                //Accumulate
polynomials in double precision.

      if ((ax=fabs(x)) < 8.0)                   //Direct rational
approximation.
      {
            y=x*x;

            ans1=x*(72362614232.0+y*(-7895059235.0+y*(242396853.1
                  +y*(-2972611.439+y*(15704.48260+y*(-
30.16036606))))));

            ans2=144725228442.0+y*(2300535178.0+y*(18583304.74
                  +y*(99447.43394+y*(376.9991397+y*1.0))));
```

```
            ans=ans1/ans2;
        }
        else
        {
            z=8.0/ax;
            y=z*z;
            xx=ax-2.356194491;

            ans1=1.0+y*(0.183105e-2+y*(-0.3516396496e-4
                +y*(0.2457520174e-5+y*(-0.240337019e-6))));

            ans2=0.04687499995+y*(-0.2002690873e-3
                +y*(0.8449199096e-5+y*(-0.88228987e-6
                +y*0.105787412e-6)));

            ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);

            if (x < 0.0)
            {
                ans = -ans;
            }
        }
        return ans;
}
```

```
//Returns the Bessel function Jn(x) for any real x and n = 2.
float bessj(int n, float x)
{
	int j,jsum,m,itemp;
	double dtemp;
	float ax,bj,bjm,bjp,sum,tox,ans;

	if (n < 2)
	{
		printf("n = %d\n",n);
		nrerror("Index n less than 2 in bessj");

	}

	ax=fabs(x);

	if (ax == 0.0)
	{
		return 0.0;
	}
	else if (ax > (float) n)		//Upwards recurrence from J0
and J1.
	{
		tox=2.0/ax;
```

```
            bjm=bessj0(ax);
            bj=bessj1(ax);
            for (j=1;j<n;j++)
            {
                  bjp=j*tox*bj-bjm;
                  bjm=bj;
                  bj=bjp;
            }

            ans=bj;

      }
      else                                    //Downwards recurrence
from an even m here computed.
      {
            tox=2.0/ax;
            m=2*((n+(int) sqrt(ACC*n))/2);
      /*
            jsum will alternate between 0 and 1; when it is
            1, we accumulate in sum the even terms in
            (5.5.16).
      */
            jsum=0;
            bjp=ans=sum=0.0;
            bj=1.0;

            for (j=m;j>0;j--) //The downward recurrence.
```

```
        {
                bjm=j*tox*bj-bjp;
                bjp=bj;
                bj=bjm;

                if (fabs(bj) > BIGNO) //Renormalize to prevent
overflows.
                {
                        bj *= BIGNI;
                        bjp *= BIGNI;
                        ans *= BIGNI;
                        sum *= BIGNI;
                }
                if (jsum)
                {
                        sum += bj;              //Accumulate the sum.
                }

                jsum=!jsum;       //Change 0 to 1 or vice versa.

                if (j == n)
                {
                        ans=bjp;    // Save the unnormalized answer.
                }
        }
```

```
            sum=2.0*sum-bj;          // Compute (5.5.16)and use it to
normalize the answer.
            ans /= sum;
      }

      return x < 0.0 && (n & 1) ? -ans : ans;

}

//Geometric Field (Unobstructed Field)
double GeoField(double r, double r0, double k)
{
      double field;
      double temp;

      temp = pow( (cos( k * (r + r0) )),2) + pow(( sin( k * (r + r0) )
),2) ;
      field =  sqrt( temp ) / pow(r0,2) + 2 * r0 * r + pow(r,2);

      return field;
}

//Field modification caused by diffraction
double DifField(double u, double v)
{
      int i,j;
```

```
double M, L;
double U1, U2;
double V1, V0;
double theta;

double temp;
double field;

U1 = 0;
U2 = 0;
V1 = 0;
V0 = 0;

//Calculation of Bessel Functions

j = 1;  //Used to alternate sign in Bessel expansion
for(i = 1; i <= (Bessel_max_order * 2); i = i + 2)
{
      if(i < 2)
      {
            temp = bessj1(v);
            //printf("Bessel1(v=%f) = %f\n",v,temp);
            U1 = U1 + pow((-1),(j+1)) * ( pow((u/v),i) * temp );
      }
      else
      {
            temp = bessj(i,v) ;
```

```
                U1 = U1 + pow((-1),(j+1)) * ( pow((u/v),i) * temp);
                //printf("Bessel%d(v=%f) = %f\n",i,v,temp);
            }

            j++;
        }

        j = 1;
        for(i = 2; i <= (Bessel_max_order * 2); i = i + 2)
        {

            U2 = U2 + pow((-1),(j+1)) * ( pow((u/v),i) * bessj(i,v) );
            //printf("Bessel%d(v=%f) = %f\n",i,v,temp);
            //printf("u = %e, pow(u/v,i) = %e\n",u,( pow((u/v),i)));
            j++;
        }
        for(i = 0; i <= (Bessel_max_order * 2); i = i + 2)
        {

            if(i < 2)
            {
                V0 =  bessj0(v);
            }
            else
            {
                V0 =  V0+ pow((-1),(j+1)) * ( pow((v/u),i) *
bessj(i,v) );
```

```
            }
            j++;
      }
      for(i = 1; i <= (Bessel_max_order * 2); i = i + 2)
      {

            if(i < 2)
            {
                  V1= (v/u) * bessj1(v);
            }
            else
            {
                  V1 =  V1+ pow((-1),(j)) * ( pow((v/u),i) * bessj(i,v)
);
            }
            j++;
      }

      theta = u / 2;
      if(v > u)
      {
            M = ( U1 * sin(theta) ) - ( U2 * cos(theta) );
            L = ( U1 * cos(theta) ) + ( U2 * cos(theta) );
      }
      else
      {
            M = cos((v * v)/(2 * u)) + V0*cos(theta) - V1*sin(theta);
```

```
            L = sin((v * v)/(2 * u)) + V0*sin(theta) - V1*cos(theta);
      }

      field = pow(M,2) + pow(L,2);
      //printf("field = %e\n",field);

      return field;

}

void plot(int points, double *data)
{
      int i,j,k;
      int y = 11;
      double **plot,temp;

      FILE *fp;
      fp = fopen("Fresnel.dat","w");
      //GRID  MAX X   MAX Y
      //fprintf(fp,"GRID\t%d\t%d\n",2*points,y);
      //HOLEVALUE XSTEP YSTEP XORIGIN YORIGIN
      //fprintf(fp,"-5\t1\t1\t0\t0\n");


  MALLOC(plot, sizeof(double *) * points * 2);
//    plot = (double *) malloc(sizeof(double)*points*2);
  for (i = 0; i < points*2; i++)
```

```
  {
    MALLOC(plot[i], sizeof(double) * y);
  }


      array2Dinitialgeo(plot,points,y);
//    for(i=1;i < (points-1); i++)
      for(i=0;i < (points-1); i++)
      {
            for(j = 0; j < y; j++)
            {
                  //plot[i][j] = data[(points-1)-i];
                  plot[i][j] = data[(points-1)-i];
                  //printf("%l\n",data[i]);
            }
      }
//for(i=points+2;i < points + points -1; i++)
      for(i=0 ;i < points-1; i++)
      {
            for(j = 0; j < y; j++)
            {
                  //plot[i][j] = data[i-points-0];
                  plot[(points-1)+i][j] = data[i+1];
            }
      }
//NORMALIZE BASED UPON LARGEST VALUE IN PLOT
      temp = 0;
```

```
	for(i = 0; i < points; i++)
	{
		if( data[i] > temp)
			temp = data[i];
	}

	for(i = 0; i < (2*points); i++)
	{
		for(j = 0; j < y; j++)
		plot[i][j] = plot[i][j]/temp;
	}


	for(i = 0; i < 2*points -2; i++)
	{
		for(j=0; j < 1; j++)
		{
			fprintf(fp,"%e",plot[i][j]);
		}
		fprintf(fp,"\n");
	}

}

void FieldNormal(double *field,int size)
{
	int i;
```

```
	double temp =0;
//NORMALIZE BASED UPON LARGEST VALUE IN PLOT

	for(i = 0; i < size; i++)
	{
		if( field[i] > temp)
			temp = field[i];
	}

	for(i = 0; i < size; i++)
	{
		field[i]= field[i]/temp ;
	}


}

void array2Dinitialgeo( int **Vol, int x, int y)
{
	int i,j,k;

	for(i = 0; i < 2*x -1; i++)
	{
		for(j = 0; j < y- 1; j++)
		{
			Vol[i][j] = 0;
```

```
            }
      }
}

double *Fresnel(float *pattern, double *params)
{

      int i;



      //Prpoblem Geometry Parameters
      float pattern_length ,pattern_resolution;

            pattern_length = pattern[0];
            pattern_resolution = pattern[1];

      double ApRadius,a,dSoA,dASu,lambda,k;



      //Substrate Geometry Parameters
      double r,theta,phi,x,y,z;
      double L,m,n;     //double angles

      //Source Geometry Parameters
      double r0,theta0,phi0,x0,y0,z0;
      double L0,m0,n0; //double angles
```

```
	//Fresnel intetegral parametrs
	double c,u,v;

	double *field;

	int data_points = pattern_length / pattern_resolution;

	field = (double *) malloc(data_points * sizeof(double));


	//DEFIND THE GEOMETRY AND CONSTANTS OF PROBLEM

//IRRADIANCE PATTERN LENGTH


//APERTRUE RADIUS
		ApRadius = params[0];
		a = ApRadius;
//DISTANCE FROM SOURCE TO APERTRUE
		dSoA   = params[1];

//DISTANCE FROM APERTRUE TO SUBSTRATE
		dASu   = params[2];

//WAVELENGTH OF WAVE
		lambda       = params[3];
```

```
            k = (2 * PI) / lambda;

//SOURCE LOCATION
            x0 = 0;
            y0 = 0;
            z0 = -dSoA;

//SUBSTRATE OBSERVATION POINT
            y = 0;
            z = dASu;
            //x is varied to determine diffraction pattern

//GEOMETRY CALCULATIONS


            //Source
            r0 = sqrt(pow(x0,2)+pow(y0,2)+pow(z0,2));
            theta0 = PI/2;
            phi0= PI;

            //theta0 = acos(z0 / r0);
            //phi0= atan(y0 / x0);

            L0 = -1 * sin(theta0);
            m0 = sin(phi0) * sin(theta0);
            n0 = cos(theta0);
```

```
//*************************** FIELD CALCULATIONS
**********************//

            x = 0; // Start at origin

            for(i = 0; i < data_points; i++)
            {
                  //GEOMETRY CALCULATIONS
                  //Substrate
                  r = sqrt(pow(x,2)+pow(y,2)+pow(z,2));
                  theta = acos(z / r);
                  phi = atan(y / x);
                  L = cos(phi) * sin(theta);
                  m = sin(phi) * sin(theta);
                  n = cos(theta);


                  //INTERGRAL INPUT CALCULATIONS

                  c = r * sqrt( pow((L - L0),2) + pow((m - m0),2) );

                  u = (k * pow(a,2) * (r0 + r) ) / (r0 * r);

                  v = (k * a * c) / r;

                  field[i] = GeoField(r,r0,k) * DifField(u,v) *
cos(theta0);
```

```
                x = x + pattern_resolution;
            }

      FieldNormal(field,data_points);
      plot(data_points,field);
      return(field);

}

double getField(int *site,int *max, double *field, float *pattern)
{
      double r;
      float lattice = 0.543;  // lattice constant for Silicon in
Nanometer
      float x,y;
      int x_max, y_max;
      int x_index, y_index; //values ussing array index at SUT orgin

      int x_nano, y_nano;           //nano window dimension
      int x_low, x_limit;           //limits for valid x coordinate
      int y_low, y_limit;           //limits for valid y coordinate
      int x_site,y_site, r_int;           //array index using nano
window origin
      int x_origin, y_origin;             //origin of nanowindow in
array indicies
      float length_SUT, width_SUT;  //length of SUT in nm
```

```
      float x_length, y_length;           //lenght of site from SUT
origin
      float pattern_length, pattern_resolution;


    x_index = site[0];  //array index for x
      y_index = site[1];      //array index for y


      x_max = max[0];
      y_max = max[1];


      x_nano = x_max / 10;    //size of nano window in array index
      y_nano = y_max / 10;    //

      length_SUT = (x_nano * lattice) *5;  //total length of SUT in nm
      width_SUT  = (y_nano * lattice) *5;  //total width of SUT in nm


      x_length = x_index * (length_SUT/x_max);  //length of site from
SUT origin
      y_length = y_index * (width_SUT/ y_max);  //width of site from
SUT origin


      x_origin = x_max / 2; //Origin of nano window
      y_origin = y_max / 2;


      x_site = abs(x_index - x_origin);   //Distance of point from nano
      y_site = abs(y_index - y_origin);   //origin, center of apature.
as array index
```

```
      x = x_site * (length_SUT/x_max);//Distance of point from
      y = y_site * (width_SUT /y_max);//nano origin, in nm

      pattern_length = pattern[0];
      pattern_resolution = pattern[1];

      //Calculates the corresponding r value of field
      r = (sqrt(x*x + y*y)) *10^(-9);     //Distance from center of
Apreture in meters

      r_int = r / pattern_resolution; // index in field array for site

      return(field[r_int]);

} # include <cstdlib>
# include <iostream>
# include <iomanip>
# include <ctime>

using namespace std;


double d_uniform_01 ( int *seed );
int i4_bit_hi1 ( int n );
int i4_bit_lo0 ( int n );
```

```
void i4_sobol ( int dim_num, int *seed, float quasi[ ] );//edited
maxcol
int i4_uniform ( int b, int c, int *seed );
unsigned int i4_xor ( unsigned int i, unsigned int j );
int i8_bit_hi1 ( long int n );
int i8_bit_lo0 ( long int n );
void i8_sobol ( int dim_num, long int *seed, double quasi[ ] );
unsigned long int i8_xor ( unsigned long int i, unsigned long int j );
void timestamp ( void );
int get_seed ( void );
void timeStep(double * , float *, double *, int *);
void pdfSelection(float *, int*, int*);
void pdfVacantSelect(float *, int *, int*);
void site(int *, int , int , int *);
void siteUpdate(int ***, int *, int *, int*, int);
void ValidSite(int*,int , int, int, int*, int*,int ***,float, int **
,int);


//*********************************************************************
*********

double d_uniform_01 ( int *seed )

//*********************************************************************
*********
//
```

```
//  Purpose:
//
//    D_UNIFORM_01 returns a unit double precision pseudorandom number.
//
//  Discussion:
//
//    This routine implements the recursion
//
//      seed = 16807 * seed mod ( 2**31 - 1 )
//      d_uniform_01 = seed / ( 2**31 - 1 )
//
//    The integer arithmetic never requires more than 32 bits,
//    including a sign bit.
//
//    If the initial seed is 12345, then the first three computations
are
//
//      Input     Output      D_UNIFORM_01
//      SEED      SEED
//
//         12345   207482415  0.096616
//     207482415  1790989824  0.833995
//    1790989824  2035175616  0.947702
//
//  Modified:
//
//    11 August 2004
```

```
//
//  Author:
//
//    John Burkardt
//
//  Reference:
//
//    Paul Bratley, Bennett Fox, L E Schrage,
//    A Guide to Simulation,
//    Springer Verlag, pages 201-202, 1983.
//
//    Pierre L'Ecuyer,
//    Random Number Generation,
//    in Handbook of Simulation
//    edited by Jerry Banks,
//    Wiley Interscience, page 95, 1998.
//
//    Bennett Fox,
//    Algorithm 647:
//    Implementation and Relative Efficiency of Quasirandom
//    Sequence Generators,
//    ACM Transactions on Mathematical Software,
//    Volume 12, Number 4, pages 362-376, 1986.
//
//    P A Lewis, A S Goodman, J M Miller,
//    A Pseudo-Random Number Generator for the System/360,
//    IBM Systems Journal,
```

```
//    Volume 8, pages 136-143, 1969.
//
//  Parameters:
//
//    Input/output, int *SEED, the "seed" value.  Normally, this
//    value should not be 0.  On output, SEED has been updated.
//
//    Output, double D_UNIFORM_01, a new pseudorandom variate, strictly
between
//    0 and 1.
//
{
  int k;
  double r;

  k = *seed / 127773;

  *seed = 16807 * ( *seed - k * 127773 ) - k * 2836;

  if ( *seed < 0 )
  {
    *seed = *seed + 2147483647;
  }
//
//  Although SEED can be represented exactly as a 32 bit integer,
//  it generally cannot be represented exactly as a 32 bit real number!
//
```

```
  r = ( double ) ( *seed ) * 4.656612875E-10;

  return r;
}
//****************************************************************************
*********

int i4_bit_hi1 ( int n )

//****************************************************************************
*********
//
//  Purpose:
//
//    I4_BIT_HI1 returns the position of the high 1 bit base 2 in an
integer.
//
//  Example:
//
//       N    Binary    Hi 1
//    ----    --------  ----
//       0           0     0
//       1           1     1
//       2          10     2
//       3          11     2
//       4         100     3
//       5         101     3
```

```
//       6         110     3
//       7         111     3
//       8        1000     4
//       9        1001     4
//      10        1010     4
//      11        1011     4
//      12        1100     4
//      13        1101     4
//      14        1110     4
//      15        1111     4
//      16       10000     5
//      17       10001     5
//    1023  1111111111    10
//    1024 10000000000    11
//    1025 10000000001    11
//
//  Modified:
//
//    13 March 2003
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    Input, int N, the integer to be measured.
```

```
//    N should be nonnegative.  If N is nonpositive, I4_BIT_HI1
//    will always be 0.
//
//    Output, int I4_BIT_HI1, the location of the high order bit.
//
{
  int bit;

  bit = 0;

  while ( 0 < n )
  {
    bit = bit + 1;
    n = n / 2;
  }

  return bit;
}
//****************************************************************************
*********

int i4_bit_lo0 ( int n )

//****************************************************************************
*********
//
//  Purpose:
```

```
//
//    I4_BIT_LO0 returns the position of the low 0 bit base 2 in an
integer.
//
//  Example:
//
//       N    Binary    Lo 0
//    ----    --------  ----
//       0           0     1
//       1           1     2
//       2          10     1
//       3          11     3
//       4         100     1
//       5         101     2
//       6         110     1
//       7         111     4
//       8        1000     1
//       9        1001     2
//      10        1010     1
//      11        1011     3
//      12        1100     1
//      13        1101     2
//      14        1110     1
//      15        1111     5
//      16       10000     1
//      17       10001     2
//    1023  1111111111     1
```

```
//    1024 10000000000     1
//    1025 10000000001     1
//
//  Modified:
//
//    13 March 2003
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    Input, int N, the integer to be measured.
//    N should be nonnegative.
//
//    Output, int I4_BIT_LO0, the position of the low 1 bit.
//
{
  int bit;
  int n2;

  bit = 0;

  while ( true )
  {
    bit = bit + 1;
```

```
    n2 = n / 2;

    if ( n == 2 * n2 )
    {
      break;
    }

    n = n2;

  }

  return bit;
}
//****************************************************************************
*********

void i4_sobol ( int dim_num, int *seed, float quasi[ ] )

//****************************************************************************
*********
//
//  Purpose:
//
//    I4_SOBOL generates a new quasirandom Sobol vector with each call.
//
//  Discussion:
//
```

```
//    The routine adapts the ideas of Antonov and Saleev.
//
//  Modified:
//
//    03 August 2004
//
//  Reference:
//
//    Antonov and Saleev,
//    USSR Computational Mathematics and Mathematical Physics,
//    Volume 19, 1980, pages 252 - 256.
//
//    Paul Bratley and Bennett Fox,
//    Algorithm 659:
//    Implementing Sobol's Quasirandom Sequence Generator,
//    ACM Transactions on Mathematical Software,
//    Volume 14, Number 1, pages 88-100, 1988.
//
//    Bennett Fox,
//    Algorithm 647:
//    Implementation and Relative Efficiency of Quasirandom
//    Sequence Generators,
//    ACM Transactions on Mathematical Software,
//    Volume 12, Number 4, pages 362-376, 1986.
//
//    I Sobol,
//    USSR Computational Mathematics and Mathematical Physics,
```

```
//    Volume 16, pages 236-242, 1977.
//
//    I Sobol and Levitan,
//    The Production of Points Uniformly Distributed in a
Multidimensional
//    Cube (in Russian),
//    Preprint IPM Akad. Nauk SSSR,
//    Number 40, Moscow 1976.
//
//  Parameters:
//
//    Input, int DIM_NUM, the number of spatial dimensions.
//    DIM_NUM must satisfy 2 <= DIM_NUM <= 40.
//
//    Input/output, int *SEED, the "seed" for the sequence.
//    This is essentially the index in the sequence of the quasirandom
//    value to be generated.  On output, SEED has been set to the
//    appropriate next value, usually simply SEED+1.
//    If SEED is less than 0 on input, it is treated as though it were
0.
//    An input value of 0 requests the first (0-th) element of the
sequence.
//
//    Output, float QUASI(DIM_NUM), the next quasirandom vector.
//
{
# define DIM_MAX 40
```

```
  static int atmost = 1073741823;
  static int dim_num_save = 0;
  int i;
  bool includ[8];
  static bool initialized = false;
  int j;
  int j2;
  int k;
  int l;
  static int lastq[DIM_MAX];
  int m;
  static int maxcol;
  int newv;
  static int poly[DIM_MAX] =
  {
        1,   3,   7,  11,  13,  19,  25,  37,  59,  47,
       61,  55,  41,  67,  97,  91, 109, 103, 115, 131,
      193, 137, 145, 143, 241, 157, 185, 167, 229, 171,
      213, 191, 253, 203, 211, 239, 247, 285, 369, 299
  };
  static float recipd;
  static int seed_save = 0;
  int seed_temp;
  static int v[DIM_MAX][30];
//
  if ( !initialized || dim_num != dim_num_save )
```

```
  {
    initialized = true;
//
//  Initialize (part of) V.
//
    v[ 0][0] = 1;
    v[ 1][0] = 1;
    v[ 2][0] = 1;
    v[ 3][0] = 1;
    v[ 4][0] = 1;
    v[ 5][0] = 1;
    v[ 6][0] = 1;
    v[ 7][0] = 1;
    v[ 8][0] = 1;
    v[ 9][0] = 1;
    v[10][0] = 1;
    v[11][0] = 1;
    v[12][0] = 1;
    v[13][0] = 1;
    v[14][0] = 1;
    v[15][0] = 1;
    v[16][0] = 1;
    v[17][0] = 1;
    v[18][0] = 1;
    v[19][0] = 1;
    v[20][0] = 1;
    v[21][0] = 1;
```

```
v[22][0] = 1;
v[23][0] = 1;
v[24][0] = 1;
v[25][0] = 1;
v[26][0] = 1;
v[27][0] = 1;
v[28][0] = 1;
v[29][0] = 1;
v[30][0] = 1;
v[31][0] = 1;
v[32][0] = 1;
v[33][0] = 1;
v[34][0] = 1;
v[35][0] = 1;
v[36][0] = 1;
v[37][0] = 1;
v[38][0] = 1;
v[39][0] = 1;

v[ 2][1] = 1;
v[ 3][1] = 3;
v[ 4][1] = 1;
v[ 5][1] = 3;
v[ 6][1] = 1;
v[ 7][1] = 3;
v[ 8][1] = 3;
v[ 9][1] = 1;
```

```
v[10][1] = 3;
v[11][1] = 1;
v[12][1] = 3;
v[13][1] = 1;
v[14][1] = 3;
v[15][1] = 1;
v[16][1] = 1;
v[17][1] = 3;
v[18][1] = 1;
v[19][1] = 3;
v[20][1] = 1;
v[21][1] = 3;
v[22][1] = 1;
v[23][1] = 3;
v[24][1] = 3;
v[25][1] = 1;
v[26][1] = 3;
v[27][1] = 1;
v[28][1] = 3;
v[29][1] = 1;
v[30][1] = 3;
v[31][1] = 1;
v[32][1] = 1;
v[33][1] = 3;
v[34][1] = 1;
v[35][1] = 3;
v[36][1] = 1;
```

```
v[37][1] = 3;
v[38][1] = 1;
v[39][1] = 3;

v[ 3][2] = 7;
v[ 4][2] = 5;
v[ 5][2] = 1;
v[ 6][2] = 3;
v[ 7][2] = 3;
v[ 8][2] = 7;
v[ 9][2] = 5;
v[10][2] = 5;
v[11][2] = 7;
v[12][2] = 7;
v[13][2] = 1;
v[14][2] = 3;
v[15][2] = 3;
v[16][2] = 7;
v[17][2] = 5;
v[18][2] = 1;
v[19][2] = 1;
v[20][2] = 5;
v[21][2] = 3;
v[22][2] = 3;
v[23][2] = 1;
v[24][2] = 7;
v[25][2] = 5;
```

```
v[26][2] = 1;
v[27][2] = 3;
v[28][2] = 3;
v[29][2] = 7;
v[30][2] = 5;
v[31][2] = 1;
v[32][2] = 1;
v[33][2] = 5;
v[34][2] = 7;
v[35][2] = 7;
v[36][2] = 5;
v[37][2] = 1;
v[38][2] = 3;
v[39][2] = 3;

v[ 5][3] =  1;
v[ 6][3] =  7;
v[ 7][3] =  9;
v[ 8][3] = 13;
v[ 9][3] = 11;
v[10][3] =  1;
v[11][3] =  3;
v[12][3] =  7;
v[13][3] =  9;
v[14][3] =  5;
v[15][3] = 13;
v[16][3] = 13;
```

```
v[17][3] = 11;
v[18][3] =  3;
v[19][3] = 15;
v[20][3] =  5;
v[21][3] =  3;
v[22][3] = 15;
v[23][3] =  7;
v[24][3] =  9;
v[25][3] = 13;
v[26][3] =  9;
v[27][3] =  1;
v[28][3] = 11;
v[29][3] =  7;
v[30][3] =  5;
v[31][3] = 15;
v[32][3] =  1;
v[33][3] = 15;
v[34][3] = 11;
v[35][3] =  5;
v[36][3] =  3;
v[37][3] =  1;
v[38][3] =  7;
v[39][3] =  9;

v[ 7][4] =  9;
v[ 8][4] =  3;
v[ 9][4] = 27;
```

```
v[10][4] = 15;
v[11][4] = 29;
v[12][4] = 21;
v[13][4] = 23;
v[14][4] = 19;
v[15][4] = 11;
v[16][4] = 25;
v[17][4] =  7;
v[18][4] = 13;
v[19][4] = 17;
v[20][4] =  1;
v[21][4] = 25;
v[22][4] = 29;
v[23][4] =  3;
v[24][4] = 31;
v[25][4] = 11;
v[26][4] =  5;
v[27][4] = 23;
v[28][4] = 27;
v[29][4] = 19;
v[30][4] = 21;
v[31][4] =  5;
v[32][4] =  1;
v[33][4] = 17;
v[34][4] = 13;
v[35][4] =  7;
v[36][4] = 15;
```

```
v[37][4] =  9;
v[38][4] = 31;
v[39][4] =  9;

v[13][5] = 37;
v[14][5] = 33;
v[15][5] =  7;
v[16][5] =  5;
v[17][5] = 11;
v[18][5] = 39;
v[19][5] = 63;
v[20][5] = 27;
v[21][5] = 17;
v[22][5] = 15;
v[23][5] = 23;
v[24][5] = 29;
v[25][5] =  3;
v[26][5] = 21;
v[27][5] = 13;
v[28][5] = 31;
v[29][5] = 25;
v[30][5] =  9;
v[31][5] = 49;
v[32][5] = 33;
v[33][5] = 19;
v[34][5] = 29;
v[35][5] = 11;
```

```
v[36][5] = 19;
v[37][5] = 27;
v[38][5] = 15;
v[39][5] = 25;

v[19][6] =  13;
v[20][6] =  35;
v[21][6] = 115;
v[22][6] =  41;
v[23][6] =  79;
v[24][6] =  17;
v[25][6] =  29;
v[26][6] = 119;
v[27][6] =  75;
v[28][6] =  73;
v[29][6] = 105;
v[30][6] =   7;
v[31][6] =  59;
v[32][6] =  65;
v[33][6] =  21;
v[34][6] =   3;
v[35][6] = 113;
v[36][6] =  61;
v[37][6] =  89;
v[38][6] =  45;
v[39][6] = 107;
```

```
    v[37][7] =  7;
    v[38][7] = 23;
    v[39][7] = 39;
//
//  Check parameters.
//
    if ( dim_num < 2 || DIM_MAX < dim_num )
    {
      cout << "\n";
      cout << "I4_SOBOL - Fatal error!\n";
      cout << "  The spatial dimension DIM_NUM should satisfy:\n";
      cout << "    2 <= DIM_NUM <= " << DIM_MAX << "\n";
      cout << "  But this input value is DIM_NUM = " << dim_num <<
"\n";
      exit ( 1 );
    }

    dim_num_save = dim_num;
//
//  Find the number of bits in ATMOST.
//
    maxcol = i4_bit_hi1 ( atmost );
//
//  Initialize row 1 of V.
//
    for ( j = 0; j < maxcol; j++ )
    {
```

```
      v[0][j] = 1;
    }
//
//  Initialize the remaining rows of V.
//
    for ( i = 1; i < dim_num; i++ )
    {
//
//  The bit pattern of the integer POLY(I) gives the form
//  of polynomial I.
//
//  Find the degree of polynomial I from binary encoding.
//
      j = poly[i];
      m = 0;

      while ( true )
      {
        j = j / 2;
        if ( j <= 0 )
        {
          break;
        }
        m = m + 1;
      }
//
//  We expand this bit pattern to separate components
```

```
//  of the logical array INCLUD.
//
      j = poly[i];
      for ( k = m-1; 0 <= k; k-- )
      {
        j2 = j / 2;
        includ[k] = ( j != ( 2 * j2 ) );
        j = j2;
      }
//
//  Calculate the remaining elements of row I as explained
//  in Bratley and Fox, section 2.
//
//  Some tricky indexing here.  Did I change it correctly?
//
      for ( j = m; j < maxcol; j++ )
      {
        newv = v[i][j-m];
        l = 1;

        for ( k = 0; k < m; k++ )
        {
          l = 2 * l;

          if ( includ[k] )
          {
            newv = ( newv ^ ( l * v[i][j-k-1] ) );
```

```
          }

        }

        v[i][j] = newv;

      }

    }
//
//  Multiply columns of V by appropriate power of 2.
//
    l = 1;
    for ( j = maxcol-2; 0 <= j; j-- )
    {
      l = 2 * l;
      for ( i = 0; i < dim_num; i++ )
      {
        v[i][j] = v[i][j] * l;
      }
    }
//
//  RECIPD is 1/(common denominator of the elements in V).
//
    recipd = 1.0E+00 / ( ( float ) ( 2 * l ) );
  }
```

```
if ( *seed < 0 )
{
  *seed = 0;
}

if ( *seed == 0 )
{
  l = 1;
  for ( i = 0; i < dim_num; i++ )
  {
    lastq[i] = 0;
  }
}
else if ( *seed == seed_save + 1 )
{
  l = i4_bit_lo0 ( *seed );
}
else if ( *seed <= seed_save )
{
  seed_save = 0;
  l = 1;
  for ( i = 0; i < dim_num; i++ )
  {
    lastq[i] = 0;
  }

  for ( seed_temp = seed_save; seed_temp <= (*seed)-1; seed_temp++ )
```

```
    {

      l = i4_bit_lo0 ( seed_temp );

      for ( i = 0; i < dim_num; i++ )
      {
        lastq[i] = ( lastq[i] ^ v[i][l-1] );
      }

    }

    l = i4_bit_lo0 ( *seed );
  }
  else if ( seed_save+1 < *seed )
  {
    for ( seed_temp = seed_save+1; seed_temp <= (*seed)-1; seed_temp++
)
    {

      l = i4_bit_lo0 ( seed_temp );

      for ( i = 0; i < dim_num; i++ )
      {
        lastq[i] = ( lastq[i] ^ v[i][l-1] );
      }

    }
```

```
    l = i4_bit_lo0 ( *seed );

  }
//
//  Check that the user is not calling too many times!
//

  if ( maxcol < l )
  {
    cout << "\n";
    cout << "I4_SOBOL - Fatal error!\n";
    cout << "  Too many calls!\n";
    cout << "  MAXCOL = " << maxcol << "\n";
    cout << "  L =      " << l << "\n";
    exit ( 2 );
  }
//
//  Calculate the new components of QUASI.
//  The caret indicates the bitwise exclusive OR.
//
  for ( i = 0; i < dim_num; i++ )
  {
    quasi[i] = ( ( float ) lastq[i] ) * recipd;

    lastq[i] = ( lastq[i] ^ v[i][l-1] );
  }
```

```
  seed_save = *seed;
  *seed = *seed + 1;

  return;
# undef MAX_DIM
}
//****************************************************************************
*********

int i4_uniform ( int b, int c, int *seed )

//****************************************************************************
*********
//
//  Purpose:
//
//    I4_UNIFORM returns an integer pseudorandom number.
//
//  Discussion:
//
//    The pseudorandom number should be uniformly distributed
//    between A and B.
//
//  Modified:
//
//    27 February 2005
```

```
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    Input, int B, C, the limits of the interval.
//
//    Input/output, int *SEED, the "seed" value, which should NOT be 0.
//    On output, SEED has been updated.
//
//    Output, int I4_UNIFORM, a number between A and B.
//
{
  double d;
  int value;

  if ( b <= c )
  {
    d = ( double ) ( b ) + ( double ) ( 1 + c - b ) * d_uniform_01 (
seed );

    value = ( int ) ( d );

    if ( value < b )
    {
```

```
      value = b;
    }
    if ( c < value )
    {
      value = c;
    }
  }
  else
  {
    d = ( double ) ( c ) + ( double ) ( 1 + b - c ) * d_uniform_01 (
seed );

    value = ( int ) ( d );

    if ( value < c )
    {
      value = c;
    }
    if ( b < value )
    {
      value = b;
    }
  }

  return value;
}
```

```
//**********************************************************************
*********

unsigned int i4_xor ( unsigned int i, unsigned int j )

//**********************************************************************
*********
//
//  Purpose:
//
//    I4_XOR calculates the exclusive OR of two integers.
//
//  Modified:
//
//    16 February 2005
//
//  Author:
//
//   John Burkardt
//
//  Parameters:
//
//    Input, unsigned int I, J, two values whose exclusive OR is
needed.
//
//    Output, unsigned int I4_XOR, the exclusive OR of I and J.
//
```

```
{
  unsigned int i2;
  unsigned int j2;
  unsigned int k;
  unsigned int l;

  k = 0;
  l = 1;

  while ( i != 0 || j != 0 )
  {
    i2 = i / 2;
    j2 = j / 2;

    if (
      ( ( i == 2 * i2 ) && ( j != 2 * j2 ) ) ||
      ( ( i != 2 * i2 ) && ( j == 2 * j2 ) ) )
    {
      k = k + l;
    }

    i = i2;
    j = j2;
    l = 2 * l;
  }

  return k;
```

```
}
//*********************************************************************
*********

int i8_bit_hi1 ( long int n )

//*********************************************************************
*********
//
//  Purpose:
//
//    I8_BIT_HI1 returns the position of the high 1 bit base 2 in an
integer.
//
//  Example:
//
//       N    Binary    Hi 1
//    ----    --------  ----
//       0           0     0
//       1           1     1
//       2          10     2
//       3          11     2
//       4         100     3
//       5         101     3
//       6         110     3
//       7         111     3
//       8        1000     4
```

```
//       9        1001     4
//      10        1010     4
//      11        1011     4
//      12        1100     4
//      13        1101     4
//      14        1110     4
//      15        1111     4
//      16       10000     5
//      17       10001     5
//    1023  1111111111    10
//    1024 10000000000    11
//    1025 10000000001    11
//
//  Modified:
//
//    03 August 2004
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    Input, long int N, the integer to be measured.
//    N should be nonnegative.  If N is nonpositive, I8_BIT_HI1
//    will always be 0.
//
```

```
//    Output, int I8_BIT_HI1, the number of bits base 2.
//
{
  int bit;

  bit = 0;

  while ( 0 < n )
  {
    bit = bit + 1;
    n = n / 2;
  }

  return bit;
}
//****************************************************************************
*********

int i8_bit_lo0 ( long int n )

//****************************************************************************
*********
//
//  Purpose:
//
//    I8_BIT_LO0 returns the position of the low 0 bit base 2 in an
integer.
```

```
//
//  Example:
//
//       N    Binary    Lo 0
//    ----    --------  ----
//       0           0     1
//       1           1     2
//       2          10     1
//       3          11     3
//       4         100     1
//       5         101     2
//       6         110     1
//       7         111     4
//       8        1000     1
//       9        1001     2
//      10        1010     1
//      11        1011     3
//      12        1100     1
//      13        1101     2
//      14        1110     1
//      15        1111     5
//      16       10000     1
//      17       10001     2
//    1023  1111111111     1
//    1024 10000000000     1
//    1025 10000000001     1
//
```

```
//  Modified:
//
//    03 August 2004
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    Input, long int N, the integer to be measured.
//    N should be nonnegative.
//
//    Output, int I8_BIT_LO0, the position of the low 1 bit.
//
{
  int bit;
  long int n2;

  bit = 0;

  while ( true )
  {
    bit = bit + 1;
    n2 = n / 2;

    if ( n == 2 * n2 )
```

```
    {
      break;
    }

    n = n2;

  }

  return bit;
}
//****************************************************************************
*********

void i8_sobol ( int dim_num, long int *seed, double quasi[ ] )

//****************************************************************************
*********
//
//  Purpose:
//
//    I8_SOBOL generates a new quasirandom Sobol vector with each call.
//
//  Discussion:
//
//    The routine adapts the ideas of Antonov and Saleev.
//
//  Modified:
```

```
//
//    03 August 2004
//
//  Reference:
//
//    Antonov and Saleev,
//    USSR Computational Mathematics and Mathematical Physics,
//    Volume 19, 1980, pages 252 - 256.
//
//    Paul Bratley and Bennett Fox,
//    Algorithm 659:
//    Implementing Sobol's Quasirandom Sequence Generator,
//    ACM Transactions on Mathematical Software,
//    Volume 14, Number 1, pages 88-100, 1988.
//
//    Bennett Fox,
//    Algorithm 647:
//    Implementation and Relative Efficiency of Quasirandom
//    Sequence Generators,
//    ACM Transactions on Mathematical Software,
//    Volume 12, Number 4, pages 362-376, 1986.
//
//    I Sobol,
//    USSR Computational Mathematics and Mathematical Physics,
//    Volume 16, pages 236-242, 1977.
//
//    I Sobol and Levitan,
```

```
//    The Production of Points Uniformly Distributed in a
Multidimensional
//    Cube (in Russian),
//    Preprint IPM Akad. Nauk SSSR,
//    Number 40, Moscow 1976.
//
//  Parameters:
//
//    Input, int DIM_NUM, the number of spatial dimensions.
//    DIM_NUM must satisfy 2 <= DIM_NUM <= 40.
//
//    Input/output, long int *SEED, the "seed" for the sequence.
//    This is essentially the index in the sequence of the quasirandom
//    value to be generated.  On output, SEED has been set to the
//    appropriate next value, usually simply SEED+1.
//    If SEED is less than 0 on input, it is treated as though it were
0.
//    An input value of 0 requests the first (0-th) element of the
sequence.
//
//    Output, double QUASI[DIM_NUM], the next quasirandom vector.
//
{
# define DIM_MAX 40

  static long int atmost = 4611686018427387903;
  static int dim_num_save = 0;
```

```
  long int i;
  bool includ[8];
  static bool initialized = false;
  long int j;
  long int j2;
  long int k;
  long int l;
  static long int lastq[DIM_MAX];
  long int m;
  static long int maxcol;
  long int newv;
  static long int poly[DIM_MAX] =
  {
        1,   3,   7,  11,  13,  19,  25,  37,  59,  47,
       61,  55,  41,  67,  97,  91, 109, 103, 115, 131,
      193, 137, 145, 143, 241, 157, 185, 167, 229, 171,
      213, 191, 253, 203, 211, 239, 247, 285, 369, 299
  };
  static double recipd;
  static long int seed_save = 0;
  long int seed_temp;
  static long int v[DIM_MAX][30];

  if ( !initialized || dim_num != dim_num_save )
  {
    initialized = true;
//
```

```
//  Initialize (part of) V.
//
    v[ 0][0] = 1;
    v[ 1][0] = 1;
    v[ 2][0] = 1;
    v[ 3][0] = 1;
    v[ 4][0] = 1;
    v[ 5][0] = 1;
    v[ 6][0] = 1;
    v[ 7][0] = 1;
    v[ 8][0] = 1;
    v[ 9][0] = 1;
    v[10][0] = 1;
    v[11][0] = 1;
    v[12][0] = 1;
    v[13][0] = 1;
    v[14][0] = 1;
    v[15][0] = 1;
    v[16][0] = 1;
    v[17][0] = 1;
    v[18][0] = 1;
    v[19][0] = 1;
    v[20][0] = 1;
    v[21][0] = 1;
    v[22][0] = 1;
    v[23][0] = 1;
    v[24][0] = 1;
```

```
v[25][0] = 1;
v[26][0] = 1;
v[27][0] = 1;
v[28][0] = 1;
v[29][0] = 1;
v[30][0] = 1;
v[31][0] = 1;
v[32][0] = 1;
v[33][0] = 1;
v[34][0] = 1;
v[35][0] = 1;
v[36][0] = 1;
v[37][0] = 1;
v[38][0] = 1;
v[39][0] = 1;

v[ 2][1] = 1;
v[ 3][1] = 3;
v[ 4][1] = 1;
v[ 5][1] = 3;
v[ 6][1] = 1;
v[ 7][1] = 3;
v[ 8][1] = 3;
v[ 9][1] = 1;
v[10][1] = 3;
v[11][1] = 1;
v[12][1] = 3;
```

```
v[13][1] = 1;
v[14][1] = 3;
v[15][1] = 1;
v[16][1] = 1;
v[17][1] = 3;
v[18][1] = 1;
v[19][1] = 3;
v[20][1] = 1;
v[21][1] = 3;
v[22][1] = 1;
v[23][1] = 3;
v[24][1] = 3;
v[25][1] = 1;
v[26][1] = 3;
v[27][1] = 1;
v[28][1] = 3;
v[29][1] = 1;
v[30][1] = 3;
v[31][1] = 1;
v[32][1] = 1;
v[33][1] = 3;
v[34][1] = 1;
v[35][1] = 3;
v[36][1] = 1;
v[37][1] = 3;
v[38][1] = 1;
v[39][1] = 3;
```

```
v[ 3][2] = 7;
v[ 4][2] = 5;
v[ 5][2] = 1;
v[ 6][2] = 3;
v[ 7][2] = 3;
v[ 8][2] = 7;
v[ 9][2] = 5;
v[10][2] = 5;
v[11][2] = 7;
v[12][2] = 7;
v[13][2] = 1;
v[14][2] = 3;
v[15][2] = 3;
v[16][2] = 7;
v[17][2] = 5;
v[18][2] = 1;
v[19][2] = 1;
v[20][2] = 5;
v[21][2] = 3;
v[22][2] = 3;
v[23][2] = 1;
v[24][2] = 7;
v[25][2] = 5;
v[26][2] = 1;
v[27][2] = 3;
v[28][2] = 3;
```

```
v[29][2] = 7;
v[30][2] = 5;
v[31][2] = 1;
v[32][2] = 1;
v[33][2] = 5;
v[34][2] = 7;
v[35][2] = 7;
v[36][2] = 5;
v[37][2] = 1;
v[38][2] = 3;
v[39][2] = 3;

v[ 5][3] =  1;
v[ 6][3] =  7;
v[ 7][3] =  9;
v[ 8][3] = 13;
v[ 9][3] = 11;
v[10][3] =  1;
v[11][3] =  3;
v[12][3] =  7;
v[13][3] =  9;
v[14][3] =  5;
v[15][3] = 13;
v[16][3] = 13;
v[17][3] = 11;
v[18][3] =  3;
v[19][3] = 15;
```

```
v[20][3] =  5;
v[21][3] =  3;
v[22][3] = 15;
v[23][3] =  7;
v[24][3] =  9;
v[25][3] = 13;
v[26][3] =  9;
v[27][3] =  1;
v[28][3] = 11;
v[29][3] =  7;
v[30][3] =  5;
v[31][3] = 15;
v[32][3] =  1;
v[33][3] = 15;
v[34][3] = 11;
v[35][3] =  5;
v[36][3] =  3;
v[37][3] =  1;
v[38][3] =  7;
v[39][3] =  9;

v[ 7][4] =  9;
v[ 8][4] =  3;
v[ 9][4] = 27;
v[10][4] = 15;
v[11][4] = 29;
v[12][4] = 21;
```

```
v[13][4] = 23;
v[14][4] = 19;
v[15][4] = 11;
v[16][4] = 25;
v[17][4] =  7;
v[18][4] = 13;
v[19][4] = 17;
v[20][4] =  1;
v[21][4] = 25;
v[22][4] = 29;
v[23][4] =  3;
v[24][4] = 31;
v[25][4] = 11;
v[26][4] =  5;
v[27][4] = 23;
v[28][4] = 27;
v[29][4] = 19;
v[30][4] = 21;
v[31][4] =  5;
v[32][4] =  1;
v[33][4] = 17;
v[34][4] = 13;
v[35][4] =  7;
v[36][4] = 15;
v[37][4] =  9;
v[38][4] = 31;
v[39][4] =  9;
```

```
v[13][5] = 37;
v[14][5] = 33;
v[15][5] =  7;
v[16][5] =  5;
v[17][5] = 11;
v[18][5] = 39;
v[19][5] = 63;
v[20][5] = 27;
v[21][5] = 17;
v[22][5] = 15;
v[23][5] = 23;
v[24][5] = 29;
v[25][5] =  3;
v[26][5] = 21;
v[27][5] = 13;
v[28][5] = 31;
v[29][5] = 25;
v[30][5] =  9;
v[31][5] = 49;
v[32][5] = 33;
v[33][5] = 19;
v[34][5] = 29;
v[35][5] = 11;
v[36][5] = 19;
v[37][5] = 27;
v[38][5] = 15;
```

```
v[39][5] = 25;


v[19][6] =  13;
v[20][6] =  35;
v[21][6] = 115;
v[22][6] =  41;
v[23][6] =  79;
v[24][6] =  17;
v[25][6] =  29;
v[26][6] = 119;
v[27][6] =  75;
v[28][6] =  73;
v[29][6] = 105;
v[30][6] =   7;
v[31][6] =  59;
v[32][6] =  65;
v[33][6] =  21;
v[34][6] =   3;
v[35][6] = 113;
v[36][6] =  61;
v[37][6] =  89;
v[38][6] =  45;
v[39][6] = 107;


v[37][7] =  7;
v[38][7] = 23;
v[39][7] = 39;
```

```
//
//  Check parameters.
//
    if ( dim_num < 2 || DIM_MAX < dim_num )
    {
      cout << "\n";
      cout << "I8_SOBOL - Fatal error!\n";
      cout << "  The spatial dimension DIM_NUM should satisfy:\n";
      cout << "    2 <= DIM_NUM <= " << DIM_MAX << "\n";
      cout << "  But this input value is DIM_NUM = " << dim_num <<
"\n";
      exit ( 1 );
    }

    dim_num_save = dim_num;
//
//  Find the number of bits in ATMOST.
//
    maxcol = i8_bit_hi1 ( atmost );
//
//  Initialize row 1 of V.
//
    for ( j = 0; j < maxcol; j++ )
    {
      v[0][j] = 1;
    }
//
```

```
//  Initialize the remaining rows of V.
//
    for ( i = 1; i < dim_num; i++ )
    {
//
//  The bit pattern of the integer POLY(I) gives the form
//  of polynomial I.
//
//  Find the degree of polynomial I from binary encoding.
//
      j = poly[i];
      m = 0;

      while ( true )
      {
        j = j / 2;
        if ( j <= 0 )
        {
          break;
        }
        m = m + 1;
      }
//
//  We expand this bit pattern to separate components
//  of the logical array INCLUD.
//
      j = poly[i];
```

```
      for ( k = m-1; 0 <= k; k-- )
      {
        j2 = j / 2;
        includ[k] = ( j != ( 2 * j2 ) );
        j = j2;
      }
//
//  Calculate the remaining elements of row I as explained
//  in Bratley and Fox, section 2.
//
//  Some tricky indexing here.  Did I change it correctly?
//
      for ( j = m; j < maxcol; j++ )
      {
        newv = v[i][j-m];
        l = 1;

        for ( k = 0; k < m; k++ )
        {
          l = 2 * l;

          if ( includ[k] )
          {
            newv = ( newv ^ ( l * v[i][j-k-1] ) );
          }

        }
```

```
        v[i][j] = newv;

      }

    }
//
//  Multiply columns of V by appropriate power of 2.
//
    l = 1;
    for ( j = maxcol - 2; 0 <= j; j-- )
    {
      l = 2 * l;
      for ( i = 0; i < dim_num; i++ )
      {
        v[i][j] = v[i][j] * l;
      }
    }
//
//  RECIPD is 1/(common denominator of the elements in V).
//
    recipd = 1.0E+00 / ( ( double ) ( 2 * l ) );
  }

  if ( *seed < 0 )
  {
    *seed = 0;
```

```
}

if ( *seed == 0 )
{
  l = 1;
  for ( i = 0; i < dim_num; i++ )
  {
    lastq[i] = 0;
  }
}
else if ( *seed == seed_save + 1 )
{
  l = i8_bit_lo0 ( *seed );
}
else if ( *seed <= seed_save )
{
  seed_save = 0;
  l = 1;
  for ( i = 0; i < dim_num; i++ )
  {
    lastq[i] = 0;
  }

  for ( seed_temp = seed_save; seed_temp <= (*seed)-1; seed_temp++ )
  {

    l = i8_bit_lo0 ( seed_temp );
```

```
      for ( i = 0; i < dim_num; i++ )
      {
        lastq[i] = ( lastq[i] ^ v[i][l-1] );
      }

    }

    l = i8_bit_lo0 ( *seed );
  }
  else if ( seed_save+1 < *seed )
  {
    for ( seed_temp = seed_save+1; seed_temp <= (*seed)-1; seed_temp++
)
    {

      l = i8_bit_lo0 ( seed_temp );

      for ( i = 0; i < dim_num; i++ )
      {
        lastq[i] = ( lastq[i] ^ v[i][l-1] );
      }

    }

    l = i8_bit_lo0 ( *seed );
```

```
  }
//
//  Check that the user is not calling too many times!
//
  if ( maxcol < l )
  {
    cout << "\n";
    cout << "I8_SOBOL - Fatal error!\n";
    cout << "  Too many calls!\n";
    cout << "  MAXCOL = " << maxcol << "\n";
    cout << "  L =      " << l << "\n";
    exit ( 2 );
  }
//
//  Calculate the new components of QUASI.
//  The caret indicates the bitwise exclusive OR.
//
  for ( i = 0; i < dim_num; i++ )
  {
    quasi[i] = ( ( double ) lastq[i] ) * recipd;

    lastq[i] = ( lastq[i] ^ v[i][l-1] );
  }

  seed_save = *seed;
  *seed = *seed + 1;
```

```
  return;
# undef MAX_DIM
}
//****************************************************************************
*********

long int i8_uniform ( long int b, long int c, int *seed )

//****************************************************************************
*********
//
//  Purpose:
//
//    I8_UNIFORM returns an integer pseudorandom number.
//
//  Discussion:
//
//    The pseudorandom number should be uniformly distributed
//    between A and B.
//
//  Modified:
//
//    27 February 2005
//
//  Author:
//
//    John Burkardt
```

```
//
//  Parameters:
//
//    Input, long int B, C, the limits of the interval.
//
//    Input/output, int *SEED, the "seed" value, which should NOT be 0.
//    On output, SEED has been updated.
//
//    Output, long int I8_UNIFORM, a number between A and B.
//
{
  double d;
  long int value;

  if ( b <= c )
  {
    d = ( double ) ( b ) + ( double ) ( 1 + c - b ) * d_uniform_01 (
seed );

    value = ( long int ) ( d );

    if ( value < b )
    {
      value = b;
    }
    if ( c < value )
    {
```

```
      value = c;
    }
  }
  else
  {
    d = ( double ) ( c ) + ( double ) ( 1 + b - c ) * d_uniform_01 (
seed );

    value = ( long int ) ( d );

    if ( value < c )
    {
      value = c;
    }
    if ( b < value )
    {
      value = b;
    }
  }

  return value;
}
//****************************************************************************
*********

unsigned long int i8_xor ( unsigned long int i, unsigned long int j )
```

```
//**********************************************************************
*********
//
//  Purpose:
//
//    I8_XOR calculates the exclusive OR of two integers.
//
//  Modified:
//
//    16 February 2005
//
//  Author:
//
//   John Burkardt
//
//  Parameters:
//
//    Input, unsigned long int I, J, two values whose exclusive OR is
needed.
//
//    Output, unsigned long int I8_XOR, the exclusive OR of I and J.
//
{
  unsigned long int i2;
  unsigned long int j2;
  unsigned long int k;
  unsigned long int l;
```

```
  k = 0;
  l = 1;

  while ( i != 0 || j != 0 )
  {
    i2 = i / 2;
    j2 = j / 2;

    if (
      ( ( i == 2 * i2 ) && ( j != 2 * j2 ) ) ||
      ( ( i != 2 * i2 ) && ( j == 2 * j2 ) ) )
    {
      k = k + l;
    }

    i = i2;
    j = j2;
    l = 2 * l;
  }

  return k;
}
//****************************************************************************
*

void timestamp ( void )
```

```
//****************************************************************************
*
//
//  Purpose:
//
//    TIMESTAMP prints the current YMDHMS date as a time stamp.
//
//  Example:
//
//    May 31 2001 09:45:54 AM
//
//  Modified:
//
//    04 October 2003
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    None
//
{
#define TIME_SIZE 40
```

```
  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;
            ofstream FinalReport("GrowthReport.txt", ios::app);
  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm
);
      cout << time_buffer << "\n";
  FinalReport << time_buffer << "\n";

  return;
#undef TIME_SIZE
}
int get_seed ( void )

//**********************************************************************
*********
//
//  Purpose:
//
//    GET_SEED returns a random seed for the random number generator.
//
//  Modified:
//
```

```
//    17 November 2004
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    Output, int GET_SEED, a random seed value.
//
{
# define I_MAX 2147483647
  time_t clock;
  int i;
  int ihour;
  int imin;
  int isec;
  int seed;
  struct tm *lt;
  time_t tloc;
//
//  If the internal seed is 0, generate a value based on the time.
//
  clock = time ( &tloc );
  lt = localtime ( &clock );
//
//  Hours is 1, 2, ..., 12.
```

```
//
  ihour = lt->tm_hour;

  if ( 12 < ihour )
  {
    ihour = ihour - 12;
  }
//
//  Move Hours to 0, 1, ..., 11
//
  ihour = ihour - 1;

  imin = lt->tm_min;

  isec = lt->tm_sec;

  seed = isec + 60 * ( imin + 60 * ihour );
//
//  We want values in [1,43200], not [0,43199].
//
  seed = seed + 1;
//
//  Remap ISEED from [1,43200] to [1,IMAX].
//
  seed = ( int )
    ( ( ( double ) seed )
    * ( ( double ) I_MAX ) / ( 60.0E+00 * 60.0E+00 * 12.0E+00 ) );
```

```
//
//  Never use a seed of 0.
//
  if ( seed == 0 )
  {
    seed = 1;
  }

  return seed;
# undef I_MAX
}
//**********************************************************************
*********

void timeStep(double *time, float *pdf, double *area, int *trans)
{
      ofstream Timetable("time.txt", ios::app);
      int seed;
      double rand;
      double sum;
      double ln;
      double current_time;
      double time_step;

      current_time = *time;

      seed = get_seed();
```

```
        rand = d_uniform_01 ( &seed );

        sum = (trans[0]*pdf[0]) + (trans[1]*pdf[1]) + (trans[2]*pdf[2]);

        ln = (-log(rand));

        time_step = ln / sum;

        *time =  current_time + time_step;
        Timetable <<"-----------------"<<endl;
        Timetable <<"Current Time = "<<current_time<<endl;
        Timetable <<"Time Step = " << ln/sum << endl;
        Timetable <<"Transition took = "<<*time <<endl;
        Timetable <<"Rand = "<<rand<<endl;
        Timetable << "Pads = " << pdf[0] << ", Pdes = " << pdf[1] << ",
Pdif = " << pdf[2] << endl;
        Timetable <<"Sum of pdf = "<<sum <<endl;
        Timetable <<"-Ln(rand) = "<<ln<<endl;



}

void pdfSelection(float *pdf, int *selection, int *tran)
{

        int seed;
```

```
	seed = get_seed();
	double Urand = d_uniform_01 ( &seed );

	double total = pdf[0] + pdf[1] + pdf[2];
	double no_event = 0.0 * total;
	total = pdf[0] + pdf[1] + pdf[2] + no_event;
	*selection = -1;

	if( Urand < (pdf[0]/total))
	{
		*selection = 0;
		tran[0]++;
	}

	else if( (Urand >= (pdf[0]/total)) && (Urand < ((pdf[0] + pdf[1])
/ total)) )
	{
		*selection = 1;
		tran[1]++;
	}

	else if( (Urand >= ((pdf[0] + pdf[1]) / total)) && (Urand <
((pdf[0] + pdf[1]+ pdf[2]) / total)) )
	{
		*selection = 2;
		tran[2]++;
	}
```

```
	else
	{
		*selection = -1;
	}
	//cout<<"selection is: "<<*selection<<endl;
}

void pdfVacantSelect(float *pdf, int *selection, int *tran)
{

	int seed;
	double no_event = pdf[0] * 0.1;
	double stick = pdf[0] + no_event;
	seed = get_seed();
	double Urand = d_uniform_01 ( &seed );
	*selection = -1;

	if(Urand < (pdf[0] / stick))
	{
		*selection = 0;
		tran[0]++;
	}
	else
		*selection = -1;
/*
	if(pdf[0] <= 0.5)
	{
```

```
            *selection = -1;
      }*/
}
void site(int *seed, int x, int y, int *out)
{
      float r[2];
      int dim_num = 2;
      int num;
      //static int u_seed  = 0;

      //double Urand_x,Urand_y;
      num = *seed;

      //if(u_seed == 0)
      //    u_seed = get_seed();

    i4_sobol ( dim_num, &num, r );
      //cout << "site obtained" << endl;
      //u_seed++;
      //Urand_x = d_uniform_01 ( &u_seed );
      //u_seed++;
      //Urand_y = d_uniform_01 ( &u_seed );

      //num = num + 2;
    *seed = num;

      //out[0] = Urand_x * x; //r[0] * x;
```

```
	//out[1] = Urand_y * y; //r[1] * y;
	out[0] = r[0] * x;
	out[1] = r[1] * y;
}
void siteUpdate(int ***Vol, int *coord, int *selection, int *index,int
count)
{
	//cout << "Entering Site Update"<<endl;
	//siteUpdate(volume, coord, &select, index,adjacent);
	int choice;		//Process to be preformed on the surface
site
	int i;			//index variable
	int x,y,z;		//chosen site on surface
	int seed;		//seed used for random number
	double Urand;		//Random number used to pick which
neighbor start diffusion with
	int start = 0;		//Shows which neighbor to diffuse
	int diffused = 0; //used to show when diffusion had occured
	int next_door = 0;	//Number of neighboring atoms

	choice = *selection;	//save slection for this itteration
	*selection = -1;	//Reset selection for next iterration

	x = coord[0];
	y = coord[1];
	z = coord[2];
	//cout<<"z: "<<z<<endl;
```

```
seed = get_seed();
Urand = d_uniform_01( &seed );




//cout<<"Choice was: "<<choice <<", Passed RNG"<<endl;



switch(choice)
{
	case -1://No transition occured on site
	{

		break;
	}
	case 0://Adsorption occured
	{

		Vol[x][y][z] = 1;
		break;

	}
	case 1://Desorption occured
	{
		Vol[x][y][z] = 0;
```

```
		break;
	}
	case 2://Diffusion occured
	{
		if(Urand < .25)
		{
			start = 0;
		}
		if((Urand >= .25) && (Urand < .5))
		{
			start = 1;
		}
		if((Urand >= .5) && (Urand < .75))
		{
			start = 2;
		}
		if((Urand >= .75) && (Urand <= 1))
		{
			start = 3;
		}
		//cout <<"Start was set to: "<<start<<endl;
		neighbors(Vol,coord,index,&next_door);
		if(next_door < 4)
		{
			for(i = 0; i <=3; i++)
			{
				while((diffused == 0) )
```

```
{
	if(start == 0)
	{
		if(Vol[x][y + 2][z] == 0)
		{
			Vol[x][y + 2][z] = 1;
			Vol[x][y][z] = 0;
			diffused = 1;
		}
	}

	if(start == 1)
	{
		if(Vol[x + 2][y][z] == 0)
		{
			Vol[x + 2][y][z] = 1;
			Vol[x][y][z] = 0;
			diffused = 1;
		}
	}

	if(start == 2)
	{
		if(Vol[x][y - 2][z] == 0)
		{
			Vol[x][y - 2][z] = 1;
			Vol[x][y][z] = 0;
```

```
                        diffused = 1;
                    }
                }

                if(start == 3)
                {
                    if(Vol[x - 2][y][z] == 0)
                    {
                        Vol[x - 2][y][z] = 1;
                        Vol[x][y][z] = 0;
                        diffused = 1;
                    }
                }

                start++;
                if(start == 4)
                {
                    start = 0;
                }
            }
        }
    }
    else
    {
        Vol[x][y][z] = 1;
    }
    break;
```

```
            }
            default://Error
            {
                  cout<<"...Invalid Selection in PDF...."<<endl;
            }
      }
}
/*********************************************************
Valid Site will take the max indicies of the SUT
and return a site that falls within the domain of the SUT
that falls within the valid site domain
*********************************************************/
void ValidSite(int *seed,int xs, int ys, int zs, int
*nano,int*coord,int***Volume, float standard,int **history, int
history_length)
{
      int SUT_coord[3] = {-1,-1,-1};            //0 - x, 1 - y, 2 - z
      int valid = 0;                //set to 1 when valid site has been
found
      int x,y,z;                    //Valid sites chosen and returned
      int i,j,k,l,q;                      //indexing variable
      int x_nano, y_nano,r_nano;          //nano window dimension
      int offset_x, offset_y,offset_r; //offsets used to center the
nano window on SUT
      int x_low, x_limit;           //limits for valid x coordinate
      int y_low, y_limit;           //limits for valid y coordinate
      int x_range, y_range;   //diffrence between low and limit
```

```
	int r_low, r_limit;
	int SUT_r,xr, SUT_rx, SUT_ry;
	int circular = 1;
	//    ofstream Report("SobolCalls.txt", ios::app);
	//x_nano = xs / 2;
	//y_nano = ys / 2;

	x_nano = nano[0];
	y_nano = nano[1];

/*
	offset_x = (xs - (x_nano * 8))/2;
	offset_y = (ys - (y_nano * 8))/2;
*/
	offset_x = (xs - (x_nano * (10-standard)))/2;
	offset_y = (ys - (y_nano * (10-standard)))/2;

	r_nano =xs/2 - offset_x;

	xr = xs/2;

	offset_r = xr - r_nano ;

	x_low = offset_x;
	x_limit = xs - offset_x;
	x_range = x_limit - x_low;
```

```
	y_low = offset_y;
	y_limit = ys - offset_y;
	y_range = y_limit -  y_low;

	r_low =offset_r;
	r_limit = xr - offset_r;

	//cout << " x low = "<< x_low << ", x high = " << x_limit <<
endl;
	//cout << " y low = "<< y_low << ", y high = " << y_limit <<
endl;
	k=0;
	l=0;
	while(valid == 0)
	{

	if(circular == 1)
	{
		SUT_rx = abs(SUT_coord[0] - xr);
		SUT_ry = abs(SUT_coord[1] - xr);
		SUT_r = sqrt(double (SUT_rx*SUT_rx + SUT_ry*SUT_ry));
		//cout<<"SUT_r: "<<SUT_r<<endl;
		//cout<<"r low: "<<r_low<<", r limit: "<<r_limit<<endl;
		while( !( (SUT_r < r_limit) &&
				(SUT_coord[0] > x_low) && (SUT_coord[0] <
x_limit)
```

```
                    && (SUT_coord[1] > y_low) && (SUT_coord[1] <
y_limit) ) )
            {
                  l++;
                  //Obtains a site from the Sobol generator
                  //function will be called untill a point is selected
                  //that falls within the valid region


                  site(seed, x_range, y_range, SUT_coord);

                  SUT_coord[0] = SUT_coord[0] + x_low;
                  SUT_coord[1] = SUT_coord[1] + y_low;

                  SUT_rx = abs(SUT_coord[0] - xs/2);
                  SUT_ry = abs(SUT_coord[1] - ys/2);

                  SUT_r = sqrt(double(SUT_rx*SUT_rx + SUT_ry*SUT_ry));

                  //Report << "Sobol Call #" << l <<endl;
                  //Report << "x = "<< SUT_coord[0] << ", y = " <<
SUT_coord[1] << endl;

            }
      }
      else
      {
```

```
            while( !(((SUT_coord[0] > x_low) && (SUT_coord[0] <
x_limit)) &&
                  ((SUT_coord[1] > y_low) && (SUT_coord[1] < y_limit)))
)
            {
                  l++;
                  //Obtains a site from the Sobol generator
                  //function will be called untill a point is selected
                  //that falls within the valid region

                  site(seed, xs, ys, SUT_coord);
                  //cout << "Sobol Call #" << l <<endl;
                  //cout << "x = "<< SUT_coord[0] << ", y = " <<
SUT_coord[1] << endl;

            }
      }
            l = 0;
            x = SUT_coord[0];
            y = SUT_coord[1];

            SUT_coord[0] = -1;
            SUT_coord[1] = -1;

            //cout << "Valid x = " << x <<", y = " << y <<endl;

            i = 0;
```

```
            while( (Volume[x][y][i] != 0) && (i <= zs) )
            {
                  //cout << "Top of Stack Trial #" << k <<endl;
                  //cout << "Array value = " << Volume[x][y][i] <<
endl;
                  /*Possible Limitation
                  program will only allow you to find an
                  avaliable spot that is within the size of the Volume
array
                  created to model this growth, if this x,y coordinate
is occupied
                  to the top of the Volume array, another x,y
coordinate will be found
                  and this site will be skipped*/
                  k++;
                  i++;
            }
            k = 0;
            //cout << "i = " << i <<", zs = "<< zs << endl;
            if( i <= zs)
            {
                  valid = 1;
                  z = i;
            }

            for(q = 0; q < history_length; q++)
```

```
            {
                  if( (history[q][0] == x) && (history[q][1] == y))
                        valid = 0;
            }


      }
            coord[0] = x;
            coord[1] = y;
        coord[2] = z;

//          Report.close();
}

#include "island_header.h"


void main(void)
{
      int **map;
      int x, y;
      int total_islands;
      char choice;
      node list;

//Create map from text file
      map = create_map(&x,&y);
```

```
//Normalize map to 0's and -1's
      cout << "Normalizing Map...."<<endl;

      normalize_map(map,x,y);
      cout << "... Normalization Complete"<<endl;

      array_out(map, x, y);

      //cout << "Press any key to continue, The normalized map can be
reviewed ";
      //cin >> choice;
      //cout << "\n" << endl;

//Connects normalized map into islands and returns the total
// number of islands in map

      cout <<"Connecting Islands ....."<<endl;
      total_islands = connect_map(map,x,y);
      cout <<" ..... Islands Connected"<<endl;

      //array_out(map, x, y);

      cout << "Total number of islands: "<<total_islands<<endl;

      //cout << "Press any key to continue, The connected map can be
reviewed ";
```

```
	//cin >> choice;
	//cout << "\n" << endl;

//Create linked list of islands

	list = (node) malloc(total_islands * sizeof(island));

	link_list(list,total_islands);

//Save max and min values and Calculate Diameters in Linked List

	max_and_min(map,list,total_islands,x,y);

//Ouput Linked List for review

	linked_list_out(list,total_islands);

//
}

#include <iostream>


using std::cout;
using std::cin;
using std::ios;
using std::cerr;
```

```
using std:: endl;

#include <fstream>

using std::ofstream;
#include <iomanip>   // format manipulation
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

using namespace std;

struct data
{
      int x_max;
      int x_low;
      int y_max;
      int y_low;
      float diameter;

      struct data *next;
};

typedef struct data * node;
typedef struct data island;

void save_x_max(node ptr, int x)
```

```
{
	ptr->x_max = x;
}

void save_y_max(node ptr, int y)
{
	ptr->y_max = y;
}

void save_x_low(node ptr, int x)
{
	ptr->x_low = x;
}

void save_y_low(node ptr, int y)
{
	ptr->y_low = y;
}
void save_diameter(node ptr, int d)
{
	ptr->diameter = d;
}

void link_list(node list,int length)
{
	int i;
```

```
	for(i = 0; i < (length - 1); i++)
	{
		list[i].next = &list[i+1];
	}

	list[i].next = NULL;
}
void max_and_min(int **map,node list,int max_islands,int x,int y)
{
	int i,j;
	int islands = 0;
	int max_x, min_x;
	int max_y, min_y;
	int diameter;


	while(islands < max_islands)
	{
		max_x = 0;
		max_y = 0;
		min_x = x;
		min_y = y;
		diameter = 0;

		for(i = 0; i < x; i++)
		{
			for(j = 0; j < y; j++)
```

```
                {
                        if(map[i][j] == islands)
                        {
                                if( (i < min_x) )
                                        min_x = i;
                                if( (i > max_x) )
                                        max_x = i;
                                if( (j < min_y) )
                                        min_y = j;
                                if( (j > max_y) )
                                        max_y = j;
                        }
                }

        }

        diameter = ( (max_x - min_x + 1) + (max_y - min_y + 1) ) /
2;

        save_diameter(&list[islands],diameter);
        save_x_max(&list[islands],max_x);
        save_y_max(&list[islands],max_y);
        save_x_low(&list[islands],min_x);
        save_y_low(&list[islands],min_y);
```

```
			islands++;
		}

}

void linked_list_out(node list,int length)
{

			ofstream outData("linked_list.txt", ios::out);
		int i;

			outData << "X MAX \t X MIN \t Y MAX \t Y MIN \t
DIAMETER"<<endl;
			for(i = 0; i < length; i++)
			{
				outData << list[i].x_max <<"\t" << list[i].x_low
<<"\t"<<list[i].y_max<<"\t"<<list[i].y_low<<"\t"<<list[i].diameter<<end
l;
			}

			outData.close();

}
int** array2Dcreate(int x,int y)
{
		int i, j;
		int ** array2D = new int*[x];
```

```
	for(i = 0; i < x; i++)
	{
		array2D[i] = new int[y];
	}

	return array2D;
}

void array2Dinitial( int **Vol, int x, int y)
{

	for(int i = 0; i < x; i++)
	{
		for(int j = 0; j < y; j++)
		{
			Vol[i][j] = 0;

		}
	}
}
int ** create_map(int *dimx, int *dimy)
{
	ifstream fp_in;
	ofstream fp_out;

	fp_in.open("surface.dat",ios::in);
```

```
int **map;                      //Holds data map from text file
  int x, y;                     //size of data map
  int i,j,k,trash;        //index variables

  char line[80];

  //Stripping Data from file

  fp_in >> line;

  //Size of Grid
  fp_in >> x;
  fp_in >> y;

  cout << "Current "<<line<<" is set to "<<x<<" by "<<y<<endl;
  *dimx = x;
  *dimy = y;

  //Removing unwanted characters from file
fp_in >>trash;
        cout << trash<<"\t";
  fp_in >>trash;
        cout << trash<<"\t";
  fp_in >>trash;
        cout << trash<<"\t";
  fp_in >>trash;
```

```
		cout << trash<<"\t";
	fp_in >>trash;
		cout << trash << endl;

	//Creating and intializing array for map

	map = array2Dcreate(x ,y );
	array2Dinitial( map, x, y);

	//Stripping values from .dat to array
	for(i = 0; i < x; i++)
	{
		for(j = 0; j < y; j++)
		{
			fp_in >> k;

			map[i][j] = k;

			//cout << in[i][j];
		}
		//cout <<"\n";
	}

	fp_in.close();

	return(map);
}
```

```
void normalize_map(int **map,int x,int y)
{
	//normalize 2D array to 0's and -1's
	//Place an -1 where data is found, everywhere else will be a 0

	int i,j,temp;

	for(i = 0; i < x; i++)
	{
		for(j = 0; j < y; j++)
		{
			temp = map[i][j];
			if(temp != 0)
				map[i][j] = -1;

		}
	}
}
void array_out(int **output, int x, int y)
{
	int i,j;

	ofstream outData("island.txt", ios::out);
	outData <<"GRID\t"<<x<<"\t"<<y<<endl;
	outData <<"-5\t1\t1\t0\t0"<<endl;
	output[0][0]=0;
	for(i = 0; i < x; i++)
```

```
	{
		for(j = 0; j < y; j++)
		{
			outData  << output[i][j] <<  "\t";
		}
		outData <<endl;
	}

	outData.close();
}
void connections(int *island, int **map,int x, int y)
{

	island[0] = map[x-1][y];
	island[1] = map[x-1][y+1];
	island[2] = map[x][y+1];
	island[3] = map[x+1][y+1];
	island[4] = map[x+1][y];
	island[5] = map[x+1][y-1];
	island[6] = map[x][y-1];
	island[7] = map[x-1][y-1];



}
int straight(int *island)
{
	int test = 0;
```

```
if((island[0] > 0) || (island[0] == -1))
{
      if( (test > 0) && (island[0] == -1) )
            test = test;

      if( (test > 0) && (island[0] > 0) )
            test = island[0];

      if( (test < 0) && (island[0] > 0) )
            test = island[0];

      if( (test < 0) && (island[0] == -1) )
            test = island[0];

      if(test == 0)
            test = island[0];
}



if((island[2] > 0) || (island[2] == -1))
{
      if( (test > 0) && (island[2] == -1) )
            test = test;

      if( (test > 0) && (island[2] > 0) )
```

```
            test = island[2];

      if( (test < 0) && (island[2] > 0) )
            test = island[2];

      if( (test < 0) && (island[2] == -1) )
            test = island[2];

      if(test == 0)
            test = island[2];
}


if((island[4] > 0) || (island[4] == -1))
{
      if( (test > 0) && (island[4] == -1) )
            test = test;

      if( (test > 0) && (island[4] > 0) )
            test = island[4];

      if( (test < 0) && (island[4] > 0) )
            test = island[4];

      if( (test < 0) && (island[4] == -1) )
            test = island[4];
```

```
            if(test == 0)
                  test = island[4];
      }



      if((island[6] > 0) || (island[6] == -1))
      {
            if( (test > 0) && (island[6] == -1) )
                  test = test;

            if( (test > 0) && (island[6] > 0) )
                  test = island[6];

            if( (test < 0) && (island[6] > 0) )
                  test = island[6];

            if( (test < 0) && (island[6] == -1) )
                  test = island[6];

            if(test == 0)
                  test = island[6];
      }


      return(test);
}
```

```
int diagonal(int *island)
{
	int test = 0;


	if((island[1] > 0) || (island[1] == -1))
	{
		if( (test > 0) && (island[1] == -1) )
			test = test;

		if( (test > 0) && (island[1] > 0) )
			test = island[1];

		if( (test < 0) && (island[1] > 0) )
			test = island[1];

		if( (test < 0) && (island[1] == -1) )
			test = island[0];

		if(test == 0)
			test = island[1];
	}


	if((island[3] > 0) || (island[3] == -1))
	{
```

```
	if( (test > 0) && (island[3] == -1) )
		test = test;

	if( (test > 0) && (island[3] > 0) )
		test = island[3];

	if( (test < 0) && (island[3] > 0) )
		test = island[3];

	if( (test < 0) && (island[3] == -1) )
		test = island[3];

	if(test == 0)
		test = island[3];
}



if((island[5] > 0) || (island[5] == -1))
{
	if( (test > 0) && (island[5] == -1) )
		test = test;

	if( (test > 0) && (island[5] > 0) )
		test = island[5];

	if( (test < 0) && (island[5] > 0) )
```

```
            test = island[5];

      if( (test < 0) && (island[5] == -1) )
            test = island[5];

      if(test == 0)
            test = island[5];
}



if((island[7] > 0) || (island[7] == -1))
{
      if( (test > 0) && (island[7] == -1) )
            test = test;

      if( (test > 0) && (island[7] > 0) )
            test = island[7];

      if( (test < 0) && (island[7] > 0) )
            test = island[7];

      if( (test < 0) && (island[7] == -1) )
            test = island[7];

      if(test == 0)
            test = island[7];
```

```
	}

	return(test);
}
int d_valid(int *diag_valid,int **map,int x,int y)
{

	int test = 0;
	diag_valid[0] = 0;
	diag_valid[1] = 0;
	diag_valid[2] = 0;
	diag_valid[3] = 0;

	if( (map[x-1][y] != 0) || (map[x][y+1] != 0) )
	{
		diag_valid[0] = 1;
		test = 1;
	}

	if( (map[x+1][y] != 0) || (map[x][y+1] != 0) )
	{
		diag_valid[1] = 1;
		test = 1;
	}

	if( (map[x+1][y] != 0) || (map[x][y-1] != 0) )
	{
```

```
            diag_valid[2] = 1;
            test = 1;
      }

      if( (map[x-1][y] != 0) || (map[x][y-1] != 0) )
      {
            diag_valid[3] = 1;
            test = 1;
      }

      return(test);
}
void ext_side(int *side, int **map,int x,int y)
{
      *side = 0;

      if(map[x][y+1] == -1)
      {
            ext_side(side,map,x,y+1);
      }

      if(map[x-1][y] > 0)
            *side = map[x-1][y];

}
int connect_map(int **map,int x,int y)
{
```

```
	int i,j;
	int total_islands = 0;
	int neighbors[8] = {0,0,0,0,0,0,0,0};
	int diag_valid[4] = {0,0,0,0};
	int s_neigh, d_neigh, ext_s_neigh;
	int test;

	for(i = 0; i < x; i++)
	{
		for(j = 0; j < y; j++)
		{
			if(map[i][j] == -1)
			{
				if(total_islands == 0)
				{//first island
					total_islands++;
					map[i][j] = total_islands;
				}
				else
				{
					//Read values of non diaganol neighbors
					//		if any value other than 0 or
-1 is found
					//		change map value to the value
of neighbor
					//		otherwise, increment
total_islands and update
```

```
                        //          map to total_islands number

                        //Loads map value of neighboring sites
                        connections(neighbors,map,i,j);

                        s_neigh = straight(neighbors);

                        d_neigh = diagonal(neighbors);

                        if((s_neigh == 0))
                        {//single dot with no neighbors,
independent island
                              total_islands++;
                              map[i][j] = total_islands;
                        }

                        else if( (s_neigh == -1) && (d_neigh == -
1) )
                        {
                              ext_side(&ext_s_neigh ,map,i,j);

                              if(ext_s_neigh > 0)
                              {//Value is connected to island,
add to existing island
                                    //do not increment
                                    map[i][j] = ext_s_neigh;
                              }
```

```
                                    else
                                    {//new island, s_neigh and d_neigh
have no values
                                          //but island does have
s_neighbors
                                          total_islands++;
                                          map[i][j] = total_islands;
                                    }
                              }
                              else if( s_neigh != 0)
                              {//has a connection on straight side
                                    //must determine island number

                                    if(s_neigh > 0)
                                    {//takes number of island, but does
not increment
                                          //adds to pre existing island
                                          map[i][j] = s_neigh;
                                    }
                                    else if( s_neigh == -1)
                                    {//Have to make sure none of side
neighbors later in the
                                                //island have already
been calculated
                                          ext_side(&ext_s_neigh
,map,i,j);
```

```
                                        if(ext_s_neigh > 0)
                                        {//Value is connected to
island, add to existing island
                                              //do not increment
                                              map[i][j] =
ext_s_neigh;
                                        }
                                        else
                                        {//new island, s_neigh and
d_neigh have no values
                                              //but island does have
s_neighbors
                                              total_islands++;
                                              map[i][j] =
total_islands;
                                        }
                                  }
                                  else
                                  {//existing island has not been
determined from
                                        //straight neighbors, must
evaluate diag neighbors

                                        if(d_neigh == 0)
                                        {//no diagonal neighbors, but
has straight neighbors
```

```
                                        //with no island number
-> new island, update total
                                        //and save value

                                        total_islands++;
                                        map[i][j] =
total_islands;
                                    }
                                    //need to determine if any of
diagonal neighbors
                                    //are valid for
identification of connected island
                                    //          diag must be
above or under str neighbor

                                    test =
d_valid(diag_valid,map,i,j);

                                    if(!test)
                                    {//No valid diagonal
neighbors
                                        //new island, update
total and save value
                                        total_islands++;
                                        map[i][j] =
total_islands;
```

```
                                    }
                                    else
                                    {
                                          if(d_neigh > 0)
                                          {//diag neighbor,
connected to island exist
                                                //must make sure
that the diag connected to island
                                                //is a valid diag
for this island, if so will be added to island
                                                if(
(diag_valid[0] > 0) && (neighbors[1] > 0) )
                                                {//diag is valid
and connected to island
                                                      //added to
existing island
                                                      map[i][j] =
neighbors[1];

                                                }
                                                else if(
(diag_valid[1] > 0) && (neighbors[3] > 0) )
                                                {//diag is valid
and connected to island
                                                      //added to
existing island
```

```
                                                        map[i][j] =
neighbors[3];

                                                    }
                                                    else if(
(diag_valid[2] > 0) && (neighbors[5] > 0) )
                                                    {//diag is valid
and connected to island
                                                        //added to
existing island
                                                        map[i][j] =
neighbors[5];

                                                    }
                                                    else if(
(diag_valid[3] > 0) && (neighbors[7] > 0) )
                                                    {//diag is valid
and connected to island
                                                        //added to
existing island
                                                        map[i][j] =
neighbors[7];

                                                    }
                                                    else
                                                    {//valid diag
have not been connected to island
```

```
                                                        //therefore
new island, update total and save value

      total_islands++;

                                                        map[i][j] =
total_islands;

                                                    }

                                                }

                                                else

                                                {//no diag neighbord
connected to island exist

                                                    //therefore new
island, update total and save value

                                                    total_islands++;

                                                    map[i][j] =
total_islands;

                                                }

                                            }

                                        }

                                    }

                                }

                            }
```

```
            }
        }



        return(total_islands);
    }
```

REFERENCES

[1] M. R. Werner and W. r. Fahrner, "Review on Materials, Microsensors, Systems, and Devices for High Temperature and Harsh-Environment Applications," *IEEE Transactions on Industrial Electronics*, vol. 48, pp. 249-257, 2001

[2] J. McMurry and R. C. Fay, *Chemistry*, Second ed. Upper Saddle River: Prentice Hall, 1998.

[3] P. Atkins, *Physical Chemistry*, Sixth ed. New York: Freeman, 1997.

[4] R. Held, "GROWTH KINETICS OF GaN GROWN BY MOLECULAR BEAM EPITAXY USING Ga AND AMMONIA," University of Minnesota, 1999, pp. 169.

[5] D. L. Smith, *Thin-Film Deposition*. New York: McGraw Hill, 1995.

[6] D. E. Crawford, "The Effects of Growth Kinetics and Thermodynamics on Properties of GaN Growth by Molecular Beam Epitaxy," University of Minnesota, 1996, pp. 160.

[7] Crystal Growth: from fundamental to technology Muller, G, Metois, Jean, Rudolph, P.; 2004 ;Elsevier

[8] C. Baggio, R. Vardavas, and D. D. Vvedensky. Fokker-Planck equation for lattice deposition models. Physical Review E, 64(4):art. no. 045103 Part 2, 2001

[9] C. C. Battaile and D. J. Srolovitz. A kinetic Monte Carlo method for the atomic-scale simulation of chemical vapor deposition: application to diamond. Journal of Applied Physics, 82:6293–6300, 1997.

[10] K. A. Fichthorn and W. H. Weinberg. Theoretical foundations of dynamical Monte Carlo simulations. Journal of Chemical Physics, 95:1090–1096, 1991.

[11] D. T. Gillespie. The chemical Langevin equation. Journal of Chemical Physics, 13(1), 2000.

[12] G. H. Gilmer and P. Bennema. Simulation of crystal growth with surface difusion. Journal of Applied Physics, 43(4):1347–1360, 1972.

[13] M. F. Gyure, C. Ratsch, B. Merriman, R. E. Caflisch, S. Osher, and J. J. Zinck. Level-set methods for the simulation of epitaxial phenomena. Physical Review E, 58(6):R6927–R6930, 1998.

[14] B. A. Joyce, D. D. Vvedensky, G. R. Bell, J. G. Belk, M. Itoh, and T. S. Jones. Nucleation and growth mechanisms during MBE of III-V compounds. Materials Science and Engineering B, 67:7–16, 1999.

[15] W. W. Mullins. Theory of thermal grooving. Journal of Applied Physics, 28(3):333–339, 1957.

[16] K. Seshan, editor. Handbook of Thin-Film Deposition Processes and Techniques. Noyes Publications, Norwich, NY, 2002.

[17] D. D. Vvedensky, A. Zangwill, C. N. Luse, and M. R. Wilby. Stochastic equations of motion for epitaxial-growth. Physical Review E, 48:852–862, 1993.
[18] S. Wiggins. Introduction to Applied Nonlinear Dynamical Systems and Chaos. Springer-Verlag, New York, 1990.
[19] Paul Bratley and Bennett Fox, Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator, ACM Transactions on Mathematical Software, Volume 14, Number 1, pages 88-100, 1988.
[20] Bennett Fox, Algorithm 647: Implementation and Relative Efficiency of Quasirandom Sequence Generators, ACM Transactions on Mathematical Software, Volume 12, Number 4, pages 362-376, 1986.
[21] Gallivan, Martha A., “Modeling and Control of Epitaxial Thin Film Growth” Technical Report for the Division of Engineering and Applied Science, California Institute of Technology; Pasadena California, 2003
[22] Burkardt, John. School of Computational Science, Florida State University.
[23] Press, William et al; Numerical Recipes in C, The Art of Scientific Computing.; Cambride University Press, India 2002
[24] H.-C. Jeong and ED Williams, Surf. Sci. Rep. 34, 171 (1999). Y. Cui, X. Duan, Y. Huang and C. M. Lieber, "Nanowires as Building Blocks for Nanoscale Science and Technology" in Nanowires and Nanobelts " Materials, Properties and Devices, Z.L. Wang, ed. 3-68 (Kluwer Academic/Plenum Publishers, 2003).
[25] Y. Huang, X. Duan, Y. Cui, and C.M. Lieber "Gallium Nitride Nanowire Nanodevices," Nano Lett. 2, 101-104 (2002).
[26] C.M. Lieber "Nanowire Superlattices," Nano Lett. 2, 81-82 (2002).
[27] M.S. Gudiksen, L.J. Lauhon, J. Wang, D. Smith, and C.M. Lieber "Growth of Nanowire Superlattice Structures for Nanoscale Photonics and Electronics," Nature 415, 617-620 (2002).
[28] C.M. Lieber "The Incredible Shrinking Circuit," Sci. Am. 285, 50-56 (2001).
[29] X. Duan and C.M. Lieber, "General Synthesis of Compound Semiconductor Nanowires" Adv. Mat. 12, 298-302 (2000)
[30] X. Duan and C.M. Lieber, "Laser-Assisted Catalytic Growth of Single Crystal GaN Nanowires" J. Am. Chem. Soc. 122, 188-189 (2000).
[31] Y. Cui, Z. Zhong, D. Wang, W. U. Wang and C.M. Lieber, "High Performance Silicon Nanowire Field Effect Transistors," Nano Lett. 3, 149-152 (2003).
[32] G. Zheng, W. Lu, S. Jin and C.M. Lieber, "Synthesis and Fabrication of High-Performance n-Type Silicon Nanowire Transistors," Adv. Mater. 16, 1890-1893 (2004)
[33] C.J. Barrelet, A.B. Greytak and C.M. Lieber, "Nanowire Photonic Circuit Elements," Nano Lett. 4, 1981-1985 (2004).
[34] Balanis, C., Advanced Engineering Electromagnetics, John Wiley and Sons, New York 1989
[35] Liang, W.L., et al., Micromachining of circular ring micorostructures by femtosecond laser pulses., Optics and Laser Technology, 35 (2003), pp285-290.

[36] Bonod, Nicolas, *et al.*, Light transmission through a subwavelength micostructured aperture: electromagnetic theory and applications. Optics Communications, 245 (2005), pp. 355-361.
[37] Popov, Evgeny, et al., Enhanced transmission of light through a circularly structured aperture. Optical society of America, 2005, 240.6680.
[38] Sheppard, C.J.R, Fresnel diffraction by a circular aperture with off axis illumination and its use in deconvolution of microscope images. J. Opt. Soc. Am. A, Vol. 21, No. 4, April 2004
[39] Xing, Zhang-fan, *et al.*, Efficient method for the calculation of mean extinction. III. Approximation or representation of particle-size distributions by rational functions. J. Opt. Soc. Am. A, Vol. 11, No. 2, February 1994.
[40] Guo, Jiang, *et al.* General integral expressions for on-axis spherical waves diffracted at a circular aperture. Optics Communications 260 (2006), pp. 57-61.
[41] Romero, Julio *et al.*, Vectorial approach to Huygens's principle for plane waves: circular aperture and zone plates. J. Opt. Soc. Am. A, Vol. 23, No. 5, May 2006.
[42] Mielenz, Klaus, Algorithms for Fresnel Diffraction at Rectangular and Circular Apertures, Journal of Research of the National Institute of Standards and Technology. 103, 497, September-October 1998.
[43] Rapaort, D. C. , The Art of Molecular Dynamics and Simulation. Cambridge University Press, New York 2004.
[44] Pedrotti, F, Pedrotti, L. , Introduction to Optics. Prentice Hall, New Jersey, 1987.
[45] Gans, Werner, Fundamental Principles of Molecular Modeling. Plenum Press, New York, 1996.
[46] Leach, Andrew, Molecular Modeling Principles and Applications. Longman, England 1996
[47] Campbell, Stephen. The Science and Engineering of Microelectronics Fabrication. Oxford University Press, New York, 2001.
[48] Young, Hugh. Optics and Modern Physics, McGraw Hill, New York 1968
[49] Press, William, Numerical Recipes in C, The Art of Scientific Computing. Cambridge University Press, New York , 1996
[50] Hayt, William. Engineering Electromagnetics. McGraw Hill, New York, 1989.
[51] Vvedensky, Dimitri, Low Dimensional Semiconductor Structures: Fundamentals and Device Applications. Cambridge, New York. 2001.
[52] Hartell, A.D. *et al* The devlopment of RAS and RHEED as in situ probes to monitor dopant segregation in GS-MBE on Si (001). J. Crystal Growth 227-228 (2001) 729-734
[53] Liliya A. *et al* Comparision of numerical algorithms for simulation of molecular beam epitaxy. Vacuum 69 (2003)411-417.
[54] Einax M. *et al.* Simulation of MBE-growth of alloy nanoclusters in a magnetic field. Matr Sci and Engr. C 2006 Article in Press
[55] Pinto N. Strain-driven morphology of Si Ge islands grown on Si(100). Micron 31(2000) 315-321
[56] Zhou, J.J. *et al.* Virtual control simulator for closed-loop epitaxial growth. J. Crystal Growth 175-175 (1997) 52-60

[57] Zhao, M. *et al.* Strain-symmerterized Si/SiGe mulit-quantum well structures grown by molecular beam epitaxy for intersubband engineering. J.o. Luminescence 121 (2006) 403-408
[58] Grein, C.H. *et al.* Epitaxial growth simulation employing a combined molecular dynamics and Monte Carlo approach. Computational Material Science 6 (1996) 123-126
[59] Liu P. *et al.* Morphological evolution of heteroepitaxial islands during Stranski-Krastonov growth. Inte J of Solids and Structures (2006) Article in Press
[60] Yildiz M. *et al.* A continuum model for the Liquid Phase Diffusion growth of bulk SiGe single crystals. Int J of Engr Sci 43 (2005)1059-1080
[61] Yu, Q. *et al.* Molecular dynamics simulation of crystal growth in SiGe/Si (100) heterostructures. J.o Crystal Growth 149(1995)45-58
[62] Albenze, Erik *et al.* Molecular Dynamics Study of Explosive Crystallization of SiGe and Boron Doped SiGe Alloys. Ind Eng. Chem. Res 2006, 5628-5639
[63] Zhang, J. *et al.* In-situ monitoring of Si and SiGe growth on Si(001) surfaces during gas-source molecular beam epitaxy using reflectance anisotropy. J o. Crystal Growth 164(1996)40-46
[64] Kaxiras, E *et al.* Atomic structure of surfactant monolayres and its role in epitaxial growth. Materials Science and Engineering B30(1995)175-186
[65] Zhang, Y *et al.* Three dimensional analysis of shape transitions in strained-heteroepitaxial islands. Applied Physics Letter Vol 78(18) April 2001
[66] Drucker, J *et al.* Activated Strain Relief of Ge/Si(100) Islands Surface Review and Letters, Vol 7(5)2000 527-531
[67] Einfeldt, S. *et al.* Strain relaxation in AlGaN under tensile plane stress. Journal of Applied Physics Vol 88(12) Dec 2000
[68] Pinto, N. *et al.* Cluster size distribution of SiGe alloys grown by MBE. Thin Solid Films 336(1998) 53-57
[69] Gray, J. *et al.* Control of surface morphology thorugh variation of growth rates in SiGe/Si(100) epitaxial films: Nucleation of "quantum fortresses". Applied Physics Letters Vol 80,13 (2002) September
[70] Obayashi, Y. *et al.* Directional dependence of surface morphology stability of heteroepitaxial layer. Journal of Applied Physics Vol 84,6 (1998) September
[71] Spencer, B *et al.* Dislocation energetics in epitaxial strained islands. Applied Physics Letters Vol 77,16 (2000) October
[72] Floro, J.A. *et al.* Dynamic self-organized of strained islands during SiGe epitaxial growth. Applied Physics Letters Vol 73,7 (1998) August
[73] Chaparro, S. Evolution of Ge/Si(100) islands: Island size and temperature dependence. Journal of Applied Physics Vol 87,5 (2000) March
[74] Smith, D. *et al.* Micostructural evolutino of Ge/Si(100) nanoscale islands. J o. Crystal Growth 259 (2003) 232-244
[75] Spencer, B. *et al.* Equilibrium Shapes and Properties of Epitaxially Strained Islands. Physical Review Letters Vol 79,24 (1997) December

[76] Hearne, S. *et al.* Quantitative determination of tensil stress creation during island coalescence using selective are growth. Journal of Applied Physics 97,083530 (2005)
[77] LeGoues, F. *et al.* Relaxation mechanism of Ge islands/Si(001) at low temperatures. Appl Phys. Lett 67(16) October 1995
[78] Family, F. Scaling of the Droplet Size Distributino in Vapor Deposited Thin Films Physical Review Letters Vol 64(4) July 1988
[79] Barucca, B. *et al.* Strain relaxation thorugh islands formation in epitaxial SiGe thin films. App Surf. Sci 102(1996) 73-77
[80] Chaparro, S *et al.* Strain relief via trench formation in Ge/Si (100) islands. Applied Physics Letters Vol 76(24) June 2000
[81] Shchukin, V. *et al.* Stability of an hexagonal array of coherently strained conical islands against Ostwald ripening. Annals of Physcis 320(2005) 237-256
[82] Johnson, H. *et al.* Mechanics of coherent and dislocated island morphologies in strained epitaxial material systems. J. Appl. Phy. 81(9) May 1997
[83] Bird, D. *et al.* First Principles Calculation of the Structure and Energy of Si (113). Physical Review Letters Vol 69(26) December 1992
[84] Zhang, Y. *et al.* Morphological evolution driven by strain induced surface diffusion. Thin Solid Films 424 (2003) 9-14
[85] Seel, S *et al.* Tensile stress evolution during deposition of Volmer-Weber thin films. J Appl Phy. vol 88(12) December 2000
[86] J. E. Van Nostrand, S. J. Chey, and D. G. Cahill. Low temperature growth morphology of singular and vicinal Ge(001). Physical Review B, 57(19):12536–12543, 1998.

BIOGRAPHICAL STATEMENT

Mr. Clark spent three years in the U.S. Navy as a nuclear electrician. His duties focused on maintaining, monitoring and repairing the electrical distribution equipment for the nuclear power plant. Upon discharge from the Navy Mr. Clark spent two years working at the Nanofabrication center at UTA. While at the Nanofab center, his research focused on development and fabrication of wide band gap (WBG) semiconductors for use in high power, high temperature, and high frequency electronic device applications. His WBG semiconductor research focused on modeling of the polarization in GaN superlattice structures induced by the lattice stress due to mismatch using MATLAB and C++. Mr. Clark then worked at an R&D engineering firm where he acquired over $1,00,000 in research funding through several innovative research proposals that ranged from novel MEMS-based energy harvesting technique to flexible electronic devices. Mr. Clark than became coordinator for the Semiconductor Manufacturing Program at South East Tarrant County College. Mr. Clark than obtained an Assistant Professor position at DeVry University. Mr. Clark also started a small engineering firm that focuses on embedded development as well as statistical modeling applications. Mr. Clark obtained his Master of Science in Electrical Engineering in December of 2002 and his PhD in Electrical Engineering in August of 2006.