COMPUTATIONAL ANALYSIS OF THE EFFECT OF SURFACE ROUGHNESS ON THE DEFLECTION OF GOLD COATED SILICON MICRO-CANTILEVERS DUE TO MOLECULAR ADSORPTION

VICTOR HAYDEN

COMPUTATIONAL ANALYSIS OF THE EFFECT OF SURFACE ROUGHNESS ON THE DEFLECTION OF GOLD COATED SILICON MICRO-CANTILEVERS DUE TO MOLECULAR ADSORPTION

By
© Victor Hayden

A thesis submitted to the
Department of Physics and Physical Oceanography
in partial fulfillment of the requirements for the degree of
Masters of Science

Department of Physics and Physical Oceanography
Memorial University of Newfoundland

February 28, 2012

St. John's Newfoundland

Table of Contents

List of Tables

List of Figures

Abstract

In this work numerical simulations were performed in order to study the effects of surface roughness on the deflection of gold coated silicon cantilevers due to molecular adsorption. The cantilever was modeled using a ball and spring system where the spring constants for the Si-Si, Si-Au, and Au-Au bonds were obtained from first principal calculations. The molecular adsorption process was simulated by elongating the natural bond length at available bonding sites chosen randomly on the cantilever. Increasing the bond length created a surface stress on the cantilever causing it to deflect. In all cases the structure refinement was performed by minimizing the energy of the system using a simulated annealing algorithm and a high quality random number generator called Mersenne Twister. The system studied consisted of a 1 micrometer by 1 micrometer portion of a cantilever of various surface roughnesses with variable boundary condition and was processed in parallel on the ACEnet (Atlantic Computational Excellence Network) cluster. The results have indicated that cantilevers with a rougher gold surface deflected more than those with a smoother surface. The increase in deflection is attributed to an increase in stress raisers in the gold film localized around the surface features. The onset of stress raisers increases the differential stress between the top and bottom surfaces and results in an increase in the deflection of the cantilever.

Acknowledgments

I would like to thank my family for supporting me throughout my master's degree. Thank-you Mom, Dad, and Edward, without all of you I would never have been able to finish it. My friends whom have been around when I have needed them, I thank you. My office mates and friends whom have kept me company and provided a sounding board to discuss my theories, Thank- you, Kathryn Manning, Nicole Brown, Samantha Payne, and Nick Butt. I also would like to thank my high school technology teacher, Cliff Reid, without whom I may never have gotten so involved with programming. I would also like to thank those who got me interested in physics and math in the first place, my high school physics teacher, Doug Sheppard, and my math teachers Rouse Brake and Don Sheppard without their encouragement and patience I would have never had considered doing a Physics degree. I would also like to thank my supervisor Dr. Luc Beaulieu.

Chapter 1 Introduction

Cantilever based sensors are a promising technology for use as a wide variety of sensors including biosensors. However, before they can be used in any commercial application, it is important to understand and better control how they work. In this chapter a brief history of cantilever sensors, how a cantilever is modified to become a biosensor, and a look at previous work done in the area of adsorption of a self assembled monolayers on the thin gold film will be presented. It is not the intention of this work to model the behaviour of a biosensor, but instead to look at the surface stress effects related to the construction of a biosensor.

1.1 Micro-Cantilevers

Micro-cantilevers are small V-shaped or rectangular cantilevers (typically made of silicon nitride (Si_3N_4) or silicon (Si)) which are typically around 350 μm in length, 35 μm in width, and 1 μm in thickness. Micro-cantilever spring constants are typically in the range of 0.01-8.75 N/m depending on the dimensionality. Due to this small spring constant, it has been found that

micro-cantilevers are sensitive to changes in vibration, noise, humidity, surface stress and temperature [1][2][3].

In order to be used as a biosensor the cantilever must be modified so as to be able to detect target particles. However, modifying the micro-cantilever will obviously change how the cantilever responds to changes in humidity, vibrations, adsorption, etc. For example, how will the resonant frequency or deflection of the modified cantilever be changed upon exposure to a change in humidity, vibration, adsorption, etc? This must be taken into consideration before new sensors can be deployed commercially.

1.1.1 Development of Micro-Cantilever Based Biosensors

Originally micro-cantilevers were constructed to modify Scanning Tunnelling Microscopes (STM) to allow for the measurement of non-conductive samples. This system eventually evolved into the Atomic Force Microscope (AFM).

In 1994, researchers at Oak Ridge National Laboratories in conjunction with researchers at IBM realized that they could modify micro-cantilevers to become a new family of biosensors. They found that by measuring the change in the resonance frequency of the AFM cantilever that they could determine the change in mass of the cantilever with greater accuracy than conventional piezoelectric gravimetric sensors.

1.1.2 Construction of a Micro-Cantilever Biosensor

In order to create a biosensor the surface of a micro-cantilever needs to be functionalized. Surface functionalization is the process by which one surface of the cantilever is made to be physically, chemically, or biologically receptive to the specific physical phenomena, molecule, compound etc to be detected. Functionalizing the surface is often accomplished using a modified self assembled monolayer (SAM) that only interacts with the target molecule to be detected.

In order to allow for the growth of a SAM, the sensing surface of a cantilever must first be coated with a thin gold film. The deposition of gold on the silicon surface can be achieved in a variety of methods such as sputter deposition or chemical vapour deposition.

Coating the cantilever with a thin gold film allows for the use of alkanethiol based functionalized SAMs to be used. Alkanethiol based sensing layers are alkane chains, typically with 4 to 12 methylene units, with a thiol terminal group at one end and a receptive end group at the other. Thiol refers to a sulphur atom terminal group. It is known that sulphur has a strong attraction to gold and will readily bond with it. Therefore it should be possible to use this strong interaction in order to bond a SAM to the sensing surface of the micro-cantilever. This system can be schematically seen in Figure 1.1.

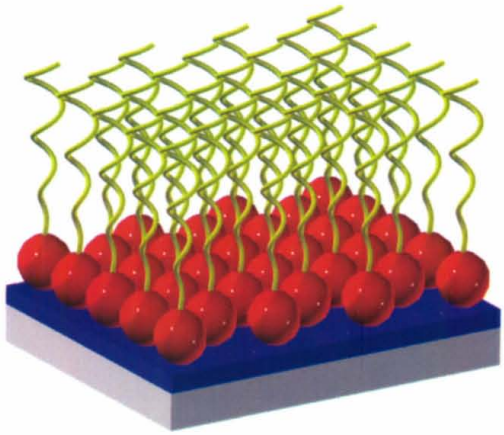


Figure 1.1: Micro-cantilever based biosensor, here the thiol head group is indicated by the red spheres, the alkane chain is represented by the yellow curls, the gold coating is in blue and the silicon substrate is in grey

Once this SAM has been grown on the surface the micro-cantilever has been functionalized and is ready to become a biosensor. The adsorption of this SAM on the surface will induce a surface stress on the micro-cantilever, as depicted schematically in Figure 1.2. This surface stress will cause a pre-bending of the cantilever before the target molecules have been introduced.

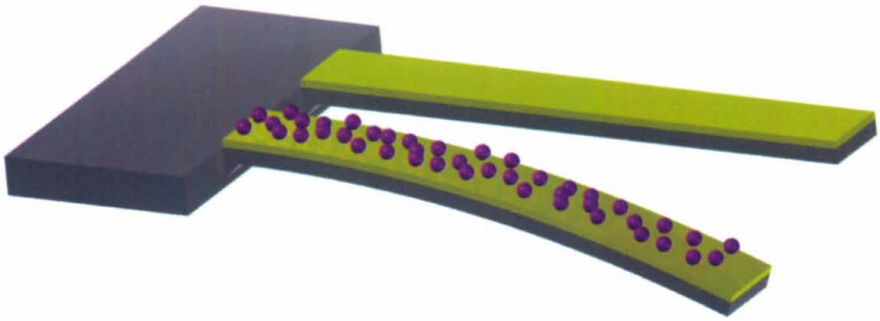


Figure 1.2: Functionalized Gold coated Silicon micro-cantilevers; (Foreground) adsorption of the SAM (depicted here as purple spheres) on the surface has resulted in the deflection of the micro-cantilever alkane chains removed for clarity. (Background) Un-deflected cantilever system.

The amount of pre-bending of these cantilevers is a subject of some controversy in the relevant literature. Some publications report that micro-cantilevers with rough gold surfaces will have a larger pre-bend than their smoother counterparts, while others report they will have a smaller pre-bend [4][5][6]. In order to be able to determine the true deflection of a micro-cantilever it is important to know the amount that the cantilever is bent before the experiment begins.

The deflection of the micro-cantilever is measured by the optical beam technique similar to that used in AFM. In the optical beam technique, a laser is focused towards the free end of the micro-cantilever. The beam is then reflected from the surface (either top or bottom

depending on the desired configuration) of the cantilever onto a position-sensitive detector (PSD)[7]. A small deflection of the cantilever will affect the path of the reflected beam and change the position of beam on the PSD. This enables the detector to determine the position of the cantilever. This PSD can measure displacements on its surface as small as 1 nm; which corresponds to miniscule deflections of the micro-cantilever [8].

1.1.3 Current Simulation Software

The majority of publications in this area usually use a commercially available modeling software called ANSYS® (discussed in detail in chapter 4) in order to model their cantilever systems. It was found that in preparing for this work that ANSYS® suffered from a few deflects. These were the difficultly in creating the surface mesh at the cantilevers scale and the determination of the bulk properties of the materials (directionally dependent). Based upon these discoveries it was decided to design a custom piece of simulation software.

There exist very few computational simulations in this area based upon established theory, most of the simulation data that exists is based upon ANSYS® or other similar commercial general purpose simulation software. Most depend upon the correlation of experimental data with raw theory. Weissmüller [9], and Baskaran [10] are a few of the authors that developed custom software for solving this problem, the exact computational methods used by these authors is not reported. Therefore the simulation presented in this work is fairly unique in this regard and it will be shown that it can solve atomistically for the deflection of a microcantilever.

1.2 Motivation

In a previous report by Godin et al. [4] it was found that the surface morphology had a strong influence on the adsorption of alkanethiols on gold coated micro-cantilevers [4]. Roughness of the gold surface was found to significantly affect the magnitude of the induced surface stress. It was found that for surfaces with a small grain structure experienced a smaller surface stress as compared to a large grain surface [4]. This surface stress is responsible for the deflection of the cantilever [3]. However other authors have found the complete opposite, Mertens et al. [6] for example found that cantilevers with rough surfaces will deflect more than those with smoother surfaces. Mertens found that the surface stress was actually increased as opposed to decrease as observed by Godin [6]. However, some researchers, Desikan et al. for instance, did not observe any significant increase in surface stress due to surface roughness [5]. Therefore there exists a large amount of contention on the subject of cantilever deflection with a rough surface.

The adsorption of the SAM on the surface will result in a pre-bend to exist on the functionalized micro-cantilever. This means that the cantilever is already deflected before target particles have been introduced into the system.

For a cantilever the surface stress can be calculated by the use of Stoney's equation using either the deflection or curvature of the cantilever. The surface stress calculated by Stoney's equation will be isotropic. The bend of the cantilever is not strictly circular, but can be

assumed to be so, under the assumption that the deflection is much less than the length of the cantilever [3].

Stoney's equation is dependent on the film thickness, and assumes a uniform film thickness over the length of the lever. Changing the surface roughness should affect the amount of stress on the surface; this in turn will change the deflection of the cantilever.

The motivation for this thesis was to answer the following questions. Assuming that the SAM has bonded to every possible binding site how is the deflection of the cantilever affected by the rough surface? How much pre-bending is to be expected? Would it be beneficial for the gold surface to be atomically smooth, or would a rough textured surface be more effective?

1.3 Scope of Thesis

In this thesis, a 2d "ball and spring" model of a one micron section of a bimetallic micro-cantilever (gold/silicon) with varying surface roughness is simulated. The numerical simulations seek to determine the optimum configuration of the lattice based upon a Monte Carlo "Simulated Annealing" technique.

Chapter 2 outlines the numerical model used in this simulation. This includes a description of lattice model used. Then the concepts of stress and strain are defined. Stoney's equation is then derived. This is the most accepted model for relating cantilever curvature to surface stress. Since Stoney's equation is the lowest order approximation for the curvature, a more modern theory based upon Timoshenko beam theory is also presented. The concepts of

Monte Carlo simulations and simulated annealing are then presented. These are the numerical methods used to determine the minimum energy of the lattice.

Chapter 3 outlines the computational aspects of the simulation, including the lattice creation program as well as the control interface. The atom movement routine is then presented, as well as the concept of the variable boundaries that are used in the simulation to mimic a much larger system.

Chapter 4 presents the results generated by the simulations, explains the results, and shows how surface roughness is connected to the magnitude of the deflection of the cantilever.

Chapter 5 contains a brief discussion, conclusions of this work and recommendations for future work on this subject.

Chapter 2 Numerical Model

In this chapter the numerical model will be presented. This will include a description of the model, as well as a derivation and modification of Stoney's equation to relate the change in the curvature of the top and bottom surface of the cantilever. Following this the model inter-atomic potential is outlined, as well as a description of the computational models used to determine the minimum lattice energy.

2.1 Ball and Spring Model

One of the models that can be used to study the properties of a crystal lattice is the so called "ball and spring" model. Here the atoms within the lattice are assumed to be balls or hard spheres that are connected to each other by means of ideal springs. The atoms are assumed to interact only through these springs, with all other atomic interactions ignored. The

spring constants have been determined by others in our group by means of *ab initio* calculations using Gaussian[1].

The model normally only connects an atom to its nearest and next-nearest neighbours, and assumes that any interaction beyond that point to be small and un-important [11]. A sample simple square lattice is shown in Figure 2.1. This model is commonly used to simulate surface stress and compressive loading [11][12].

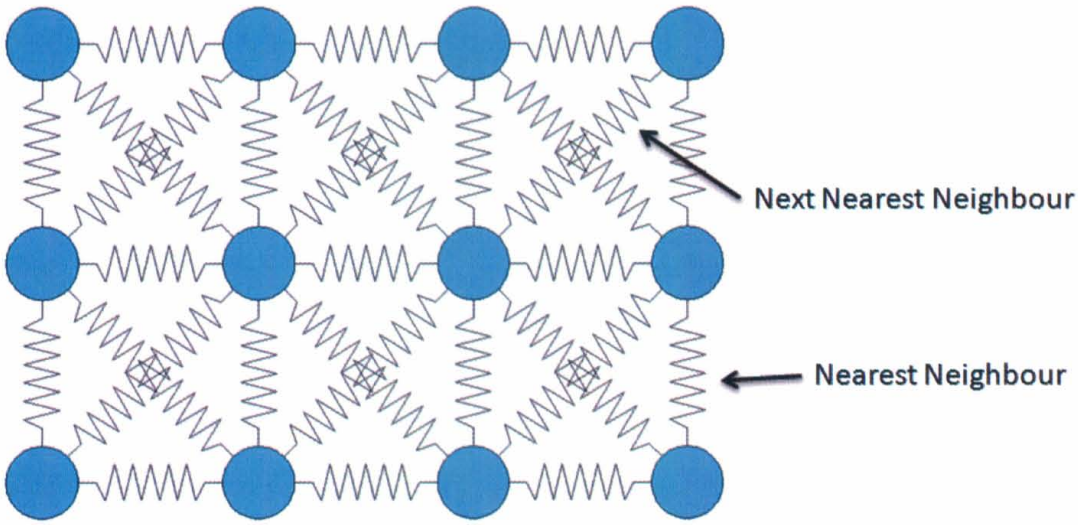


Figure 2.1: Spring Lattice Model. Atoms in blue are connected by means of ideal springs.

[1] Gaussian is software developed by Gaussian Inc. for electronic structure modeling.

2.2 Stress and Strain

The idea of surface stress was first proposed by Josiah W. Gibbs in 1906. He defined surface stress as "the amount of reversible work per unit area needed to elastically stretch a pre-existing surface" [13] . Stress then is a measure of the internal resistance of a material to the distortion caused by an external load.

The stress on an object is given by its stress tensor

$$\sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix} \quad 2.1$$

where $\sigma_{\alpha\beta}$is the stress associated with the α plane along the β direction.

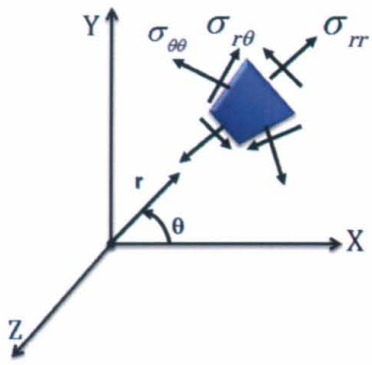


Figure 2.2: Cylindrical stress on a planar surface

In the case of the modeled cantilever system this stress tensor is best rewritten in terms of cylindrical coordinates (see Figure 2.2) allowing the stress tensor to be written as:

$$\sigma = \begin{pmatrix} \sigma_{rr} & \sigma_{r\theta} & \sigma_{rz} \\ \sigma_{\theta r} & \sigma_{\theta\theta} & \sigma_{\theta z} \\ \sigma_{zr} & \sigma_{z\theta} & \sigma_{zz} \end{pmatrix}. \quad 2.2$$

The modeled system is planar, therefore any stress that is out of plane is zero leading to a two dimensional stress tensor

$$\sigma = \begin{pmatrix} \sigma_{rr} & \sigma_{r\theta} \\ \sigma_{\theta r} & \sigma_{\theta\theta} \end{pmatrix}. \tag{2.3}$$

Referring to Figure 2.2 the values $\sigma_{rr}$and $\sigma_{\theta r}$ must be zero, since the only stress in the system is applied along the θ direction, therefore the only non-zero element of the stress tensor is $\sigma_{\theta\theta}$.

Strain is a normalized measure of distortion representing the displacement between configurations of a body relative to its original length. Strain is given by (in one dimension, or for isotropic materials)

$$\varepsilon = \frac{\Delta L}{L_0}, \tag{2.4}$$

where ΔL is the change in the length of the object and L_0 is the original length of the object. Here the change in length for a deflected cantilever system will be the change in length of the cantilever for a fixed radial value.

Plotting stress versus strain yields many properties of a material. The curve varies from material to material. Figure 2.3 is a sample plot of a typical low carbon steel stress strain curve. Most solid materials will have a Stress-Strain curve with similar features.

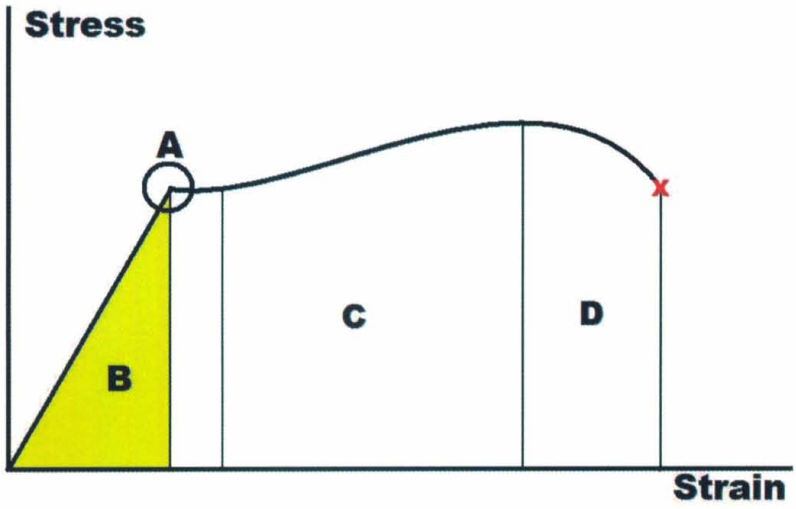


Figure 2.3: A is the Yield Strength above which the material is no longer Hookean, B is the Elastic region, C is the Strain Hardening Region, D is the Necking Region, the spot designated by the X is the rupture or fracture point, the transition point between C and D is the Ultimate Strength.

If a material is still in region B of Figure 2.3 then its behaviour is considered to be Hookean. Hookean materials when deformed will return to their original state once the application of force has been removed, i.e. the total strain on the object is below the yield strength. Therefore any point below point A the material is still in the elastic region. Above the yield strength the Stress Strain curve becomes non-linear, see Figure 2.3, causing any deformation to be permanent. Removal of the applied force beyond point A will not result in the object reverting back to its original shape.

For any Hookean material the Stress (σ) is proportional to the Strain (ε) through

$$\sigma = E\varepsilon, \tag{2.5}$$

where E is the modulus of elasticity of the material.

Upon substitution of equation 2.4 into equation 2.5, stress can be then written as

$$\sigma = E\frac{\Delta L}{L_0}. \tag{2.6}$$

Again here the change in length for a deflected cantilever system will be the change in length of the cantilever for a fixed radial value.

2.3 Stoney's Equation and Cantilever Curvature

The classical model for an isotropic thin film coated cantilever was published originally in 1909 by G. Gerald Stoney. In that paper Stoney argued that the tension per unit area for a thin film could be determined using equation 2.7. Given σ is the tension per unit area and t is the thickness of the thin film then,

$$\sigma t = \int_0^{\boldsymbol{d}} \frac{\boldsymbol{E}}{r}(b-x)dx = \frac{\boldsymbol{E}}{r}\left(bd - \frac{d^2}{2}\right), \tag{2.7}$$

where r is the radius of curvature, $\boldsymbol{E}$ is the biaxial modulus of the substrate, d is the thickness of the cantilever, and b is the distance from the film/substrate boundary to the neutral axis of the cantilever [14]. (See Figure 2.4)

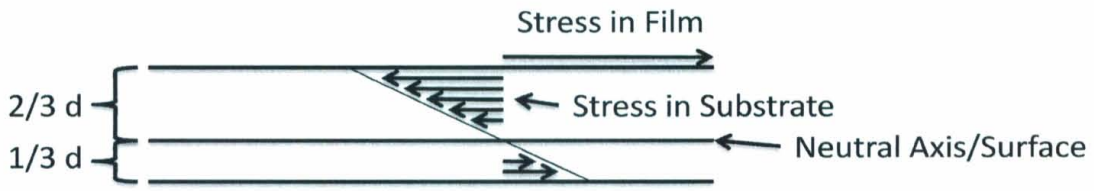


Figure 2.4: Location of neutral axis and stress distribution according to Stoney's equation.

By using the condition that the sum of the moments of forces about the X-axis is equal to zero allows the depth to the neutral axis to be calculated from

$$\int_0^d \frac{\boldsymbol{E}}{r}(b-x)x dx = 0. \tag{2.8}$$

Solving equation 2.8 yields,

$$\frac{\boldsymbol{E}}{r}\left(\frac{bd^2}{2}-\frac{d^3}{3}\right) = 0 \tag{2.9}$$

or

$$b = \frac{2}{3}d.$$

Substituting the last result into equation 2.7 gives [14]

$$\sigma t = \frac{1}{6}\boldsymbol{E}\frac{d^2}{r}. \tag{2.10}$$

The substrate biaxial modulus $\boldsymbol{E}$ can be rewritten as

$$\boldsymbol{E} = \frac{E_s}{(1-\nu_s)}, \tag{2.11}$$

where E_s is the Young's modulus, and ν_s is the Poisson's ratio for the substrate [15].

Substituting equation 2.11 into equation 2.10 and rearranging results in

$$\Delta r = \frac{1}{6}\frac{E_s}{(1-\nu_s)}\frac{d^2}{\Delta\sigma t}. \tag{2.12}$$

This equation relates the change in radius between the top and bottom surfaces to the differential mechanical stress.

Equation 2.12 can be rearranged into the following,

$$\Delta\frac{1}{r} = 6\frac{(1-\nu_s)}{E_s}\frac{\Delta\sigma t}{d^2}. \tag{2.13}$$

Alternatively equation 2.13 can be re-written using the change in curvature, ΔK,

$$\Delta K = 6\frac{t(1-\nu_s)}{E_s d^2}\Delta\sigma. \tag{2.14}$$

The curvature of a cantilever can be related to the deflection of the free end by

$$K = \frac{2\Delta z}{l^2}, \tag{2.15}$$

substituting into the previous result yields the deflection of the free end of the cantilever as

$$\Delta z = \frac{3l^2(1-\nu_s)}{E_s d^2}\Delta\sigma. \tag{2.16}$$

2.3.1 Timoshenko Beam Theory

Stoney's equation serves as a simplistic method for looking at a thin film coated cantilever. However, not all of the material properties of both the micro-cantilever and the thin film are taken into account in the derivation of Stoney's equation. For instance the stress/strain of the gold/silicon interface is not included in Stoney's derivation. Therefore to gain a better understanding of the deflection of the cantilever it becomes necessary to look at the beam theory proposed by Timoshenko in 1925 [16].

It has been shown by Yoshikawa that if a cantilever has been coated by a film that has an induced isotropic internal strain (ε_f) then its deflection verses stress/strain is different as compared to the accepted Stoney's equation. [17] However it will, under the assumption that $d \gg t$, reduces to Stoney's equation.

Stoney's equation for the deflection of the free end of the cantilever (Δz) is,

$$\Delta z = \frac{3l^2(1-\nu_c)}{E_c d^2}\sigma_{surf}. \qquad 2.17$$

Where l is the cantilever length, ν_c is the Poisson ratio of the cantilever, E_c is the Young's modulus of the cantilever, and σ_{surf} is the surface stress.

Yoshikawa used the conditions for equilibrium (forces and moments), and the balance of strains on the cantilever/film interface to demonstrate that the deflection of a bi-layered cantilever is,

$$\Delta z = \frac{3l^2(t+d)}{(A+4)t^2 + (A^{-1}+4)d^2 + 6td}\varepsilon_f \qquad 2.18$$

with

$$A = \frac{E_f w_f t(1-\nu_c)}{E_c w_c d(1-\nu_f)},$$

here ε_f is the internal strain in the film, E_f is the Young's modulus of the film, w_f width of the film, w_c is the width of the cantilever, and ν_f is the Poisson ratio of the film [17].

Here ε_f can be replaced by,

$$\varepsilon_f = \sigma_f \frac{(1-\nu_f)}{E_f} \quad \text{or} \quad \varepsilon_f = \frac{\sigma_{surf}}{t}\frac{(1-\nu_f)}{E_f} \qquad 2.19$$

where σ_f is the internal stress ($\sigma_f = \frac{\sigma_{surf}}{t}$). Typically Stoney's equation is written in terms of the surface stress, therefore equation 2.18 becomes,

$$\Delta z = \frac{3l^2(t+d)}{(A+4)t^2 + (A^{-1}+4)d^2 + 6td} \frac{\sigma_{surf}}{t} \frac{(1-\nu_f)}{E_f}. \qquad 2.20$$

Equation 2.20, relates the deflection of the free end of the cantilever with the applied surface stress, without assuming that $d \gg t$. Applying the assumption, that $d \gg t$ and $w_c = w_f$ to equation 2.20, yields upon simplification, Stoney's Equation (equation 2.17).

2.3.2 Applying Stoney's Equation to the "Ball and Spring" Model

Equation 2.14 can be re-written in terms of the film modulus of elasticity and the change in length by substitution of Equation 2.6 for the stress. Therefore Stoney's equation 2.14 becomes

$$(\mathrm{K_T} - \mathrm{K_B}) = 6\frac{tE_F(1-\nu_s)}{E_s d^2 L_0} \cdot (\Delta L_T - \Delta L_B), \qquad 2.21$$

where $\mathrm{K_T}$ and $\mathrm{K_B}$ are the top surface and bottom surface curvatures, respectively. Similarly, ΔL_T and ΔL_B are the changes in the length of the top and bottom surfaces as compared to the original length. The change in curvature can be calculated by fitting a circle to the points that define the top and bottom surfaces, and the changes in length can also be similarly calculated.

2.4 Modeling Interatomic Interaction

It is known that the energy between two atoms can be modelled by the Lennard-Jones 12-6 potential given by

$$V(R) = \varepsilon \left(\left(\frac{L_{Spring}}{R} \right)^{12} - 2 \left(\frac{L_{Spring}}{R} \right)^{6} \right) \qquad 2.22$$

or

$$V(R) = 4\varepsilon \left(\left(\frac{\sigma_{LJ}}{R} \right)^{12} - \left(\frac{\sigma_{LJ}}{R} \right)^{6} \right),$$

where R is the interatomic radial separation between the atoms, L_{Spring} is the natural spring length of the spring, ε is the depth of the potential well and σ_{LJ} is the characteristic Lennard-Jones length (typically the diameter of the smallest particle)[18]. A sample plot of the Lennard-Jones potential is shown as plot C in Figure 2.5.

While this potential describes the dual attractive and repulsive interactions between a pair of neutral atoms or molecules, it also covers both of the long range and short range forces. The nature of this simulation is not interested in long range interactions since the atoms within the lattice are not expected to increase in bond length by a large amount. So it is possible to use the most basic potential, this would be a simple harmonic.

2.4.1 Simple Harmonic Potential

One of the simplest models of a crystal lattice can be constructed from the assumption that the lattice system can be modeled as if all the atoms contained within the lattice are connected through a set of imaginary springs. This model works sufficiently well enough that it can be used to model simple crystal lattices without having to involve complex inter-atomic potentials. This model works well in systems where the expected amount of bond stretching is low, i.e. the equilibrium bond distance is very close to that of the stretched bonds. The atoms located within a solid crystal are not expected to change their position to a great extent.

The potential between two atoms can therefore be modeled as a simple harmonic, and is given the form

$$Energy = \frac{1}{2} k_{IAS} \left(Radius - L_{Spring} \right)^2, \tag{2.23}$$

where the interatomic spring restitution coefficient between the two atoms is k_{IAS}, $Radius$ is the interatomic separation of the atoms, and L_{Spring} is the un-stretched natural bond length. This potential can be as line "A" in Figure 2.5, and is favourable when it comes to computation as it is faster to calculate as compared to the Lennard-Jones equation. This is a benefit as the simulation will be running on a very large system, and should cut down on the total amount of execution time.

Comparing the two potentials, both simple harmonic and the Lennard-Jones model (Line C, Figure 2.5) it can be seen that the minimum for both potentials occurs at the same point and

that the behaviour about this point is similar. However the simple harmonic potential has a problem. While it models the regime around the critical radius well it does not model the cases where the radius becomes small, the potential is finite for zero radius, and becomes infinite as the radius tends to infinity. Therefore this potential needs to be modified to mimic the defining characteristics of the Lennard-Jones potential. This can be done with the addition of a simple scaling term,

$$\left(\frac{L_{Spring}}{Radius}\right)^2. \qquad 2.24$$

Combining equation 2.23 and 2.24 yields the new potential of the system to be defined as

$$Energy = \frac{1}{2} k_{IAS} \left(\frac{L_{Spring}}{Radius}\right)^2 (Radius - L_{Spring})^2. \qquad 2.25$$

This potential is shown as Line B in Figure 2.5 which shares similarities to the behaviour of the Lennard-Jones model. As $Radius$ tends to zero this potential tends to infinity, just like the Lennard-Jones, and similarly as $Radius$ goes to infinity the energy plateaus at a constant value.

Computationally the potential given by equation 2.25 is faster to calculate then that as given by equation 2.22, and applying the restriction on the simulation that the atoms will not move by large amounts then the modified simple harmonic oscillator model is acceptable. If indeed the atoms in the simulation move to a great extent relative to each other, then the

modifications made to the simple harmonic potential will help it behave like the Lennard-Jones potential.

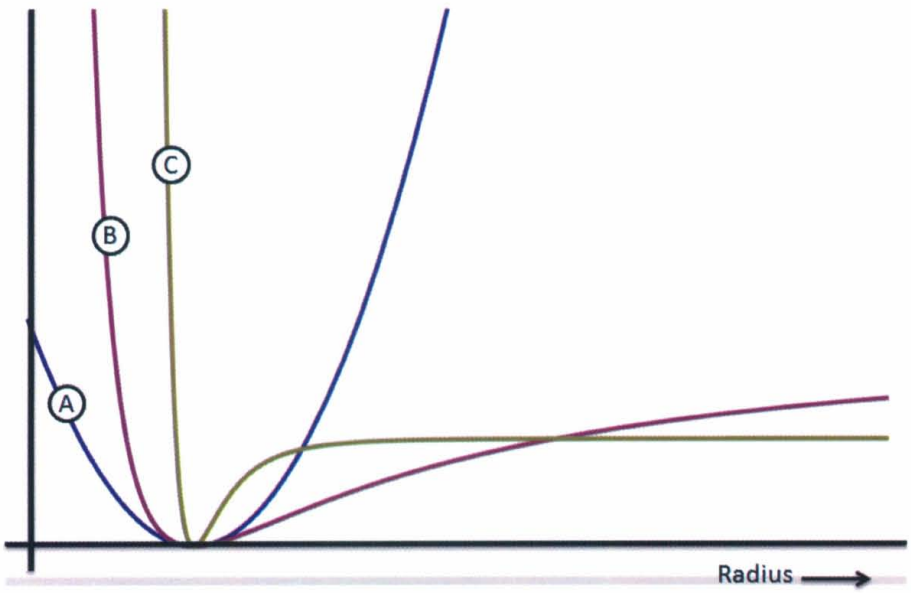


Figure 2.5: (A) Spring Energy of individual lattice spring, plot of equation 2.23. (B) Modified Spring potential that is used in the simulation, plot of equation 2.25 . (C) Lennard-Jones Potential (Equation 2.22). The modified spring potential mimics the behaviour of the Lennard-Jones potential whilst reducing the total number of calculations required since squared terms are faster to calculate then terms to the 6^{th} or 12^{th} power.

2.5 Lattice Spring Constants

The spring constants of the system were determined by others in our group using Gaussian and can be found in Table 1. Gaussian reports the spring constants for each bond in terms of the atomic units system; these can be found in column two of Table 1. The equivalent converted SI units can be found in column three of Table 1.

Bond	Coefficient of Restitution k $(H/\text{Å}^2)$[2]	Coefficient of Restitution k(N/m)
Silicon - Silicon	0.28584	124.619
Gold - Silicon	0.14600	63.652
Gold - Gold	0.30608	133.443
Surface (Explained Below)	1×10^9	4.35975×10^{11}

Table 1: Coefficients of Restitution[3]

The spring constants of the springs that are determined to be along the surface are assigned values that are approximately ten orders of magnitude larger than similar springs located within the bulk. When a molecule is adsorbed onto the surface it will force the top gold atoms to separate, this interaction is assumed to force and hold the atoms apart by a set amount. Hence by setting the coefficient of restitution to be very large these springs then

[2] While not in SI units, these units are most commonly published for these values.

[3] Determined by previous members of Dr. Luc Y. Beaulieu's Research Lab Group and were provided at the onset of this project.

behave more like rigid rods, then actual springs, when compared to the rest of the springs in the system.

In order to be classified as a surface bond a spring must be located on the top surface of the gold film, it also may not have any spring that crosses it, and must start in the horizontal position. Figure 2.6, shows an example of the bonds used in the simulation that will be used as surface springs. These springs are considered to be the binding sites, these are the locations where the thiol is expected to bond to the surface. When a thiol is adsorbed on the surface it will induce a surface stress due to the interaction of gold and thiol. The thiol will force the gold atoms surrounding the binding site to separate. Therefore only these springs will experience the induced stretch, the rest of the springs that are on the surface but are not considered to be a part of the surface springs or binding sites will be treated as if they existed purely in the bulk.

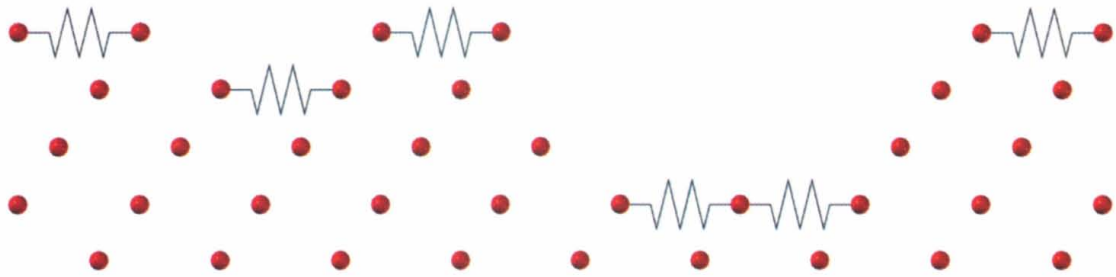


Figure 2.6: New and Remaining Surface bonds after removal of surface atoms

The spring constants in Table 1 are for the bonds that exist between the nearest neighbour atoms within the lattice. However since the simulation considers both the nearest neighbour and the next nearest neighbour bonds these coefficients need to be corrected since

the springs between the nearest and next nearest neighbours will have different restitution coefficients.

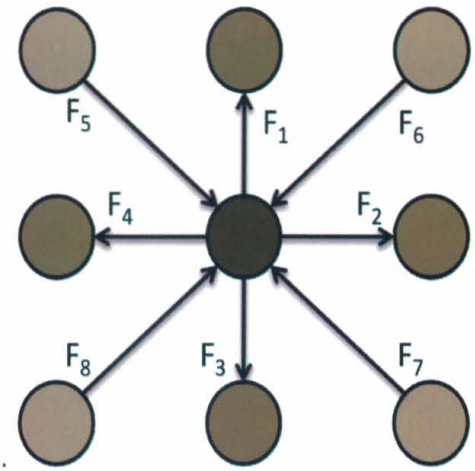


Figure 2.7: Forces between an atom and the nearest and next nearest neighbours

In order to calculate these new spring constants the forces between the atoms are needed to be considered (Figure 2.7). The lattice constant a, for a square lattice is determined by finding the value that minimises the total lattice energy for the entire rigid structure.

The nearest neighbour energy, given by $-J_1 = V_1(a)$, is purely attractive and thusly allows for atomic cohesion. The next-nearest neighbour energy, given by $-J_2 = V_2(\sqrt{2}a)$, can either be moderately repulsive or moderately attractive depending upon the exact nature of the neighbouring atoms.

The total energy of an ideal rigid lattice can therefore be written as,

$$U(a) = 2\,N\left(V_1(a) + V_2\left(\sqrt{2}a\right)\right). \tag{2.26}$$

In order to determine the lattice parameter that results in the optimum lattice configuration it becomes necessary to determine the minimum of equation 2.26. The condition for the minimum energy

$$\frac{dU(a)}{da} = 0, \tag{2.27}$$

implies that

$$-V'_1(a) = \sqrt{2}V'_2\left(\sqrt{2}a\right). \tag{2.28}$$

Figure 2.7 shows the forces that are acting on an atom. To maintain the atom in equilibrium, no external forces have been applied, the force between the atoms is such that

$$\|F_1\| = \|F_2\| = \|F_3\| = \|F_4\| = \sigma_0, \tag{2.29}$$

and

$$\|F_5\| = \|F_6\| = \|F_7\| = \|F_8\| = \frac{1}{\sqrt{2}}\sigma_0. \text{ [11]}$$

This change in the force is due to the elongation of the natural spring length of the springs between the next-nearest neighbour atoms. This change is reflected in the reduction of the next nearest neighbour spring constants.

2.6 Monte Carlo Family of Simulations

The Monte Carlo family of algorithms utilize the repeated random sampling of a variable in order to determine the desired results. Historically the name Monte Carlo is derived from work done during the 1930s and 1940s on the Manhattan Project, many computer simulations were performed to estimate the probability that the chain reaction needed for the atom bomb to function would work successfully [19].

2.6.1 Metropolis–Hastings Algorithm and Simulated Annealing

The Metropolis-Hastings algorithm has its origins in 1953 through the work of Nicholas Metropolis et al., who were attempting to develop a model to calculate the properties of a collection of interacting particles that obey classical statistics [20]. This method was updated in 1970 by Keith Hastings, and has since become known as the Metropolis-Hastings Algorithm [21]. According to classical statistics the probability of a system being in a state E_i is given by

$$P(E_i) = \frac{1}{Z} e^{-\left(\frac{E_i}{k_B T_A}\right)}, \tag{2.30}$$

where Z is the partition function, k_B is Boltzmann's constant, T_A is temperature and E_i is the energy of state i.

Referring to the systems to be solved in this work, the number of possible microstates makes the computation of the partition function in equation 2.30 to be practically impossible. The Monte Carlo algorithm is able however to get around this, by sampling this microstate

space randomly. This random selection can lead to problems, where the variable sampling may miss important portions of the domain or are located very far away from the minimum energy [22]. In order to avoid these problems the concept of importance sampling was integrated into the Monte Carlo scheme to form a new technique[22]. The concept of importance sampling states that instead of using a uniform distribution as proposed under the standard Monte Carlo scheme, a biased distribution φ is used where φ is the collection of microstates that contribute the most to the determination of the minimum energy [22].

Consider here two possible configurations, A and B. The probability that the transition from state A to B will occur under the condition for detailed balance using the same definition for temperature as detailed above is

$$P = \frac{P(E_A)}{P(E_B)} = \frac{e^{-(\frac{E_A}{T_E})}}{e^{-(\frac{E_B}{T_E})}} = e^{(\frac{E_B - E_A}{T_E})}. \qquad 2.31$$

The partition functions for both of these probabilities are the same and thus can be eliminated, leaving equation 2.31. However, a system is not energetically isolated from its surroundings, and may exchange energy with them. Here T_A from equation 2.30 has been transformed from representing temperature into a measure of how willingly the environment shares its energy ($T_E = k_B T_A$). "Temperature" here now has units of energy. The greater the "temperature", the more willing the environment is to give energy to the system; the smaller the "temperature", the more influence the environment puts on having the system in a low-energy state.

In 1983, Kirkpatrick et al. began to examine methods to computationally solve the "travelling salesman problem" [23]. This problem is classified as being an NP-hard problem in combinatorial optimization. NP-hard problems are a defined complexity class of decision problems in theoretical computer science that are intrinsically harder than those that can be solved by a nondeterministic Turing machine in polynomial time. Solving this type of problem would only be possible, in a finite time, on a computer that would allow for infinite parallelism.

The travelling salesman problem states that given a list of cities and their locations on a map find the shortest possible route that visits each city exactly once. Kirkpatrick et al. proposed to use a method of simulated annealing in order to solve problems of this nature [23].

Simulated annealing is not annealing but a computational analog to the process of annealing that can be applied to large combinatorial optimization problems. It is then called simulated annealing due to this similarity, and due to historical reasons the name has remained. However, this technique falls under the much broader family of Monte Carlo simulation algorithms.

Annealing is defined as the process in which a solid sample is heated to the point where the particles are allowed to freely rearrange themselves into a random distribution. The system is then allowed to cool at a very slow rate. To allow the particles to rearrange themselves into the lowest energy state crystal lattice. The key steps here are that the initial temperature must be sufficiently high to allow for this free rearrangement and that the sample is cooled at a slow enough rate to not inhibit the formation of the lowest energy lattice. Cooling at a rate that is

too fast can result in the sample forming a higher energy lattice or leading to a non-optimum configuration.

Similarly large combinatorial optimization problems can be thought of in a similar way since the algorithm is searching for the lowest energy configuration or the best configuration within a very large complex system. Simulated annealing seeks to find the deepest local minima to the starting position. [4]

The algorithm is an application of a Markov Chain to sample configuration space to determine the optimum configuration using transition probabilities [24]. Here the configuration space is defined to be the collection of all possible configurations for the lattice. A.A. Markov proposed the methodology of looking at probabilities that occur in complex changing systems to determine the most likely state that a system will have after a time that began in a particular state [24][25].

A simple four state Markov Chain is presented below in Figure 2.8, if it is assumed that the system begins in state A, then after a period of time it has a 25% chance of now being found in state C, similarly for state B, and a 50% chance of remaining in state A.

[4] There is no algorithm for determining the true global minimum; however simulated annealing searches to find the best local minimum that is closest to the starting point.

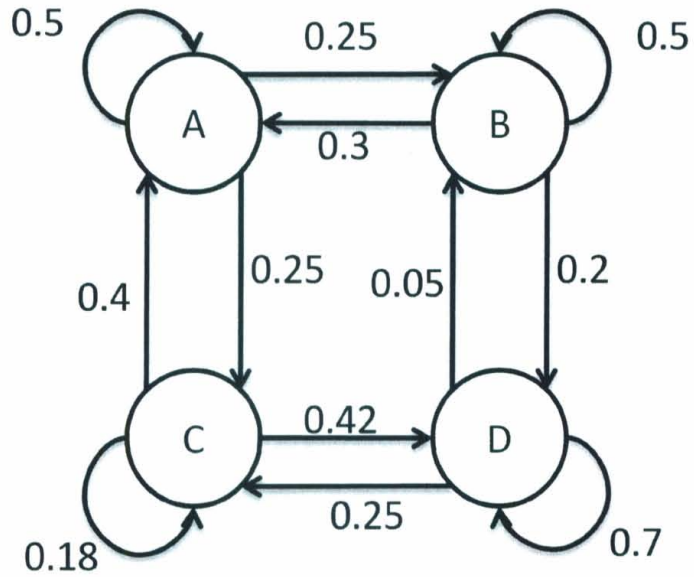


Figure 2.8: Four State Markov Chain with transition probabilities

Now consider the problem of optimization of the positions of the atoms within a crystal lattice. Given the configuration space set $C = \{C_1, C_2, C_3, C_4, \dots, C_{Ini}, \dots, C_\infty\}$, where C_{Ini} is the initial configuration. The Markov process would seek to minimize the lattice energy by looking at the transition probabilities associated with each of the configurations and would select the highest transition probability and that would result in the most optimized configuration. However, it is not possible to sample every configuration in this set, as it is unrealistic to analyse every possible lattice at once.

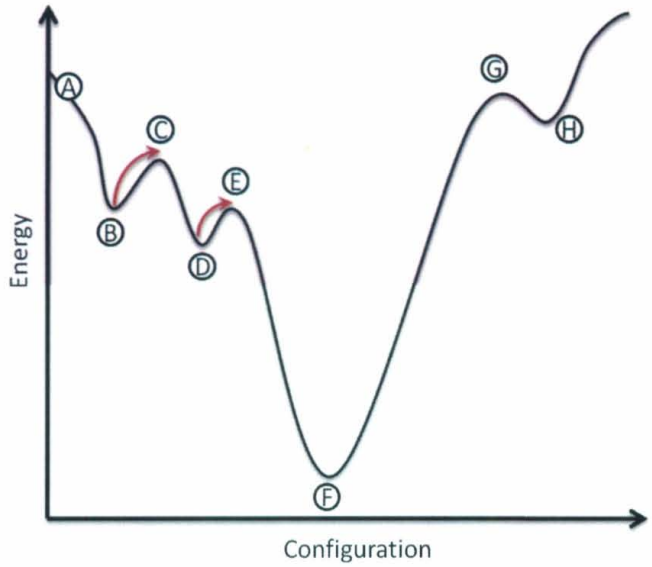


Figure 2.9: Energy VS Configuration for an arbitrary sample lattice

Referring to Figure 2.9, thinking about the Markov chain that contains all possible configurations of a crystal lattice, some configurations will have a lower energy than others. Let the sample begin in state A. Now consider state B; its energy is lower than A so it is accepted right away. The next state is C, whose energy is higher than B is clearly not a minimum and typical minimization techniques will report that state B is the minimum energy configuration for the system. However is B the best minimum that is possible?

The method of simulated annealing allows for the temporary and random jumping back into less optimized states. Here the algorithm will hold onto position B and then temporarily allow the transition to state C. The determination of whether or not to move to State C is made

by the Markov chain probability between the two states. It is a random process whether or not the transition will occur.

From Figure 2.9 is can be seen that state C is at a higher energy state then B, but the transition has been allowed temporarily to see if a better configuration can be found. This random exploration of solution space around the current best configuration is the real power of simulated annealing as it will allow for the transition from C to D, and thus replaces B as the new best configuration until it reaches the configuration at point F. The algorithm then will only accept configurations of the lattice that are around the current one, and will ignore the rest, and hence will ignore the bulk of configuration space.

Referring to Figure 2.9, why was State B chosen over State D? Under the random perturbation of State A, State B was the first minimum value that was found and so was assigned to be the best minimum. Why then was the decision made to randomly jump to state C?

In this simulation the form of equation 2.31 becomes

$$P = e^{\frac{(E_{Current} - E_{Temp})}{T}}. \qquad 2.32$$

Equation 2.32 is the acceptance probability. This can also be seen in Figure 2.10, where the current best is $E_{Current}$ and E_{Temp} is the energy of the new configuration. Here $E_{Current}$ can either be the value of the minimum or can also be the value of a temporary jump to a

higher energy state. If $E_{Temp} < E_{Current}$ then the configuration is accepted as being better since the acceptance probability becomes large.

If $E_{Current} < E_{Temp}$ and the value calculated by equation 2.32 is larger than a high quality random number[5] chosen in the range of zero to one, then the system moves into a temporary higher energy state, this is how the transition from State B to C in Figure 2.9 is allowed to occur.

The Simulated Annealing algorithm is shown below in Figure 2.10. The value of T within the algorithm controls the amount of solution space that the algorithm is allowed to explore. This allows for the initial free exploration of the surrounding states from an initial state. Once the "Temperature" begins to cool the amount of exploration that can be done decreases and a solution to the problem is determined.[6]

[5] A high quality random number comes from a long periodicity pseudorandom number generator with an even distribution of generated points, such as Mersenne Twister. Refer to Appendix A and B.

[6] Refer to Appendix E, for the Simulated Annealing source code

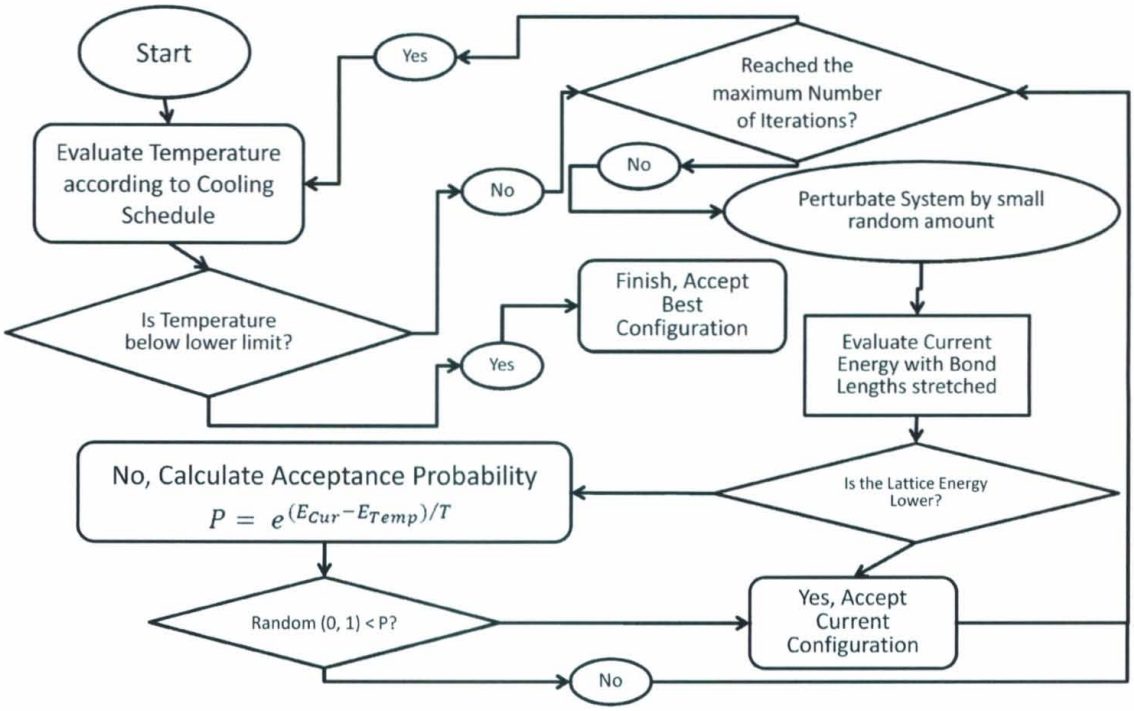


Figure 2.10: Simulated Annealing Algorithm Flow Chart

2.6.2 "Cooling" Schedule

In the simulated annealing algorithm the value of the "cooling" variable is controlled by way of a cooling schedule. The cooling temperature variable is analogous to the temperature decrease that a system would experience if it were in an actual annealing process. There are established schedules that are commonly used, and some problems require the creation of new schedules.

This simulation utilizes the simple cooling schedule, which obeys the equation

$$T_{Next} = \tau\, T_{Current}, \tag{2.33}$$

here T_{Next} and $T_{Current}$ are the next iteration "temperature" and the current iteration "temperature". The value of τ controls the decrease in "temperature", which in this work was set to 0.75. Under a simple cooling schedule this value can be in the range of $0.5 \leq \tau \leq 0.99$, and is dependent on the problem [26].

The initial temperature for the system is selected to have a value of three million degrees (where temperature is in units of energy as detailed in section 2.6.1). This high temperature allows for the free movement of the system into multiple high energy states allowing the system to select the best starting point to cool the system from.

Chapter 3 Computational Methods

In this chapter the methods used for the lattice creation, movement of atoms and variable boundaries will be presented. These methods are incorporated into the program used in this work to solve for the minimum energy for different systems of large numbers of variables (~15 million). The systems studied in this work are approximately 1 micrometre long by 1 micrometre high of silicon with varying amounts of gold. Firstly the initial lattice creation program is described.

3.1 Lattice Creation

In order to be able to produce the input lattices required for the simulations an additional program was designed to create the array of atoms including all the necessary atom information required to complete the simulation. This program can be found in Appendix G.

Silicon has a diamond cubic lattice configuration while gold is a face-centered cubic. The silicon cantilever's top surface will be (001) therefore the selection of the cross section can be made along the (100) plane, this will give a cross section for the silicon that can be seen in Figure 3.1. The selection for the orientation of the thin gold film was made by rotating the unit cell until the orientation formed flat layers. This orientation of the unit cell was determined to be along the (111) plane. Therefore the model assumes that the flat layers of gold lie parallel to the (111) plane.

Figure 3.1 is a sample representation of the lattice configuration that will be used in each of the simulations. Gold is aligned on the surface of the silicon as to force the boundaries to match on both the left and right.

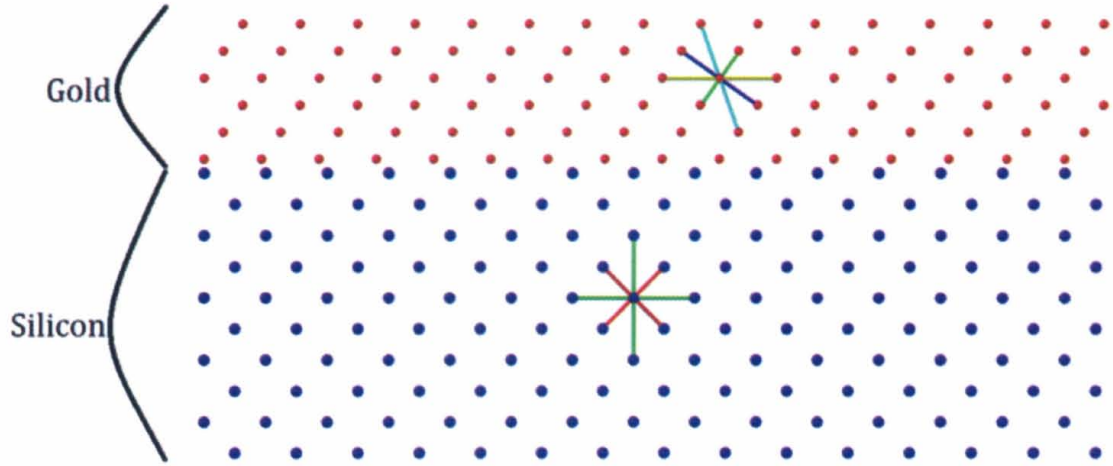


Figure 3.1 Gold Silicon Lattice, Silicon (blue), Gold (Red)

This code takes as input the number of silicon atoms, the number of gold layers, and the destination file name. Because gold and silicon have different crystal structures it is necessary to optimize the position of the gold film on the silicon cantilever. The code alters the horizontal lattice parameter of the gold film atoms until both end boundaries fit within the required space. Modifying the horizontal lattice parameter of the gold film results in the slight compression or elongation of the film as compared to its natural length. However, increasing the width of the sample reduces the deviation of the horizontal lattice parameter from its expected value. Figure 3.2 shows the decrease in the deviation as the width of the sample increases.

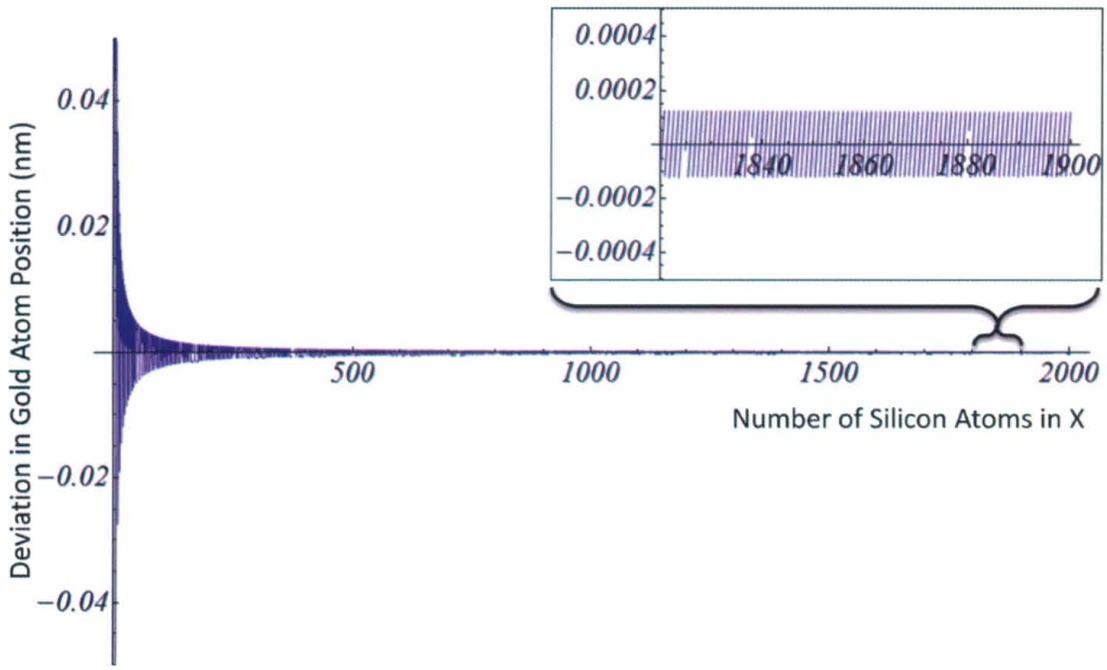


Figure 3.2: Deviation in gold atom horizontal lattice parameter on silicon substrate

The determination of the deviation in the horizontal lattice parameter of gold is calculated using the interface of the silicon atoms in the top surface of the substrate with the bottom surface of the thin gold film. The length of each of these rows is required to be the same to force the gold to fit on the silicon. Therefore the deviation of the horizontal lattice parameter can be calculated by,

$$Horizontal\ Lattice\ Parameter\ Deviation = \frac{((N-1)L_S) - RINT\left(\frac{((N-1)L_S)}{L_G}\right)L_G}{RINT\left(\frac{((N-1)L_S)}{L_G}\right)}, \quad 3.1$$

where $N$is the number of silicon atoms in the X direction, L_S and L_G is the next-nearest neighbour distances of silicon and gold respectively. Here $RINT$ rounds the value inside brackets to the nearest integer.

In equation 3.1, $(N-1)L_S$ is the total length between boundary silicon atoms in the layer. While $\frac{((N-1)L_S)}{L_G}$ is the number of gold bonds required to span the same distance this number must be an integer so the boundaries will match, therefore it is rounded to the nearest integer. Multiplying this number by the length of a normal gold bond gives the total length of the gold surface for the number of bonds. Subtracting the total length of gold from the total length of silicon gives the difference in the lengths, dividing by the number of gold bonds gives the amount that each bond must be elongated or compressed to fit on the silicon surface.

In this work the samples analysed were 1,842 silicon atoms wide in the horizontal direction which resulted in having 2002 gold atoms in the X direction. Therefore the amount of

deviation in the gold atom bond is 10^{-13}m, which is an acceptable amount of positional error that is of the same order as the resolution of the initial atom position coordinates.

3.1.1 Lattice Creation Graphical User Interface (GUI)

In order to create multiple lattices with varying surface roughness, and to provide a generalized interface to perform all of the related tasks, a java based Graphical User Interface (GUI) was created. This same interface was also used to generate the final plots, modify the code for ACEnet, and submit the code to a server for processing. The code for this interface was written using NetBeans[7], and can be found in Appendix F. This code generates the following interface:

[7] NetBeans is a Integrated Development Environment (IDE) for the creation of Java desktop applications made by the Oracle Corporation

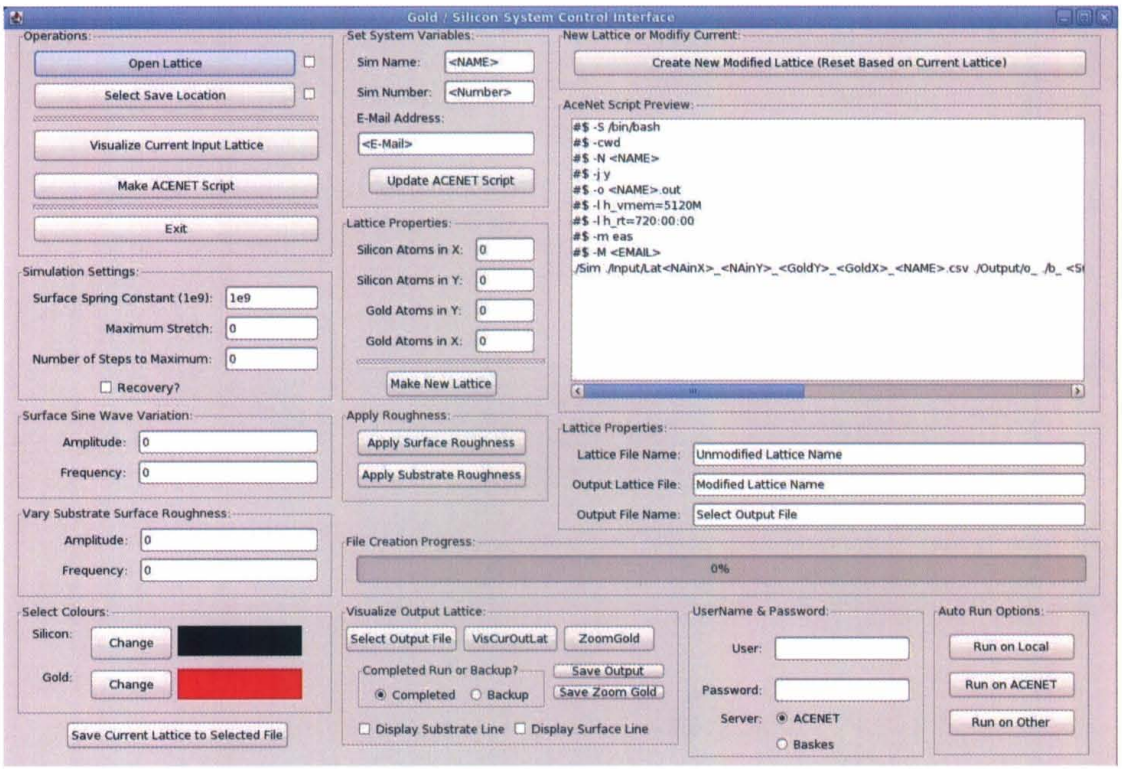


Figure 3.3: Simulation control interface.

This program modifies the input files that are required for the simulation. The input files are initially created using a program that can be found in Appendix G. This program creates a perfect 2D lattice based on the number of silicon atoms in the X direction, the number of silicon in the Y direction, and the number of layers of gold.

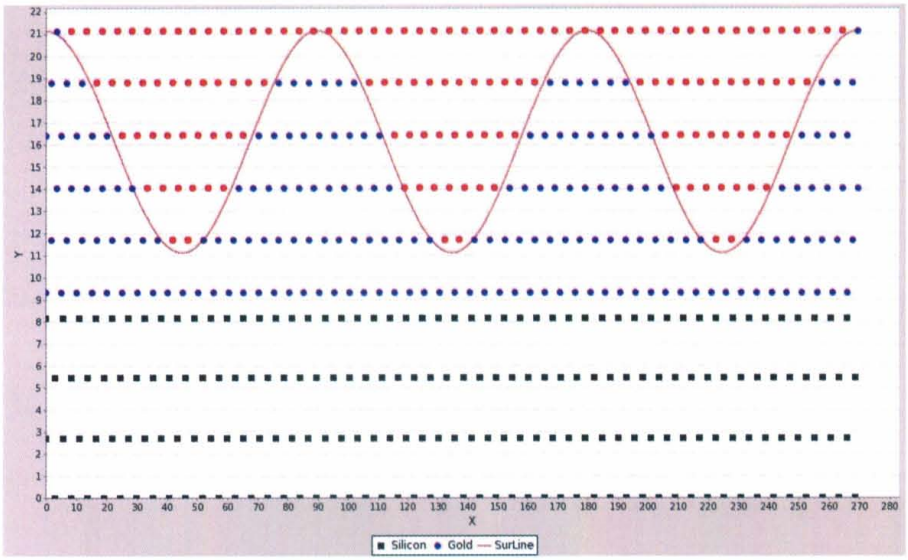
In this work a cosine function was used to define the surface roughness of the gold film. Based on the user defined frequency and amplitude of the cosine function that defines the surface the wavelength is calculated so that a maximum occurs at the boundaries. Forcing the maximum to be at the boundaries makes it possible to use variable boundaries as discussed in

section 3.3.1. Figure 3.4a illustrates the application of the surface line, for this case it has a wavelength of 1/3 μm and an amplitude of 1 nm. Any atom located above this line is declared inactive.

The red atoms shown in Figure 3.4a have been declared to be inactive. Any spring located within the red zone contributes zero energy to the total lattice energy. Therefore the equivalent system is as shown in Figure 3.4b. The roughness simulated in each of the samples is realistic for the dimensions of actual gold deposited on silicon cantilevers [27].

Figure 3.5 shows how the number of bonding sites decreases as the frequency and amplitude of the surface is changed. The number of bonding sites decreases as the frequency decreases, and also decreases as the amplitude increases.

a)



b)

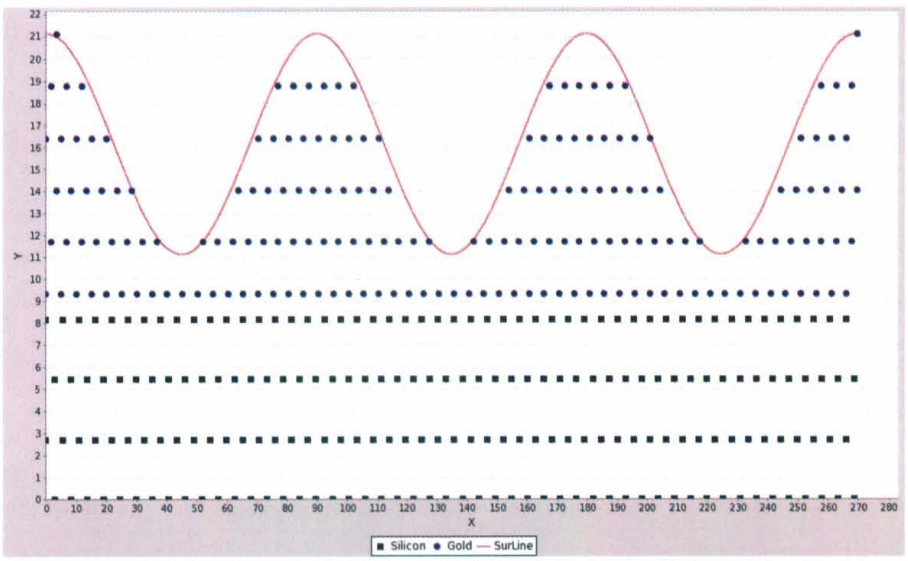


Figure 3.4: a) Application of Surface Fitting Function, b) Red atoms above line have been declared virtual

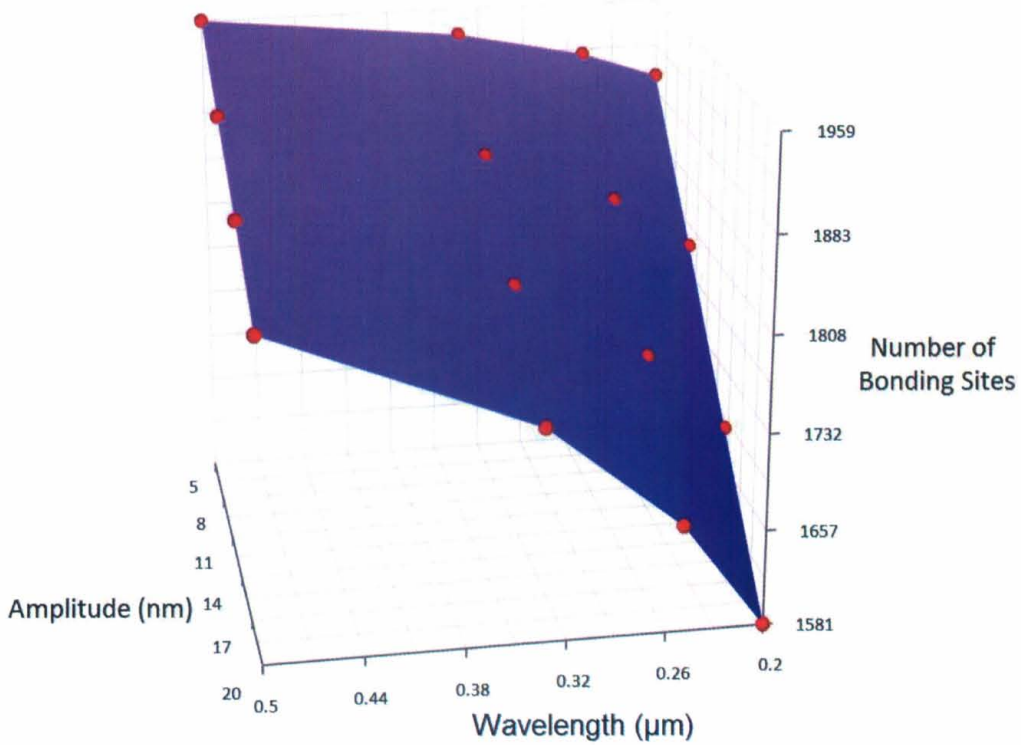


Figure 3.5: The number of bonding sites decreases as the frequency decreases and also decreases as the amplitude increases. Red dots correspond to the simulations with a film thickness of 23.5 nm.

3.2 Movement of Atoms

In order to model molecular adsorption on the thin gold film an algorithm for moving the atoms for the simulated annealing algorithm was developed. This algorithm increases the natural spring length (L_{Spring}) of the surface springs, which changes the minimum energy radius to be at the desired bond elongation. In order to reduce the energy of the system the atoms need to be moved to eliminate the new compression of the surface.

Within the one micron section of the cantilever under consideration there are a total of 6,785,928 silicon atoms as well as 200,200 gold atoms. The algorithm attempts to determine the minimum lattice energy of a system that has 13,972,256 variables (X and Y coordinates). It is computationally expensive to vary the position of each atom randomly as dictated by the simulated annealing algorithm. The execution time can be reduced beginning with the reduction of the number of free variables.

The first assumptions that are made is that the thickness of any thin cross-section of cantilever will not change and that for a cantilever undergoing bending that the shape of the beam does not change along its width [28]. This is due to the fact that any stress or strain that does not contribute to the elongation of the cantilever along the longitudinal axes can be relieved as nothing exists to hold any residual stress or strain in these directions. Therefore any compression or expansion of the beam is released immediately. This forms the basis for Stoney's equations. Bucciarelli also states that for a "beam in pure bending, plane cross sections remain plane and perpendicular to the longitudinal axis"[28]. This allows the lattice to

be divided, both silicon and gold parts, into distinct columns as shown in Figure 3.6, in this case the columns of atoms take on the role of the planes in the typical deflection model. This simplification reduces the number of independent variables from 13,972,256 down to 3,684 or twice the number of silicon atoms in the X direction.

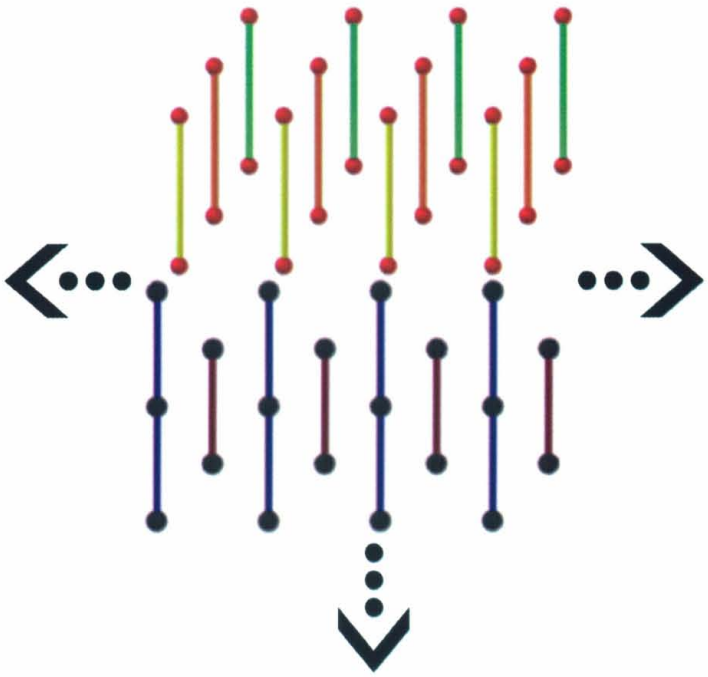


Figure 3.6: Atom Columns for Movement, red spheres are gold atoms while grey spheres are silicon atoms

The function for moving the atoms randomly sets the angle between the columns, moving every subsequent column by the same angle. This algorithm for moving the atoms is implied within Stoney's equation since the surface of the cantilever has a constant curvature.

Referring to Figure 3.7 for a visual representation of the algorithm. The code for the movement starting at column 2 is:

Step 1 Generate pseudo-random number in the range (0,1)

$$U = RANDOM(0,1)$$

Step 2 Multiply random number by the maximum possible rotation angle

$$U_N = U * (\max\ rotation\ angle)$$

Step 3 Increment current and subsequent column rotation angles by U_N

Step 4 Move to next column and repeat steps 1-4, until no columns remain

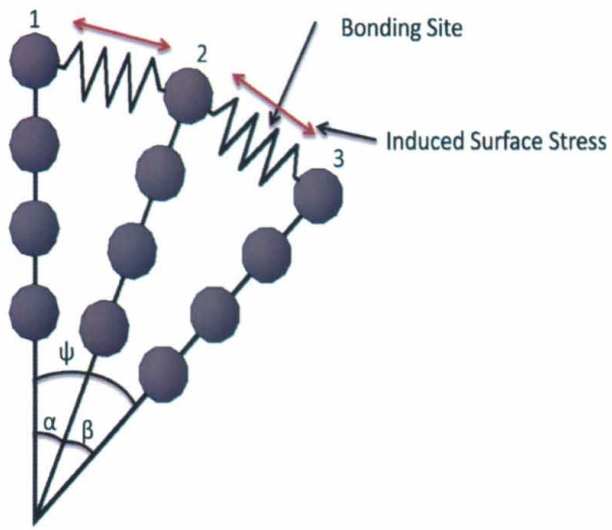


Figure 3.7: Algorithm has randomly chosen α and β and therefore has set the slope of column 3 to be equal to ψ=α+β

Once the angle of each column has been set the algorithm then rotates the columns of atoms about the global origin. This simulates the deflection of the cantilever.

3.3 Crystal Lattice Boundaries

The one micron section of cantilever to be considered within this calculation is a section of a much longer cantilever. Modeling the entire cantilever is done through the process of repeatable and variable boundary conditions.

In this system the first and last atoms in each row are considered to be the atoms that exist within the boundary. These atoms form the basis for the lines that define the boundary. The boundary atoms are fitted to a straight line using a fitting algorithm that was modified from the one found in Numerical Recipes in C (NR). This NR algorithm was modified to remove the weighted code. The code was also modified to use the LatticeArray data structure that contains all of the information on the lattice. This modified code can be found in Appendix A. The lines that define the boundaries are determined by a line of best fit of the atoms in each boundary. If these lines are vertical or near vertical the calculated slope will be very high. In fact there is nothing in the NR code that guarantees that the resulting lines will be sensibly defined in the case where the calculated lines are vertical or near vertical.[8]

[8] The compiler may allow the special INF value to be assigned

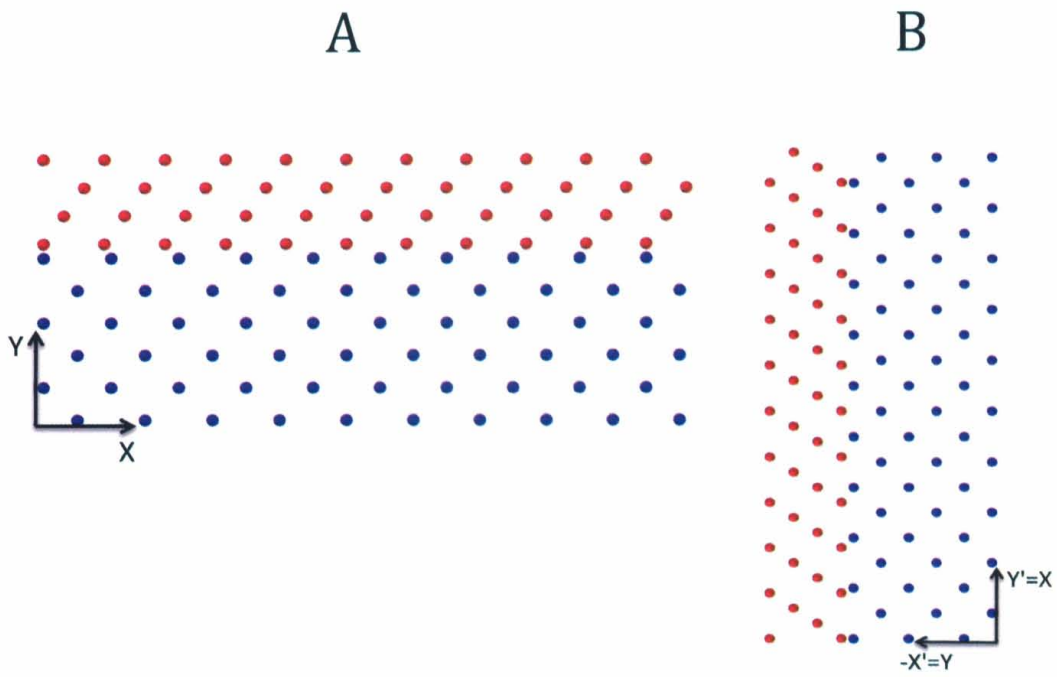


Figure 3.8: A) Un-rotated Lattice B) Rotated Cantilever with new prime coordinate system. The coordinates of the atoms are rotated using a rotation matrix by 90 degrees, this prevents the creation of infinite slopes once the boundary lines are calculated. The calculations are carried out as if the rotation never occurred.

The lattice and its coordinate system can be seen in Figure 3.8 A. To avoid this situation and any situation where the slope becomes very large, after the lattice is inputted and the springs within the lattice are assigned, the lattice is rotated by -90°. This results in the new lattice as can be seen in Figure 3.8B. This coordinate rotation changes the large slopes in the original lattice to small slopes within the rotated lattice thus preventing the creation of infinite slopes.

3.3.1 Variable Boundary Rotation

The silicon cantilever is expected to bend with a constant curvature. Therefore to model the boundary interaction it is possible to use the intersection point of boundary lines as defined above as a temporary origin about which the boundaries are rotated. If the slopes fitted to the boundary atoms are large or infinite then the intersection point may not be able to be determined.

This procedure creates a set of variable periodic boundary conditions. The establishment of periodic boundary conditions (PBCs) is often used to simulate a much larger systems by only considering a smaller sub-section of the system that is located away from the edges of the system. PBCs can only be used when the system can be broken down into a periodic array of similar systems. These simulations were designed to use a variable set of periodic boundaries to allow for the cantilever to naturally undergo deflections.

To model the boundary interactions, atoms located on the left boundary are rotated to create an image of themselves on the right boundary of the sample, and vice versa for the atoms located on the right boundary. This is illustrated in Figure 3.9.

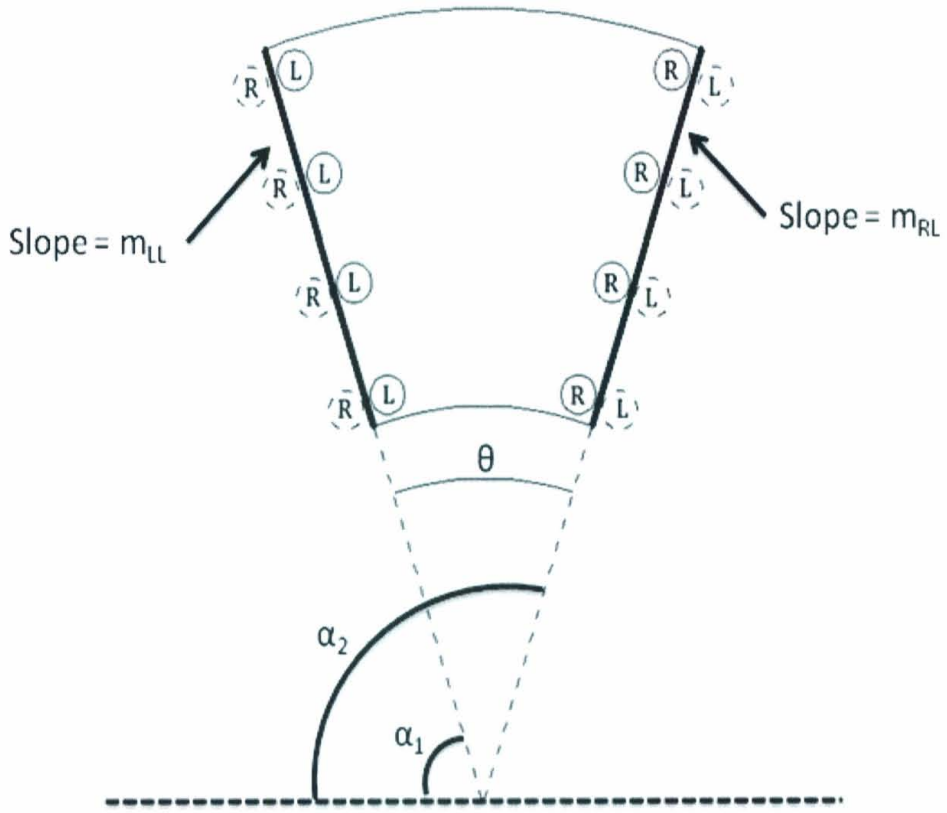


Figure 3.9: Boundary Rotation, dashed circles are the image of the boundary atom located at the next boundary

To determine the boundary rotation, the interception point of the boundary lines must be calculated. This interception point along with the slopes of the lines allows for the calculation of the rotation angle that each boundary needs to rotate. The boundary line intersection point is determined by

$$x = \frac{b_{LL} - b_{RL}}{m_{RL} - m_{LL}}$$

and

$$y = m_{RL}\, x + b_{RL}. \qquad 3.2$$

Where m and b are the slopes and intercepts respectively, with RL and LL representing Right Line and Left Line (Figure 3.9). This set of XY coordinates becomes the new temporary origin about which the boundaries are rotated. This rotation is done for any spring that extends across the boundary.

In order to perform the rotation of the boundary atoms the angle between the right and left boundary lines must be known. This operation is preformed multiple times within the simulated annealing algorithm so it would be of great benefit to determine the angle with minimal additional calculations.

Referring to Figure 3.9, it can be seen that

$$\alpha_2 = \alpha_1 + \theta \qquad 3.3$$

or upon rearrangement,

$$\theta = \alpha_2 - \alpha_1. \qquad 3.4$$

Therefore it can be said that

$$\tan\theta = \tan(\alpha_2 - \alpha_1). \tag{3.5}$$

Using the trigonometric identity for the tangent of the difference between two angles yields

$$\tan\theta = \frac{\tan\alpha_2 - \tan\alpha_1}{1 + \tan\alpha_1 \tan\alpha_2}. \tag{3.6}$$

By definition

$$m_{RL} = \tan\alpha_2,$$

$$m_{LL} = \tan\alpha_1, \tag{3.7}$$

with

$$m_{RL} > m_{LL}.$$

Therefore

$$\theta = \tan^{-1}\left(\frac{m_{RL} - m_{LL}}{1 + (m_{LL}m_{RL})}\right) \quad 0 \leq \theta \leq \frac{\pi}{2}. \tag{3.8}$$

From equation 3.7 the slopes of the lines are known due to the fitting of the boundaries. This results in a method by which to calculate the required rotation angle.

Chapter 4 : Results and Discussion

In this chapter the results from the simulated annealing minimization algorithm are presented. Initially each algorithm was executed on a very small lattice of approximately 100 to 1000 atoms. This was done to aid in the process of debugging the algorithm, sorting out memory overhead, and making each algorithm efficient. After all of the bugs were removed and the program was tuned to be as efficient as possible, a collection of lattices with varying film thicknesses, frequencies, and amplitudes were created and processed (each lattice had its own instance of the program running in parallel and took over 3.5 months to complete). The minimum energy configurations were determined for each lattice, and the results are presented below.

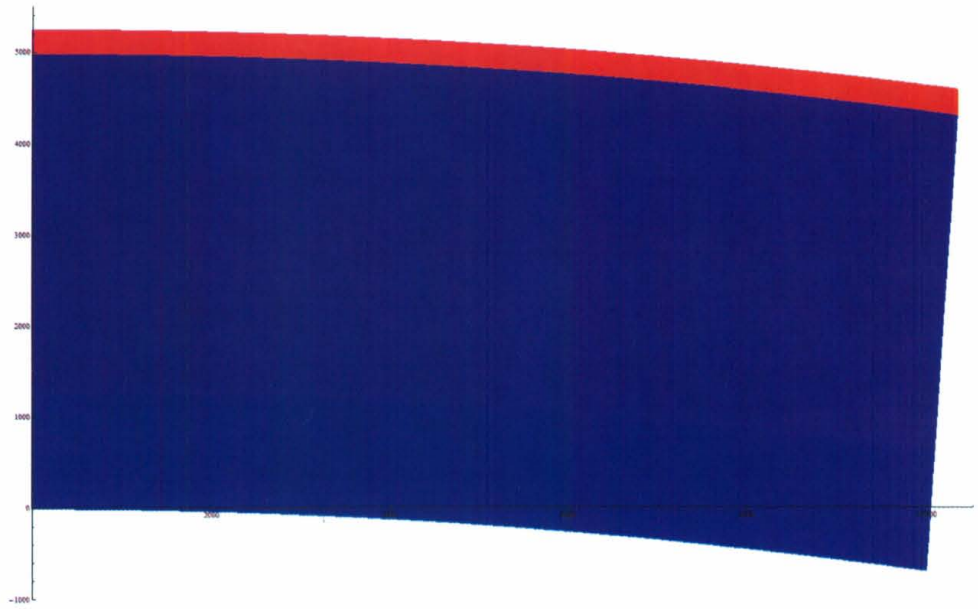
4.1 Simulated Annealing

4.1.1 Comparison to Finite Element Analysis

To verify the output of the algorithm an ANSYS®[9] finite element analysis was conducted for an atomically flat lattice that consisted of a silicon block that was 0.5 μm x 1 μm with a 10 nm thick gold film. The same system was also analyzed using the software found in appendix E. The deflection of the sample calculated by ANSYS® was 64.4 nm while the calculated value using the simulated annealing algorithm was 64.6 nm. ANSYS® uses finite element analysis in order to solve for the resulting deformation of an object and is accepted both commercially and academically as being accurate (There are many published papers that use ANSYS® as their primary means of performing calculations). In order to perform its calculations ANSYS® needs to mesh an object in a triangular grid, the grid spacing varies depending upon the surface. This grid does not map very small details very well, but can map smooth surfaces at the scale used in this work. Since the value calculated by ANSYS® for the deflection is approximately equal to that calculated by the simulation, the results from the simulation should be approximately correct. See Figure 4.1 for a comparison between the deflection calculated by ANSYS® versus the deflection calculated by the simulation.

[9] ANSYS is a general purpose commercial finite element modeling package.

a)



b)

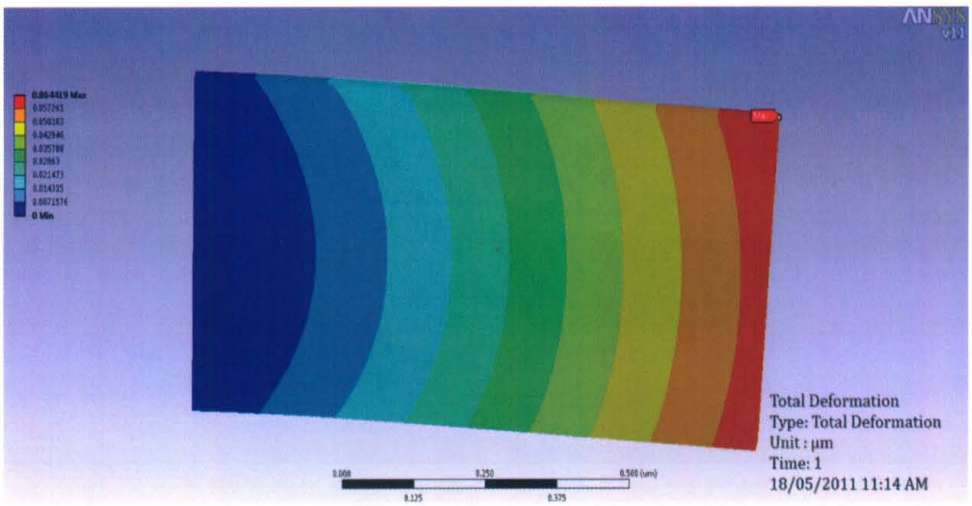


Figure 4.1: The deflection calculated by the simulation (a) was 64.6nm while the deflection calculated by ANSYS® (b) using the properties (Young's modulus, Poisson's ratio) of the materials was 64.4nm.

4.1.2 Simulation Output

Once the program output was verified the program was used on a variety of lattice configurations. Excluding the test cases 26 separate unique configurations of the lattice were analysed. A summary of the samples studied in this work are shown in Table 2. This table gives the initial film thickness, the frequency, amplitude, and the final calculated end deflection of the cantilever. Due to the presence of the surface features the thickness is assumed to be the average film thickness, equation 4.1.

$$Average\ Film\ Thickness = Maximum\ Film\ Thickness - \left(\frac{1}{2}\right) Amplitude \qquad 4.1$$

The final end defection was determined from the position of the last silicon atom in the first row (Right hand side) of the lattice. This deflection is measured from the initial position of the atom, since the initial position of the first row of atoms in the lattice is along the y=0 axes. The samples are designated by AF in Table 2 are for systems that are atomically flat.

Table 2: Summary of Simulations

Film Thickness (Average)	Film Thickness (Maximum)	Wavelength (μm) AF – Atomically Flat	Amplitude (nm)	Final End Deflection (nm)
11.7 nm	11.7 nm	AF	0	-5.62155
9.2 nm	11.7 nm	1/2	5.0	-16.2676
9.2 nm	11.7 nm	1/3	5.0	-21.7456
9.2 nm	11.7 nm	1/4	5.0	-27.8265
9.2 nm	11.7 nm	1/5	5.0	-34.6724
23.5 nm	23.5 nm	AF	0	-5.68008
21.0 nm	23.5 nm	1/2	5.0	-16.2133
21.0 nm	23.5 nm	1/3	5.0	-21.6955
21.0 nm	23.5 nm	1/4	5.0	-27.8736
21.0 nm	23.5 nm	1/5	5.0	-32.2895
18.5 nm	23.5 nm	1/2	10.0	-27.7857
18.5 nm	23.5 nm	1/3	10.0	-39.7912
18.5 nm	23.5 nm	1/4	10.0	-50.8246
18.5 nm	23.5 nm	1/5	10.0	-63.3947
16.0 nm	23.5 nm	1/2	15.0	-39.8358
16.0 nm	23.5 nm	1/3	15.0	-56.7751
16.0 nm	23.5 nm	1/4	15.0	-73.5547
16.0 nm	23.5 nm	1/5	15.0	-92.3374
13.5 nm	23.5 nm	1/2	20.0	-51.0599
13.5 nm	23.5 nm	1/3	20.0	-73.9471
13.5 nm	23.5 nm	1/4	20.0	-100.606
13.5 nm	23.5 nm	1/5	20.0	-122.372
50.0 nm	50.0 nm	AF	0	-4.71116
75.0 nm	75.0 nm	AF	0	-4.73244
100.0 nm	100.0 nm	AF	0	-4.7361
150.0 nm	150.0 nm	AF	0	-3.86491

The results of the simulations can be seen in Figure 4.2 to Figure 4.6. These correspond to the film thickness of 11.7 nm. The silicon substrate has been removed from the plots for clarity since it is a few orders of magnitude larger than the gold film. Due to the number of atoms used in the calculations the plots appear to be solid, however they are distinct points.

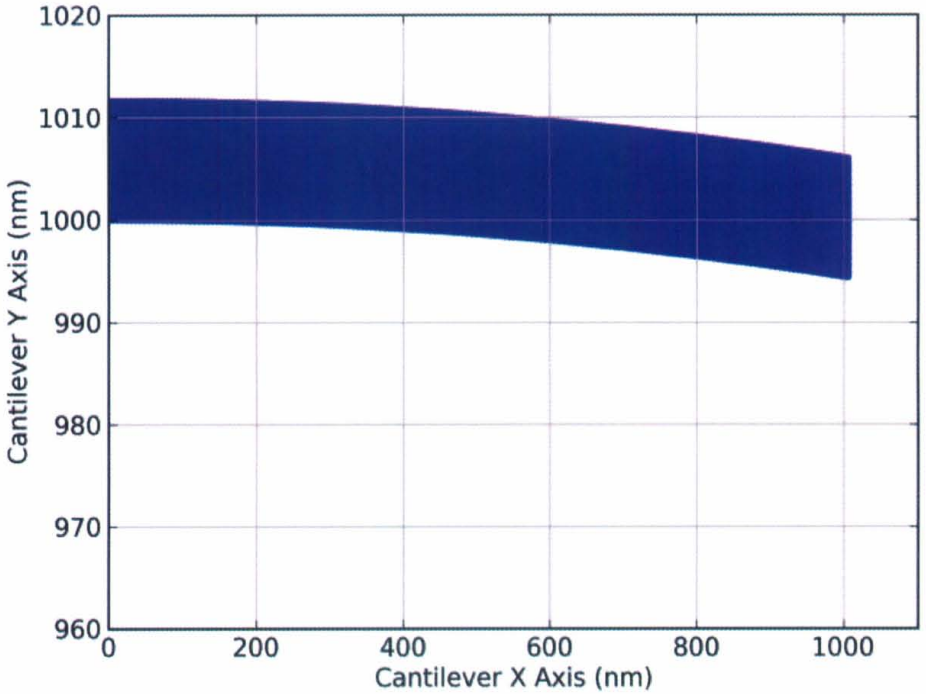


Figure 4.2: Wavelength AF, Deflection: -5.62 nm.

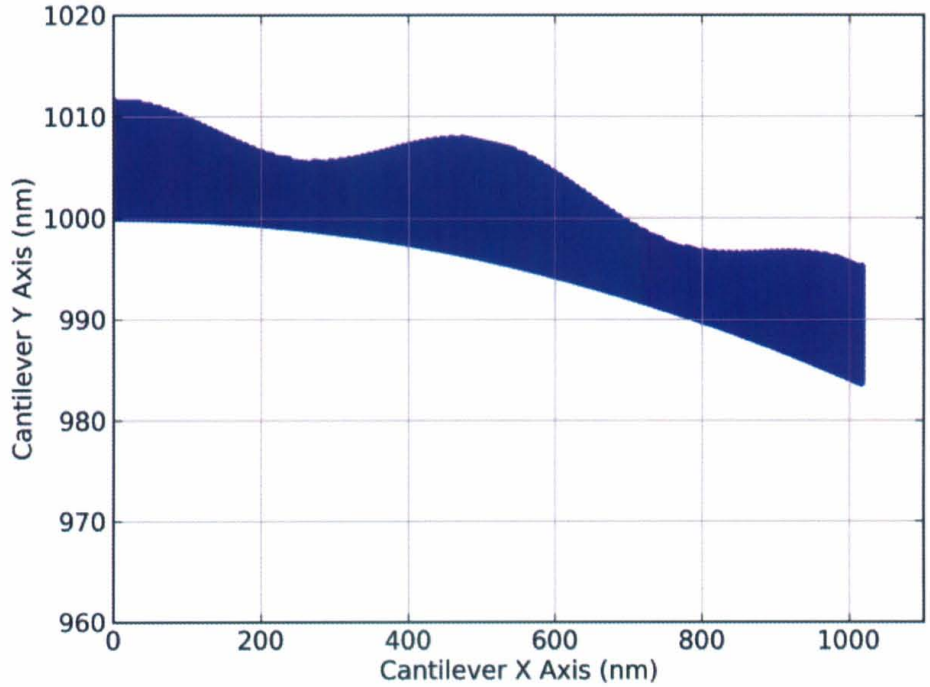


Figure 4.3: Wavelength (1/2) μm, Deflection: -16.26 nm.

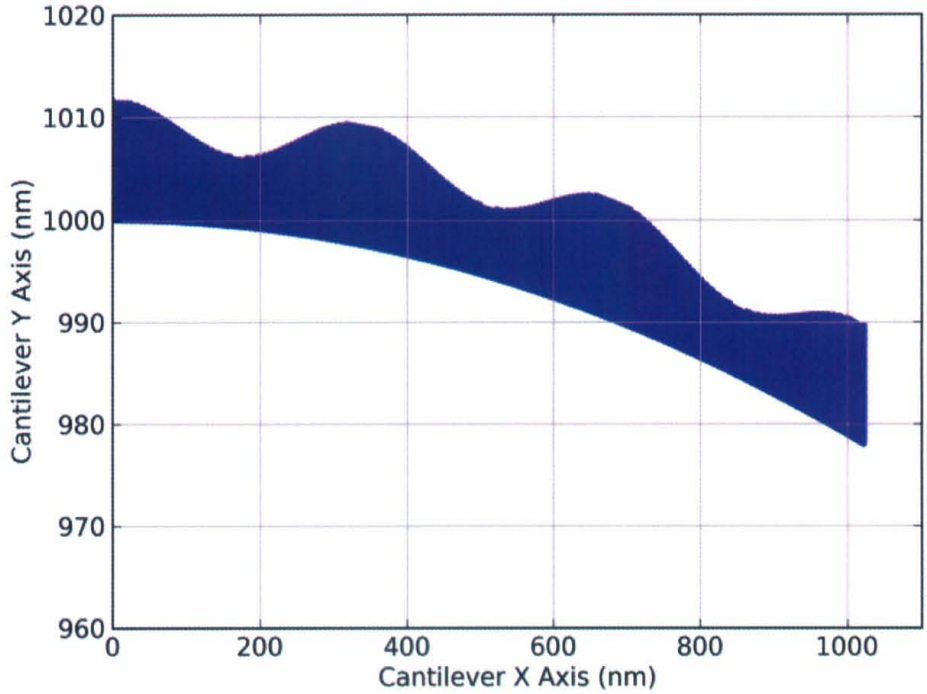


Figure 4.4: Wavelength (1/3) µm, Deflection: -21.74 nm.

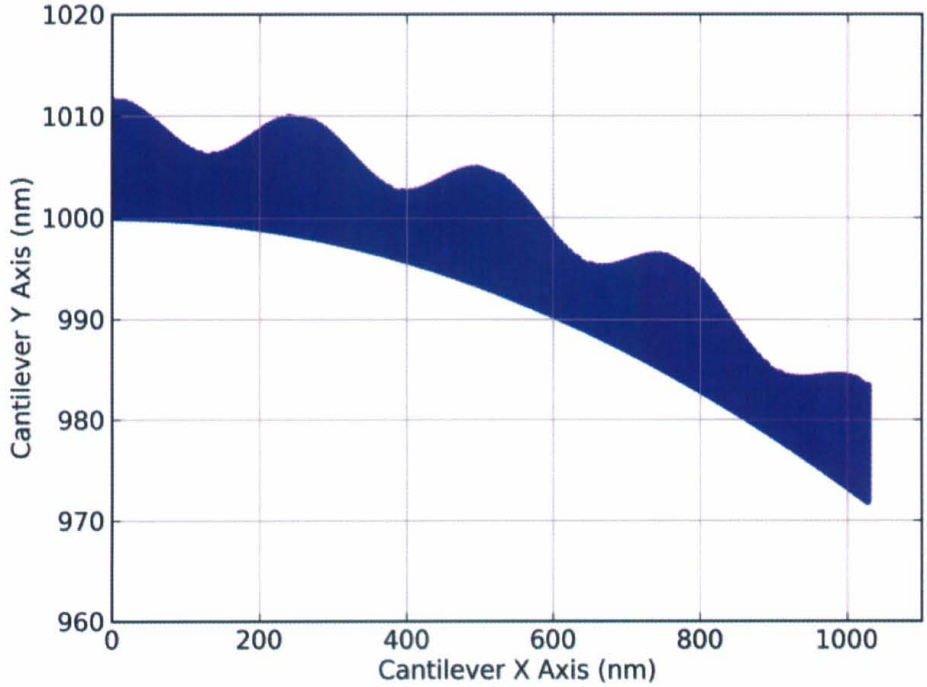


Figure 4.5: Wavelength (1/4) µm, Deflection: -27.82 nm.

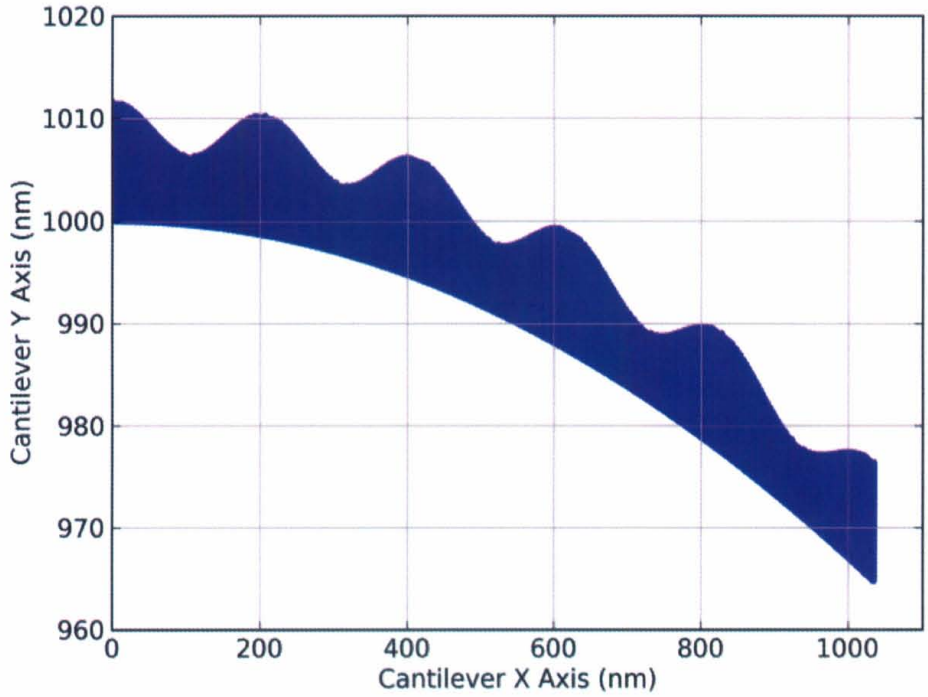


Figure 4.6: Wavelength (1/5) μm, Deflection: -34.67 nm.

In order to compare systems the bottom surface of the each lattice was plotted. The plots shown as Figure 4.7 to Figure 4.11 show the bottom silicon layer for samples with the same film thickness and amplitude but different frequencies. Figure 4.12 is similar to the series of figures (Figure 4.7 to Figure 4.11) with the exception that instead of looking at the change in wavelength this figure looks at the change in surface amplitude.

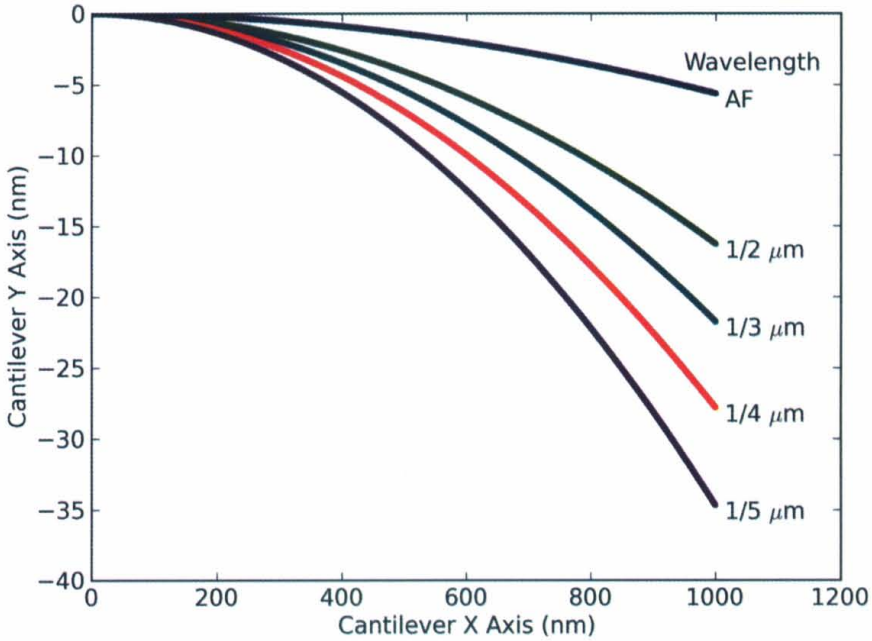


Figure 4.7: Max Film Thickness 11.7 nm, Amplitude 5.0 nm.

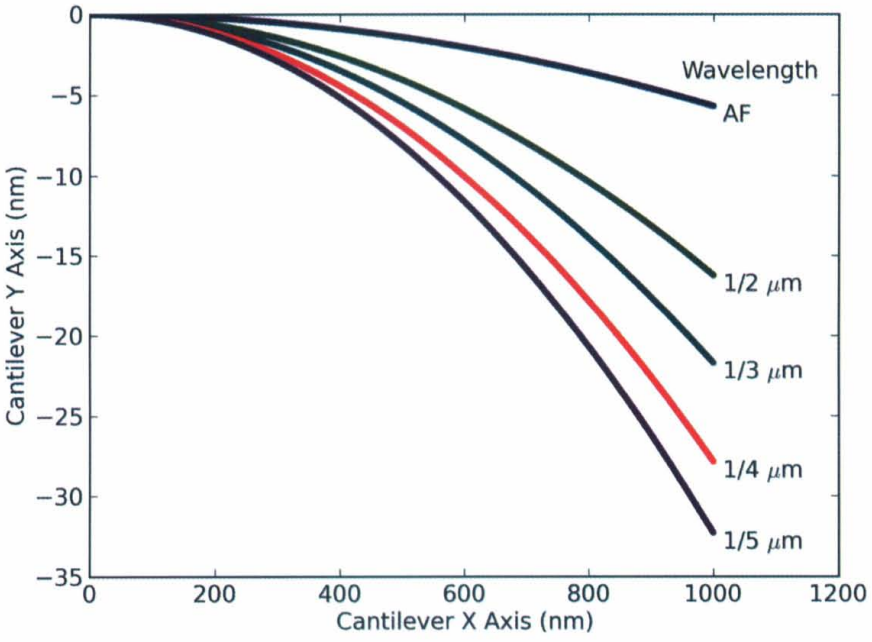


Figure 4.8: Max Film Thickness 23.5 nm, Amplitude 5 nm.

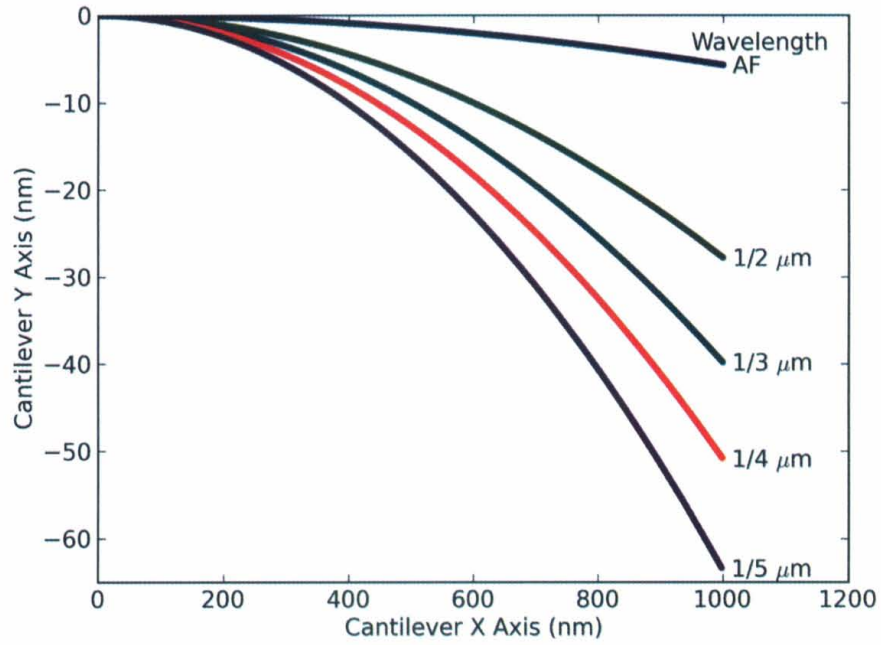


Figure 4.9: Max Film Thickness 23.5 nm, Amplitude 10 nm.

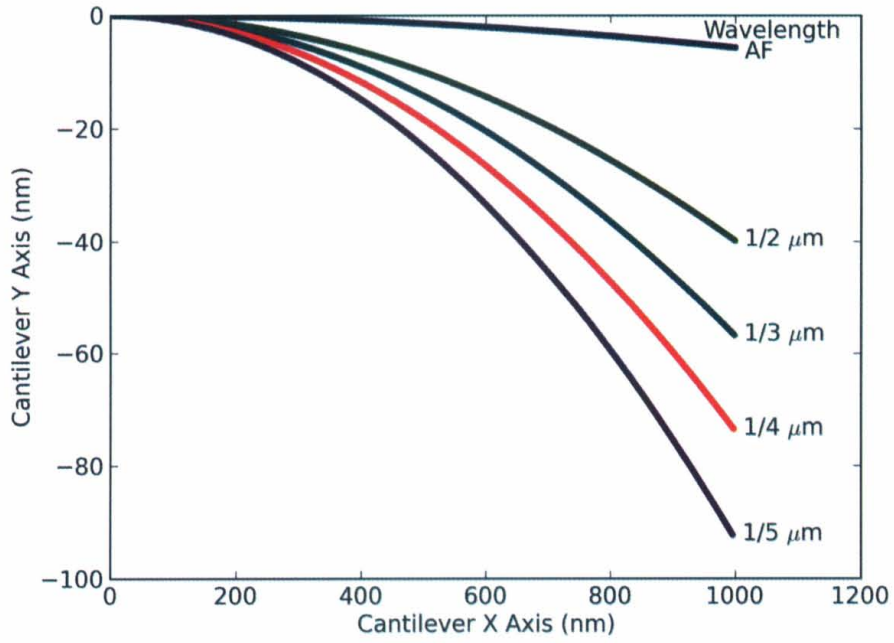


Figure 4.10: Max Film Thickness 23.5 nm, Amplitude 15 nm.

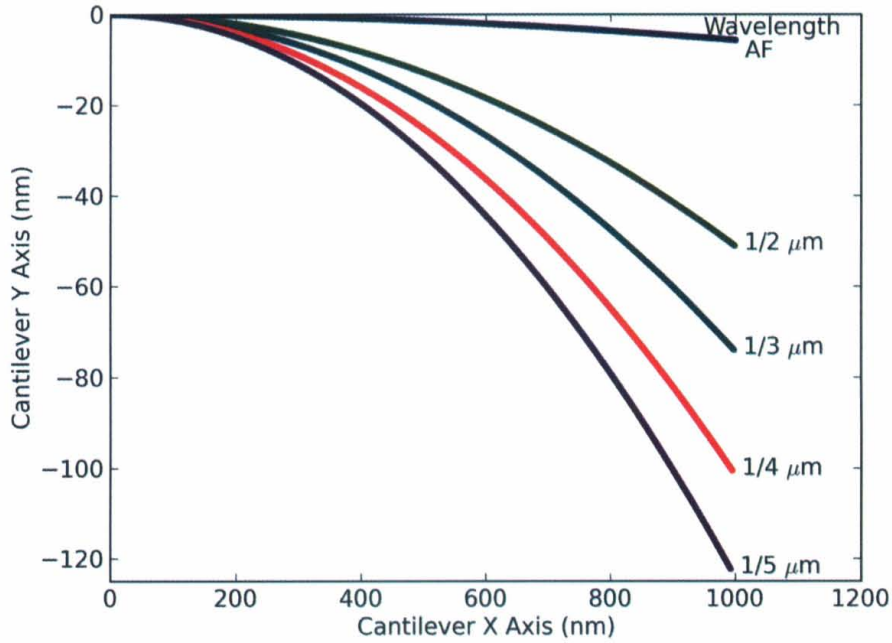


Figure 4.11: Max Film Thickness 23.5 nm, Amplitude 20 nm

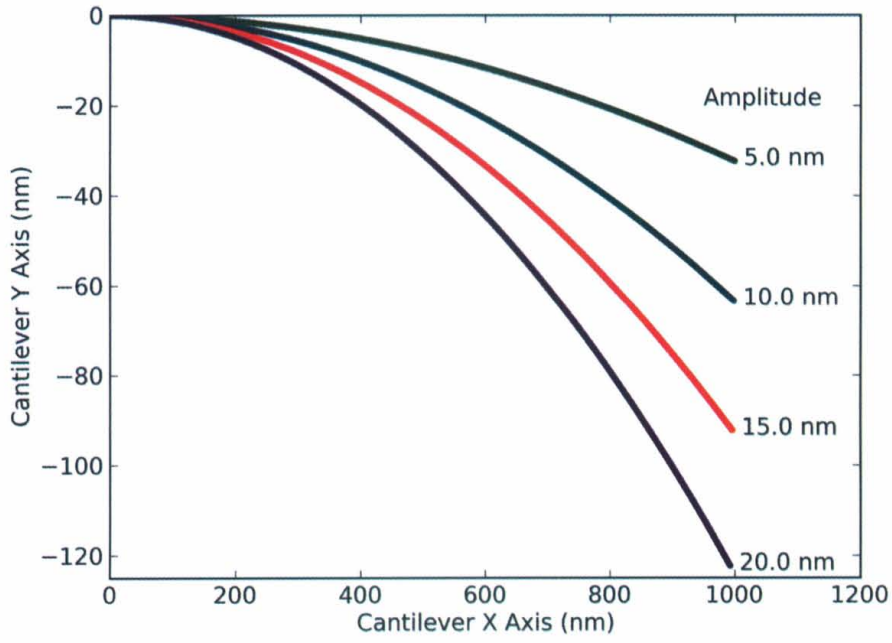


Figure 4.12: Max Film Thickness 23.5 nm, Wavelength (1/5) μm

4.2 Discussion

The total amount of cantilever deflection due to the adsorption of an alkanethiol SAM is a source of contention between various published papers. This controversy applies to both experimental and theoretical publications [5][4][6][9]. In the work of Godin et al., they conducted a study on the adsorption of alkanethiols on gold coated cantilevers [4]. In this work the authors analysed the surface stress effects of the formation of the alkanethiol SAM on the surface, including looking at the difference between the three phases of the SAM (stacked lying-down, un-stacked lying-down, and standing up). It was found that the sensitivity of the cantilever (the amount of deflection) was highly dependent on the thin film morphology where cantilevers that had surfaces that consisted of small grains (rough) deflected less than their larger grain counterparts[4].

Mertens et al. also studied the surface stress responses of cantilever based sensors to molecular adsorption. In their work they used alkylthiol mercaptohexanol (MCH) as opposed to alkanethiol SAMs [6]. The adsorption of MCH on the thin gold film resulted in an average deflection that was larger for rougher surfaces [6]. Therefore they determined that for cantilevers whose gold film was smoother (larger grains) deflected less than ones with rougher surfaces [5] [6].

From Figure 4.7 to Figure 4.11 it can be seen that as the wavelength of the surface decreases (roughness increases) the total deflection of the lattice increases. This indicates that the wavelength of the thin film surface profile affects the deflection of the cantilever. Referring

to Figure 4.7, the deflection increases by 18.36 nm as the wavelength is reduced from 1/2 to 1/5 μm. This was the same trend for each of the samples studied in this work, as the wavelength decreased the total end deflection increased. Therefore the cantilevers with the smallest wavelength (greatest roughness) deflect more.

The amplitude also plays a role in the deflection of the cantilever. This is shown in Figure 4.12. In this case the studied systems had the same maximum film thickness as well as the same wavelength. The amplitude was varied from 5.0 to 20.0 nm in 5.0 nm increments. The final end deflection for the system increased as the amplitude of the surface undulation increased. The increase in the amplitude from 5.0 to 20.0 nm resulted in an increase in the end deflection of 90.1 nm.

Hence the deflection of a cantilever is dependent upon both the wavelength and amplitude. From Figure 3.5 it is shown that the number of bonding sites decreases as the wavelength decreases and/or the amplitude increases. This would imply that the total amount of surface stress that could be applied would also decrease, as the stress is only applied at the bonding sites. Referring to Stoney's equation (2.17) if the applied surface stress is reduced the total deflection of the cantilever is also reduced. Therefore the expected result based upon Stoney's equation is that cantilevers with increased roughness deflect less than their smoother counterparts.

The results shown here indicate the opposite behaviour as expected by Stoney's equation and the reduction of bonding sites. The results demonstrate that for samples with

larger roughness the total amount of end deflection increases as the roughness increases. Why then did the simulation report the opposite of the predicted theory? Stoney's equation depends on the fact that the surface stress is uniform over the surface. For rough surfaces this is no longer true since the induced surface stress, originating from the bonding sites, varies over the surface due to the loss of bonding sites. Therefore it becomes necessary to consider more than just the application of Stoney's equation [9].

The results shown in Figure 4.7 to Figure 4.11 demonstrate that the profile of the sample plays a large role in the total expected deflection indicating that the geometry of the top surface of the thin gold film affects the transmission of the surface stress into the film and substrate.[9] The stress experienced by an object must be continuous throughout the object.[29] However, in the presence of flaws, the resulting stress must be routed around the flaw to maintain the continuity of the stress [29]. These flaws are responsible for fatigue based failures in an object. The reason why some objects fracture is due to the concentration of stress around flaws [29]. This increased local stress can alter where the material is on the stress-strain curve, possibly pushing it out of the elastic region.

The application of the cosine function to the thin film introduces stress raisers into the surface. A stress raiser is a region in a material that forces the stress in an object to be "concentrated" in a localized region. The presence of stress raisers in a surface will change the behaviour of an object as a whole [29]. The concept of stress raisers is normally applied to fatigue and crack analysis. At positions far removed from cracks the stress is just the nominal stress, provided that the applied stress is below the elastic limit. However, in the vicinity of

small cracks or flaws, the applied stress can be amplified beyond the levels predicted by normal strength of material analysis[10].

The discontinuities in the thin film introduced by the cosine function are approximately semi-elliptical. Therefore according to fracture mechanics the amount of stress that is concentrated at the tip of an ellipse is given by

$$\sigma_{max} = \sigma_{BF}\left(1 + 2\left(\frac{b}{a}\right)\right), \qquad 4.2$$

where a and b are one half of the length of the minor and major axes respectively, σ_{BF} is the stress experienced by the normal film, and σ_{max} is the concentrated stress at the discontinuity [29].

[10] Strength of materials refers to various methods of calculating stresses in structural members with loads, deformations and forces acting on the material.

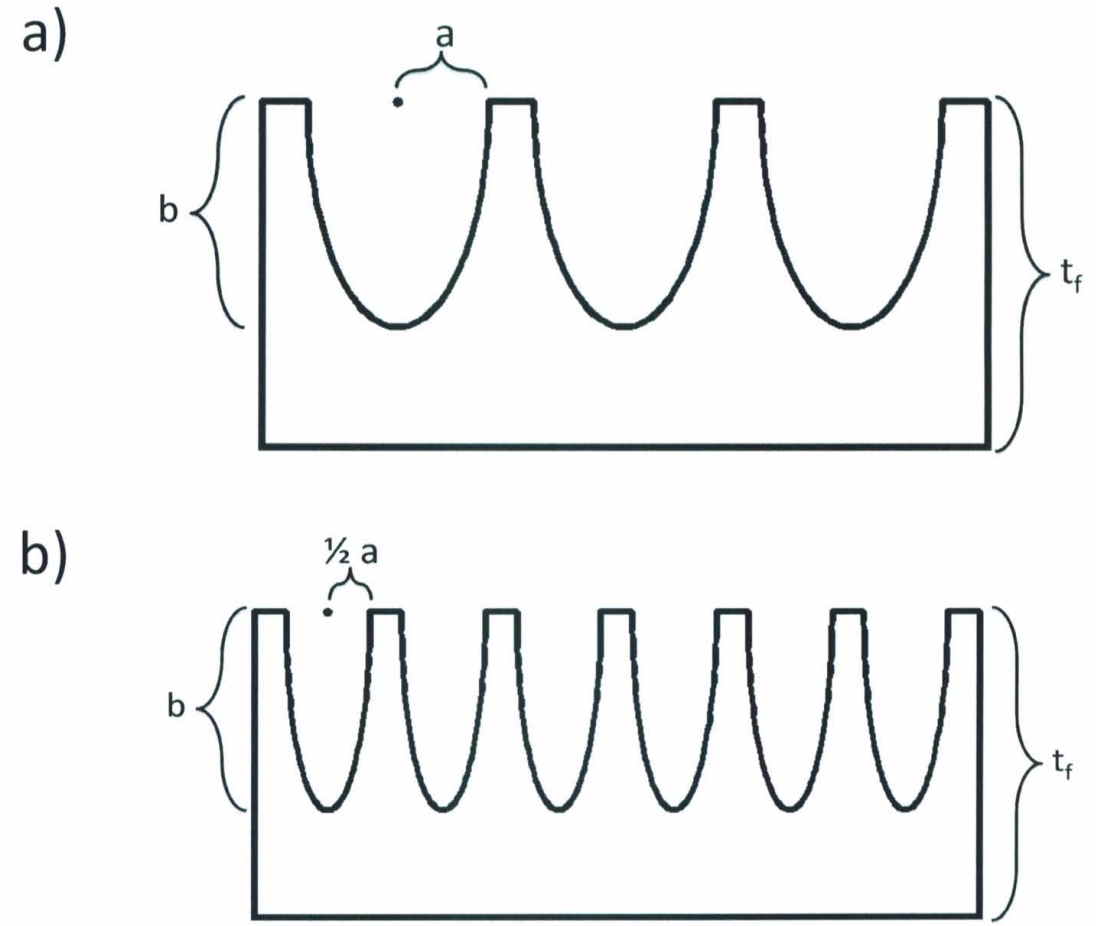


Figure 4.13: Semi-elliptical surface defects. a) Wavelength = (1/3)μm b) Wavelength = (1/6)μm

Figure 4.13 shows the approximation of the thin film for wavelengths of 1/3 μm and 1/6 μm. For Figure 4.13 the total amount of stress transferred into the substrate by the surface in a, is less then that transferred by b. This comes from the amount of stress raisers as well as the geometry of the stress raisers. The stress concentration in Figure 4.13a is

$$\sigma_{max} = \sigma_{BF}\left(1 + 2\left(\frac{b}{a}\right)\right), \tag{4.3}$$

while the stress in Figure 4.13b is given by

$$\sigma_{max} = \sigma_{BF}\left(1 + 2\left(\frac{b}{(1/2)a}\right)\right) = \sigma_{BF}\left(1 + 4\left(\frac{b}{a}\right)\right). \tag{4.4}$$

Assuming that the number of bonding sites does not change dramatically (Figure 4.13a $\sigma_{BF} \approx$ Figure 4.13b σ_{BF}) the amount of concentrated stress in Figure 4.13b will be larger than Figure 4.13a. Figure 4.14 shows the transmission of the stress through the material into the substrate.

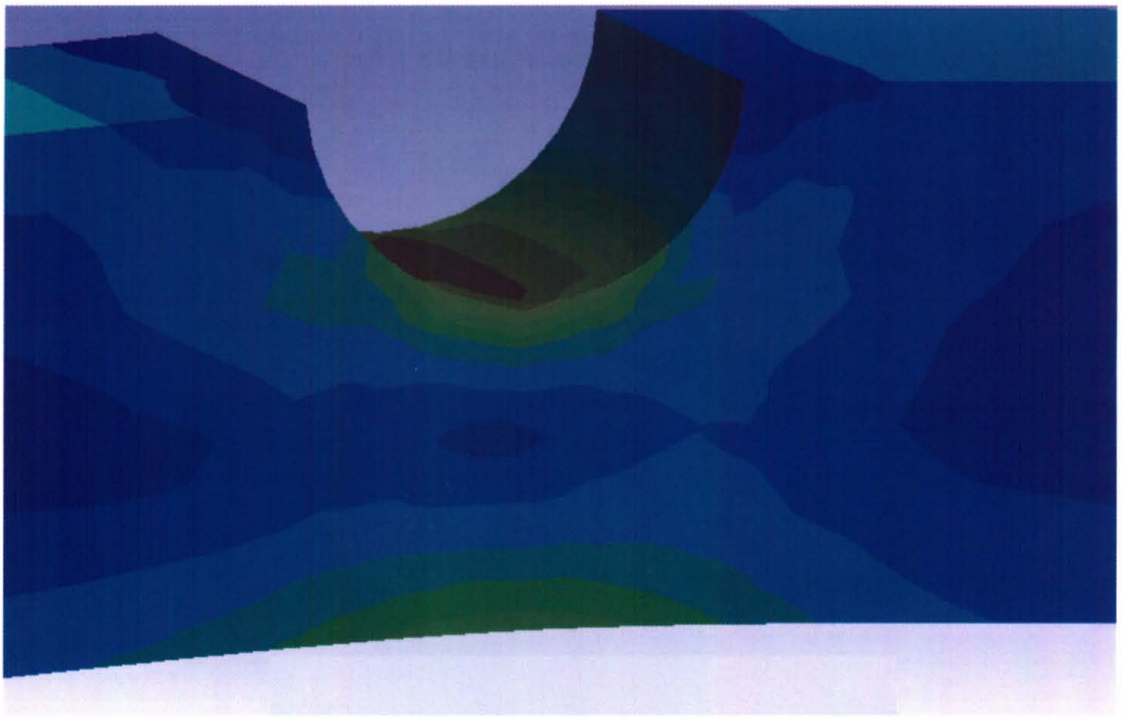


Figure 4.14: Stress is concentrated (Red) at the tip, and causes a large amount of stress to be transferred to the lower surface (yellow).

Therefore the cantilevers with an increased wavelength (Figure 4.13 a) will experience a lower amount of overall stress as compared to those that have a smaller wavelength (Figure 4.13 b). The amount of stress generated by the inclusion of the stress raisers overcompensates for the loss of the bonding sites on the surface. Therefore the cantilevers that have smaller wavelengths (more surface defects) will deflect more than their smoother counterparts. This is due to the increase in the differential surface stress in the rougher filmed cantilevers. The stress raisers increase the amount of stress that is experienced by the silicon substrates top surface, this results in an increase in the differential surface stress causing an increase in the deflection.

As well, equation 4.2 demonstrates how the increase in the amplitude (b) will increase the translated surface stress and as a result will increase the deflection of the cantilever.

In the results shown here, when the wavelength of the function was decreased (more flaws) the deflection increased, as predicted by the stress concentration theory. Similarly as the amplitude is increased the theory predicts that there will be an increase in deflection, which is what is observed in the numerical simulation. Combining both the decrease in wavelength and the increase in amplitude corresponds to the increase in observed deflection between the figures (Figure 4.7 to Figure 4.11).

Chapter 5 : Conclusion

5.1 Summary

In this work the effect of surface roughness of a thin gold film on the deflection of a silicon micro-cantilever was analyzed. A simulation was developed to model the effect of surface roughness on the adsorption induced deflection of the cantilever. In order to determine the final configuration of the lattice an energy minimization was performed. Here the top surface bonds detailed in section 2.5 were stretched and the lattice was rearranged to its lowest energy configuration. The lattices used in this work were approximately 1 μm by 1 μm of silicon coated with thin gold films of varying thicknesses.

Computationally this can be considered to be a NP-hard problem. In order to determine this minimum energy a simulated annealing algorithm was used. This proved to be an adequate method for determining the minimum energy state of the studied systems. Although each

system had a very large run time (approximately 3.5 months when running in parallel[11] using OpenMP) the simulated annealing algorithm was capable of solving very large systems.

Confusion exists in the literature about the adsorption induced deflection of the cantilever. Some authors have observed that the deflection of a cantilever with a rough surface is larger than that of a similar smoother cantilever, while other authors have reported the opposite [4][6][5].

It was found in this work that by including a sinusoidal surface roughness in the mono-crystalline thin gold film the total deflection experienced by the micro-cantilever increased. It was found that by increasing the amplitude and/or decreasing the wavelength of the surface defects the final calculated end deflection of the cantilever increased.

In the case of mono-crystalline films the inclusion of surface defects alters how the stress is transmitted throughout the film and substrate. In this case the shape of the defect must be considered to explain the effect on the deflection characteristics [29].

The results of the numerical simulation correspond to the theory based on the rough surface geometry causing stress concentration at distinct points forcing more stress to be transferred into the substrate. It was found that the amplitude as well as the wavelength had an influence on the magnitude of the deflection. Decreasing the wavelength, increasing the amplitude, or both, will increase the cantilever deflection upon molecular adsorption.

---

[11] Refer to Appendix D

5.2 Future Work

This work lays the groundwork for multiple possible future projects. It would be beneficial to model a cantilever with a rough substrate to see how the substrates surface roughness affects the deflection. It would also be of benefit to model the system to include grain boundaries since they will affect how the stress is transmitted over the length of the cantilever. It may also be of benefit to model internal pours, between the thin gold film and the substrate. Another possible project would involve looking at the intermixing of the gold/silicon at the boundary layer to see what affect it has on the deflection. It may also be of benefit to change the geometry of the surface defects or generate a random surface to see how various surface profiles affect the deflection.

References

[1] J. Mertens et al., "Effects of temperature and pressure on microcantilever resonance response," *Ultramicroscopy*, vol. 97, pp. 119-126, October - November 2003.

[2] Rong-Hua Ma, Chia-Yen Lee, Yu-Hsiang Wang, and Hao-Jen Chen, "Microcantilever-based weather station for temperature, humidity and flow rate measurement," *Microsystem Technologies*, vol. 14, no. 7, pp. 971-977, July 2008.

[3] M. Godin, V. Tabard-Cossa, P. Grütter, and P. Williams, "Quantitative surface stress measurements using a microcantilever," *Applied Physics Letter*, vol. 79, no. 551, 2001.

[4] M. Godin et al., "Surface Stress, Kinetics, and Structure of Alkanethiol Self-Assembled Monolayers," *Langmuir*, vol. 20, no. 17, pp. 7090-7096, 2004.

[5] Ramya Desikan, Ida Lee, and Thomas Thundat, "Effect of nanometer surface morphology on surface stress and adsorption kinetics of alkanethiol self-assembled monolayers," *Ultramicroscopy*, no. 106, pp. 795–799, 2006.

[6] J. Mertens, M. Calleja, D. Ramos, A. Tarýn, and J. Tamayo, "Role of the gold film nanostructure on the nanomechanical response of microcantilever sensors," *Journal Applied Physics*, vol. 101, 2007.

[7] L. Y. Beaulieu, Michel Godin, Olivier Laroche, Vincent Tabard-Cossa, and Peter Grütter, "Calibrating laser beam deflection systems for use in atomic force microscopes and cantilever sensors," *Applied Physics Letters*, vol. 88, 2006.

[8] Rebecca Howland and Lisa Benatar, *A Practical Guide to Scanning Probe Microscopy.*: ThermoMicroscopes, 2000.

[9] Jörg Weissmüller and Huiling Duan, "Cantilever Bending with Rough Surfaces," *Physical Review Letters*, vol. 101, January 2008.

[10] Arvind Baskaran, Jason Devita, and Peter Smereka, "Kinetic Monte Carlo simulation of strained heteroepitaxial growth with intermixing," *Continuum Mechanics And Thermodynamics, Volume 22, Number 1.*

[11] Vitaly A. Shchukin, Nikolai N. Ledentsov, and Bimberg Dieter, *Epitaxy of Nanostructures.* Berlin/Heidelberg: Springer, 2003.

[12] Yukio Saito, Hideaki Uemura, and Makio Uwaha, "Two-dimensional elastic lattice model with spontaneous stress," *Physical Review Series B*, vol. 63, January 2001.

[13] J.W. Gibbs, *The Scientific Papers of J. Willard Gibbs, Vol. 1.* London: Longmans-Green ,

1906.

[14] G.G. Stoney, "The tension of metallic films deposited by electrolysis," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 82, no. 553, pp. 172–175, May 1909.

[15] E. Suhir, Y. C. Lee, and C. P. Wong, *Micro- and Opto-Electronic Materials and Structures: Physics, Mechanics, Design, Reliability, Packaging*. New York: Springer, 2007.

[16] S. Timoshenko, "Analysis of Bi-Metal Thermostats," *Journal of the Optical Society of America*, vol. 11, no. 3, pp. 233-255, 1925.

[17] Genki Yoshikawa, "Mechanical analysis and optimization of a microcantilever sensor coated with a solid receptor film," *Applied Physics Letters*, vol. 98, no. 17, April 2011.

[18] Luca Peliti, *Statistical Mechanics in a Nutshell.*: Princeton University press, 2011.

[19] N. Metropolis, "The beginning of the Monte Carlo method," *Los Alamos Science*, pp. 125–130, 1987.

[20] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller, "Equations of State Calculations by Fast Computing Machines," *Journal*

of Chemical Physics, vol. 21, 1953.

[21] Keith W. Hastings, "Monte Carlo Sampling Methods using Markov Chains and Their Applications," *Biometrika*, vol. 57, pp. 97-109, 1970.

[22] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. New York: Cambridge University Press, 2007.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science, New Series*, vol. 220, no. 4598, 1983.

[24] A.A. Markov, "Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga," *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete, 2-ya seriya, tom 15*, 1906.

[25] A.A. Markov, "Extension of the limit theorems of probability theory to a sum of variables connected in a chain," in *Dynamic Probabilistic Systems, volume 1: Markov Chains*.: John Wiley and Sons, 1971.

[26] Peter J. M. van Laarhoven and Emile H. L. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht: D. Reidel, 1987.

[27] Jaroslaw Drelich, Calvin L. White, and Zhenghe Xu, "Laboratory Tests on Mercury Emission Monitoring with Resonating Gold-coated Silicon Cantilevers," *Environmental Science & Technology*, vol. 42, no. 6, pp. 2072-2078, 2008.

[28] Louis L. Bucciarelli, *Engineering Mechanics for Structures.*: Dover, 2009.

[29] A.F. Liu, *Mechanics and mechanisms of fracture: an introduction.*: ASM International, 2005.

[30] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed.: Cambridge University Press, 1992.

[31] M. Lüscher, "A portable high-quality random number generator for lattice field theory calculations," *Computer Physics Communications*, vol. 79, pp. 100–110, 1994.

[32] P. L'Ecuyer, "Combined Multiple Recursive Random Number Generators," *Operations Research*, vol. 44, no. 5, pp. 816–822, 1996.

[33] P. L'Ecuyer, "Maximally Equidistributed Combined Tausworthe Generators," *Mathematics of Computation*, vol. 65, no. 213, pp. 203–213, 1996.

[34] M. Galassi et al., *GNU Scientific Library Reference Manual*, 112th ed.: Network Theory Ltd.,

2009.

[35] Makoto Matsumoto and Takuji Nishimura, *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.*: ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, 1998.

[36] OpenMP Architecture Review Board, *OpenMP Application Program Interface.*, Version 3.0 May 2008.

[37] M.D. Hill and M.R Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, no. 7, pp. 33-38, July 2008.

[38] D. Moncrieff, R. E. Overill, and S. Wilson, "Heterogeneous computing machines and Amdahl's law," *Parallel Computing*, vol. 22, no. 3, March 1996.

[39] Barbara Chapman, Gabriele Jost, and Ruud van der Pas, *Using OpenMP: portable shared memory parallel programming, Volume 10.*: MIT Press, 2007.

[40] Ronald G. Larson, *The Structure and Rheology of Complex fluids*. New York: Oxford University Press, 1999.

[41] Vincent Tabard-Cossa et al., "Microcantilever-based Surface Stress Sensors: Effect of

Morphology, Adhesion and Cleanliness of the Sensing Substrate.," *Analytical Chemistry*, vol. 79, pp. 8136-8143, 2007.

[42] D. R. Hartree, "The Wave Mechanics of an Atom with an Non-Coulomb Central Field. Part I. Theory and Methods," *Proceedings of the Cambridge Philosophical Society*, vol. 24, p. 89, 1927.

[43] W.J. Stronge and T. Yu, *Dynamic models for structural plasticity*. London: Springer, 1995.

[44] A. E. H. Love, *A treatise on the mathematical theory of elasticity*, 3rd ed. Cambridge, England: University Press, 1920.

[45] J. M. Hammersley and D. C. Handscomb, *Monte Carlo methods*, M. S. Bartlett, Ed. London: Methuen, 1965.

[46] B. Liu, Y. Huang, H. Jiang, S. Qu, and K.C. Hwang, "The atomic-scale finite element method," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, no. 17–20, pp. 1849-1864, May 2004.

[47] Elijah Polak, *Computational methods in optimization: a unified approach*.: Academic Press, 1971.

[48] P. M. Marcus, "Bending of a film-substrate system by epitaxy," *Physical Review B*, vol. 53, no. 11, pp. 7460-7465, March 1996.

[49] Filippo G. Bosco et al., "High throughput label-free platform for statistical bio-molecular sensing," *Lab on a Chip*, no. 11, pp. 2411-2416, May 2011.

Appendices

Each section of the program is grouped based upon the file in which the information is contained. The language that that section is coded in is included in the title, if applicable, since some of the relevant code has been done in different languages to take advantage of the power of the language. Code comments have been added to assist with clarity, and to attempt to explain some of the more unusual sections.

Appendix A: Computational Random Numbers

In order to perform Monte Carlo based simulations a high quality random number generator is required. Most random number generators do not create a long period set of high quality random numbers, i.e. the number of 'numbers' that can be selected before the set loses randomness is low.

It is not possible for any computer to generate truly random numbers, this is inherent within the strict logical rules that computers are based upon. The exception here of course is to use a external random number generator based upon radioactive decay, however these do not operate at speeds that are acceptable for the generation of large quantities of random numbers. Therefore computers are reliant upon pseudorandom number generators.

Pseudorandom number generators produce an approximation of a truly random number sequence, however they are predictable since they are of course determined *via* mathematically logical methods, and will as a result exhibit a specific, repeatable pattern. These sequences are based upon initial seed values; the seed determines the numbers that will follow within the sequence by defining the internal state of the algorithm. Given knowledge then of the algorithm used to create the numbers and its internal state (i.e. seed), one could then predict all the numbers returned by subsequent calls to the algorithm, whereas with genuinely truly random numbers, knowledge of one number or an arbitrarily long sequence of numbers is of no use whatsoever in predicting the next number to be generated.

These are often confused with the concept of quasi-random numbers; in fact the terms are periodically interchanged as being the same. However, they dramatically different but are related. Most native random number routines are quasi-random. Quasi-Random numbers can only be considered to be random over a limited number of draws, as given a large number of draws the numbers generated from the sequence will form a Gaussian distribution. Extending this to a two dimensional grid, will result in discernable regular patterns as can be seen in figure 4(C) for large numbers of draws. Figure 4's data was generated using the Sobol Technique, a known generator for quasi-random numbers [30].

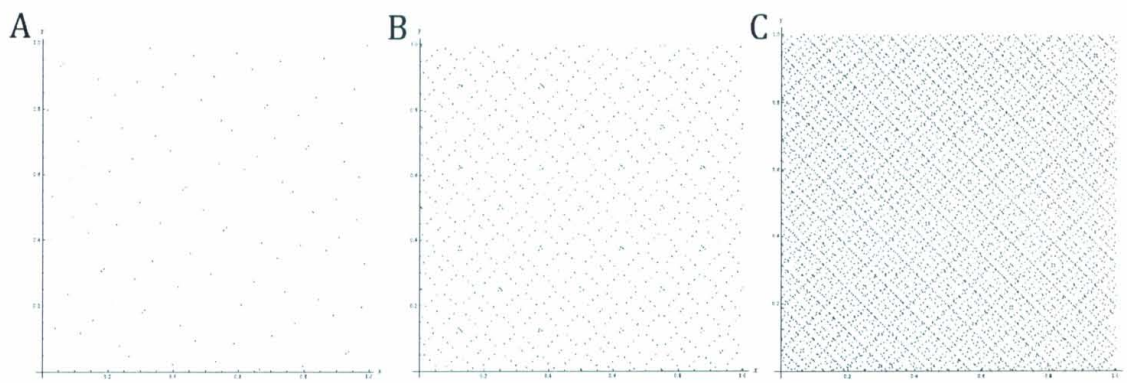


Figure A.0.1: Quasi-Random Number Generator randomizing coordinates for A) 100 Draws B) 1000 Draws C) 6000 Draws, using the Sobol Technique for random number generation.

Quasi-Random number generators then fail the conditions for random numbers after a fairly limited number of draws. They become predictable and thus can be considered to have a very short periodicity, i.e. the period in which that the numbers become less random and predictable occurs over a very limited number of draws. This of course is fine for instances

where the program uses a low number of random numbers. However if large quantities of random numbers are required, such as in Monte-Carlo based simulations or algorithms, then quasi-random number generators do not work.

Most if not all higher level computer languages contain a random function for the generation of random numbers. If the language is capable of mathematical operations then it more than likely has one. These native random functions are included for small scale testing and for low quantities of random numbers, they are quasi-random. This is done to minimize the size (memory wise) of the math libraries associated with each language. Lower level languages do not contain randomization techniques, as they are not typically used in situations where random number generation is required.

It therefore becomes necessary to design a better random number generator if it is to be used to perform any of the Monte-Carlo techniques.

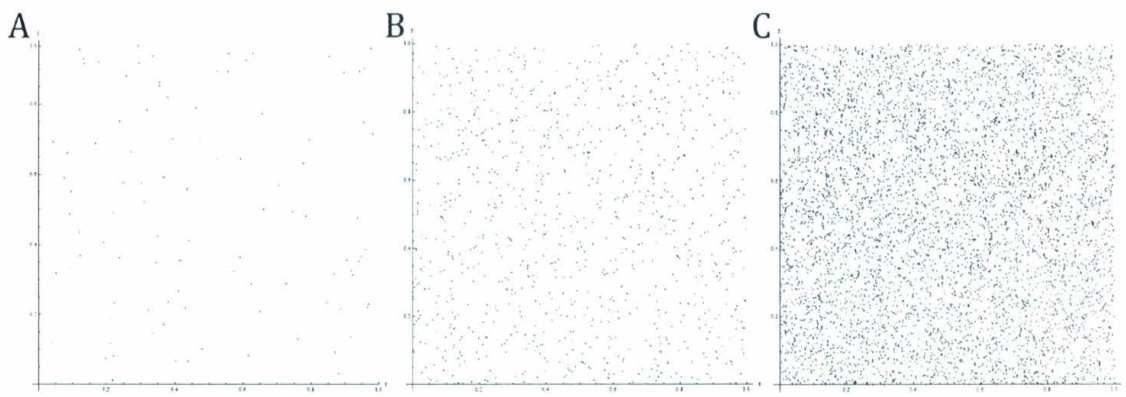


Figure A.0.2: Pseudo-Random Number Generator randomizing coordinates for A) 100 Draws B) 1000 Draws C) 6000 Draws, using the Mersenne Twister Technique for random number generation.

One of these better techniques for the generation for long periodicity high quality random numbers is the Mersenne Twister technique. The results of this technique can be seen in Figure A.0.2. A quick comparison of part C between Figure A.0.1 and Figure A.0.2 reveals that while using the Sobol technique for low numbers of random numbers is fine, it forms patterns the more the numbers are drawn, unlike Sobol, Mersenne Twisters to do have this issue and have very long periodicities, as discussed in appendix B. The Mersenne Twister algorithm was selected for this work. However other pseudorandom number generators also exist.

A few examples of high quality pseudorandom number generators are RANLUX (luxury random numbers)[31] , CMRG (combined multiple recursive generator) [32], and the Tausworthe generator[33].[34] More generators can be found in the GNU Scientific Library

Manual.[34] The selection of the random number generator depends upon the nature of the simulation itself.

Appendix B: Random Number Generator Mersenne Twister

First described in a paper by Makoto Matsumoto and Takuji Nishimura, the Mersenne Twister Pseudo-Random number generator has a periodicity of $2^N - 1$ [35]. Periodicity refers to the number of numbers that can be generated before the series starts to lose randomness. In this work N = 19937, so that corresponds to a periodicity of 4.3154×10^{6001} numbers that can be drawn. Since this original paper multiple versions of the Mersenne Twister have been released with N values ranging from 521 up to 216091. (See Table 3) For a sense of scale of how impressive this pseudo random number generator is, the lagged Fibonacci series, considered to be a very good random number generator, has a period of $(2^{24} - 1) \times 2^{94} \approx 2^{110} \approx 1.298 \times 10^{33}$ Draws.

N	Number of Draws before loss of randomness
4253	1.9080×10^{1280}
11213	2.8141×10^{3375}
19937	4.3154×10^{6001}
44497	8.5451×10^{13394}
86243	5.3693×10^{25961}

Table 3: Mersenne Twister Periodicity for Common Variations. The highlighted variation was the variation used in this numerical simulation.

The values of N are known as the Mersenne Primes, these are defined by

$$P = 2^N - 1. \tag{0.1}$$

If P is prime and N is also prime then, N is said to be a Mersenne prime.

The algorithm is based upon the linear recurrence relationship

$$x_{k+n} := x_{k+m} \oplus \left(x_k^u \middle| x_{k+1}^l\right) A \quad (k = 0,1, \dots), \tag{0.2}$$

here x is a word vector, and A is matrix. A word vector is a w-dimensional row vector over the two-element field $\mathbb{F}_2 = \{0,1\}$ [35]. Here n is the integer degree of recurrence, m is an integer in the range $1 \leq m \leq n$. Where A is given by:

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{0} & & & & \\ \mathbf{0} & & \mathbf{I}_{\text{w-1}} & & \\ \vdots & & & & \\ \mathbf{0} & & & & \\ a_{w-1} & a_{w-2} & \cdots & \cdots & a_0 \end{bmatrix}. \tag{0.3}$$

The algorithm generates a series of these word vectors, which can be considered to be a uniform pseudorandom number between 0 and $2^w - 1$, division by $2^w - 1$ results in a real number in the range [0, 1][35].

Appendix C: Data Structures

Lower level computer languages, those that offer little to no abstraction from a processor's instruction set architecture, have no means to provide a quick reference to a collection of data in memory. Here level refers to the amount of abstraction. All data that is stored in memory in this case is referenced by the memory address in which the data is stored. This makes writing complex or long programs very difficult.

In order to reduce this difficulty in creation of complex and long programs the higher level languages were developed. These languages provide a greater amount of abstraction from the instruction set architecture of the processor. It was in the creation of these languages that a computational methodology was developed, now commonly referred to as a data structure, in order to organize and sort data so that is can be accessed as one common reference. The data does not need to be of any particular type within the structure, so any given structure can contain any data type, including other data structures. This offers a few obvious benefits, movement of vast quantities of data through the use of only one variable, and organizing data associated with a particular concept are a pair of examples.

The simplest form of a data structure that can be thought of is an array, it is a special type of data structure. The array is known as a homogeneous data structure as all of the data contained within the array must be of the same type.

Consider one of the data structures that is located in the "Structures.h" file located in Appendix E, transcribed below:

```
typedef struct{
        unsigned long int AtomNumber;          //Atom Number
        Point Coords;                          //Location in Real Space
        char AtomSB;                           //Surface VS Bulk
        char AtomType;                         //Si VS Au
        bool AtomBound;                        //Atom In Boundary?
        bool First;                            //First in the row?
        bool AtomVirtual;                      //Is the Atom Real, Virtual Atoms DNE
}Atom;
```

Every piece of information that defines an atom within the simulation is contained within the structure. This then is a heterogeneous data structure since it does not contain just one type of data. It contains three primitive data types and another structure (Point - contains the X/Y coordinates of the atom). Building upon this and expand the concept further. Certain languages, such as java, are based largely on this concept. A great portion of the coding done in java is based upon the creation and manipulation of data structures or as they are referred to in java 'objects'.

It is now possible to create an array of atoms, since Atom is now considered to be a valid data type. This hierarchy can be seen in Figure A.0.3.

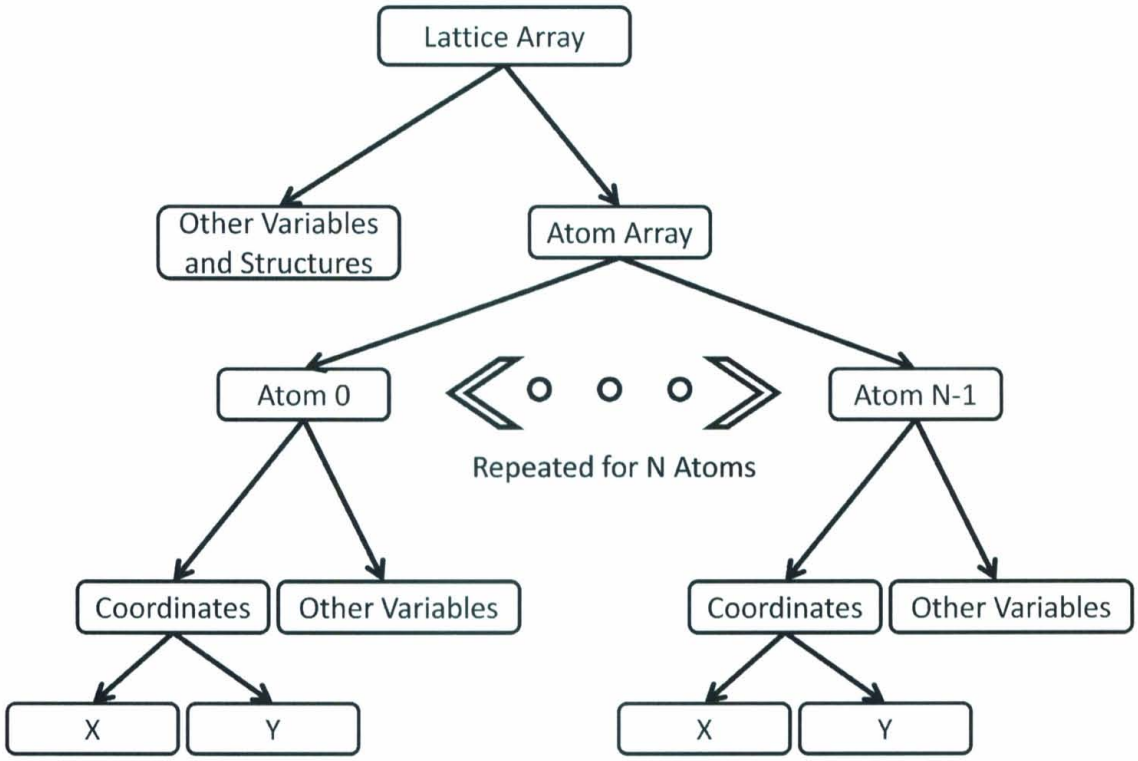


Figure A.0.3: Lattice Array Data Structure

Using data structures in programming, allows vast amounts of information to be routed around very quickly and efficiently.

Referring to Figure A.0.3, one can see how data is transferred around the main analysis program. The pointer to the LatticeArray variable is passed around the program, not the entire array itself. Everything about the lattice is connected to this one variable, except for the array of springs in the lattice, this makes moving data around easy. The spring array is not included to save space, since the simulated annealing algorithm requires three copies of the

data to work, it is not updated beyond the initial creation, and therefore it is made into a global variable so that it can be accessed by any function at any time.

The majority of the code in the appendices is coded in C99, i.e. the version of the C language that was released in 1999. Therefore it becomes a necessity to understand code wise how C99 implements data structures. For instance if the LatticeArray variable CurrentArray is passed into a sub function, and the algorithm is required to access the X coordinate of the i^{th} atom then the code segment used to access the information becomes the following: CurrentArray->AtomArray[i-1].Coords.x . Here "->" is a reference to a passed pointer, CurrentArray is passed to the function, this pointer can be considered to be an external pointer, as it does not exist within the structure, it is the structure. However, if the structure has not been passed to a sub-function it is actually an internal pointer denoted by ".", so here Coords is a local variable to AtomArray hence it is denoted with "." . This also applies to anywhere where the structure is local. For instance in order to access the same X coordinate in a local scope, the code segment would be: CurrentArray.AtomArray[i-1].Coords.x . Therefore "->" indicates a passed not local scope data structure, whereas "." indicates a local scope data structure; everything inside the structure itself is local.

The reason why data structures are used, beyond the simplification of routing information around the simulation, is that it keeps most of the data together and contiguous in memory. Looking back at Figure A.0.3, the other data and structures may or may not be contiguous with the atom array, since the arrays are of variable sizes they are allocated after runtime. Most of the data however is stored within the atomarray structure, which is

contiguous within memory. This is important once the memory/processor architecture of a typical multi-core computer system is considered. Having the data in a contiguous structure means that it will typically spent more time in the faster portions of memory and less time in the dramatically slower regions of memory. It is desirable to have the data in this lower memory as the access times are orders of magnitude smaller as compared to the upper level memory. Refer to Figure A.0.4, for access times as well as storage capacity of the different levels of memory.

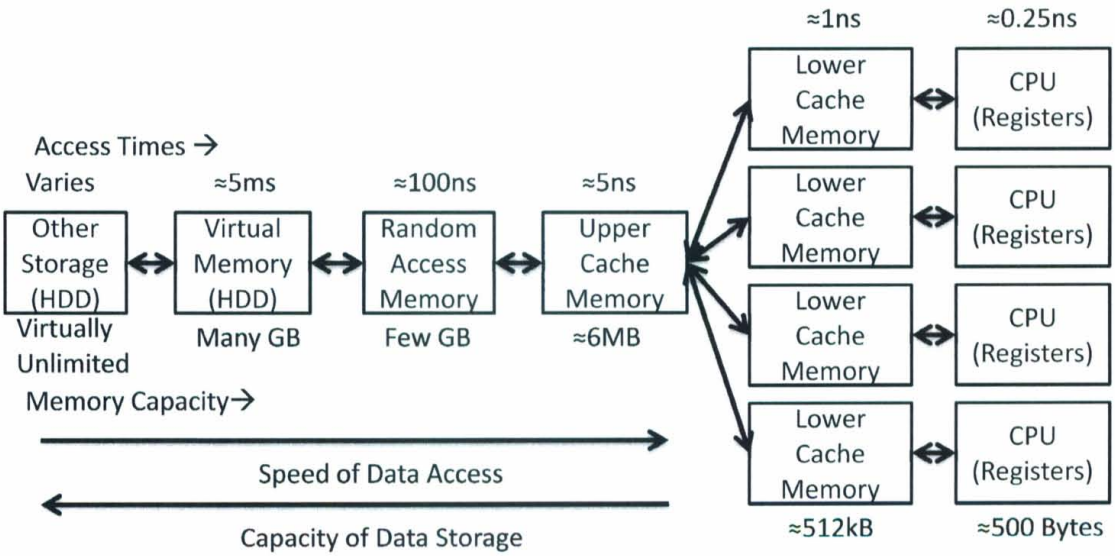


Figure A.0.4: Memory hierarchy of a typical Advanced Micro Device (AMD) quad core computer system, Intel/Sun/Apple/Others have very similar based systems

Appendix D: OpenMP Parallel Programming

The OpenMP API (Open Multi – Processing Application Programming Interface) was introduced in 1997 by a consortium of major computer hardware and software companies to serve as a common base for multi-core shared memory processing. This group of companies desired to provide a standard that could be utilized among a variety of shared memory architectures/platforms. This gave programmers the ability to utilize the power of multi-core shared memory processing without having to expressly program every parallel step. However, it is not automatic parallelization, but it is instead semi-automatic based upon the inclusion of complier pre-processing instructions. This API then allows a programmer to explicitly direct multi-threaded, shared memory parallelism within a program.[36]

This API is written in a couple of languages, notably C/C++ and FORTRAN, but is usable in a multitude of other languages as well.

The benefit of using the OpenMP API is the ease of which a program can be transformed from being executed in a serial nature into one that is executed in a combination serial and parallel nature, this is done through the use of compiler pre-compiler instructions. In C and Fortran these are known as Pragmas. The `#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself. These directives allow for programmers to explicitly dictate which sections of the code are to be executed in parallel. This creates a fork in the program execution, i.e. when the program reaches a section of program that can be executed in parallel it splits the

execution thread into multiple threads that are distributed over a number of cores (see Figure A.0.5) and once the parallel section has finished the multiple threads are then merged back into the execution thread[37][38].

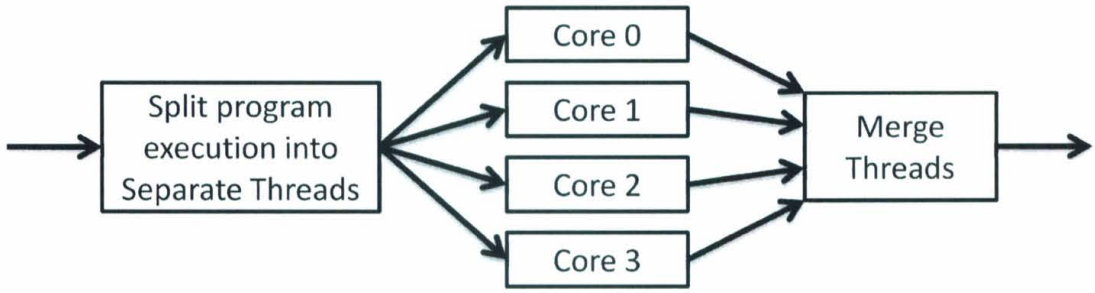


Figure A.0.5: OpenMP Multi-Threading

There is no restriction on the number of threads that can be created with the execution of a program, however the number of currently running threads is limited to the number of physical processor threads. This means that if you create 8 threads on a program and execute it on a processor that has only 4 physical threads then only 4 will be executed at a time, the other 4 will wait until the rest have finished.

OpenMP Parallel Commands

The OpenMP API defines an assortment of various commands for running calculations in parallel. Referring to "Energy.c " in Appendix C, reveals how OpenMP is implemented within the simulation. The OpenMP library is only included if the correct compiler option is included, as it is not required to be active when running small test systems this is done through the use of the ifdef (if defined) pre-compiler option. This section of code below indicates to the compiler to

check if OpenMP has been defined in the compiler options, if so then include the library that makes OpenMP work.

```
#ifdef _OPENMP
#include <omp.h>
#endif
```

Once these libraries are included the program can then be compiled to execute in parallel, these libraries contain the instructions for the compiler on how to change the following code into a parallized code. A simple example of this is the conversion of a for loop into a parallel structure.

```
#pragma omp for
 for(int n=0; n<10; n++)
 {
   printf(" %d ", n);
 }
```

The pragma tells the compiler to follow the rules specified by the OpenMP directives (omp) for the conversion of the for loop. In this case the loop will simply print the numbers from 0 to 9. Once converted the code executed by each individual thread and will be converted into the section below.

```
int this_thread = omp_get_thread_num();

int num_threads = omp_get_num_threads();

int start = (this_thread  ) * 10 / num_threads;

int end   = (this_thread+1) * 10 / num_threads;

for(int n=start; n<end; ++n)

   printf(" %d ", n);
```

Here the first four lines assign the loop iterations to each of the threads; each thread has a number associated with it. Therefore the result from running this code in parallel on two cores may be: 0 5 1 6 2 7 3 8 4 9. Each thread gets a different section of the loop, and they execute their own sections in parallel. [39] The code used in "Energy.c " is similar, but has a few additions as can be seen below.

```
#pragma omp parallel for shared(Array,SpringArray) private(i)
```

Here "parallel" is a short notation for the creation of a new set of threads, "shared ()" is used to specify any data structures that are to be shared among the threads, any thread can access and modify the data contained within the structure. "private()" is the opposite of "shared()", the variable "i" is unique to each thread and cannot be modified by another thread, and is often used to specify the iteration variable.

The purpose of the algorithm specified within the Energy.c file is to calculate the total lattice energy of the system. However, once this is moved into the parallel regime it is no longer a safe operation to add the energies for each spring to determine a total energy. This operation is not thread-safe i.e. it does not prevent two or more threads from trying to access and write to the same location in memory at the same. This can happen here if two or more threads attempt to modify the total energy variable at the same time. This is obviously a problem, as it is indeterminate what value will be written to memory, it may be the value from one of the threads, and it could also be a randomized indeterminate value.

To prevent this situation OpenMP includes a pragma command called atomic, this command prevents threads from writing to the same location in memory at the same time, but will in fact let only one thread update the value at a time, and will queue the rest until each has updated the value. This works only on the combinatorial operators, i.e. ++, --, +=, -=,*=, and /=. The atomic code seen below must be included with every instance of the medication of the Energy variable, as it only applies to the next line in the code.

```
#pragma omp atomic
Energy += Expression
```

There are a variety of other pragmas associated with the OpenMP API however they are not required for the calculations in this work.

Appendix E: Simulated Annealing Source Code (C99)

Files are listed in alphabetical order. Not included are the standard numerical recipes library files, as well as the mersenne twister algorithm. The mersenne twister algorithm is available online and can be found at http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/. All other files in this program unless otherwise stated are ©Victor C. Hayden.

BoundFit.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include "Common.h"
/*****************************************************************************
*       This is the linear regression algorithm found in Numerical Recipes in C, it
*       has been updated to work with the LatticeArray Data Structures.
*       Reference NR 15.2 Fitting Data to a Straight Line, these are unweighted points.
*****************************************************************************/
PointSlopeLine fit(LatticeArray *Array, Bound Boundary)
{
        double t,sxoss,sx=0.0,sy=0.0,st2=0.0,ss;
        PointSlopeLine line;
        line.Intercept=0.0;
        line.Slope = 0.0;

        for (int i=0; i < Boundary.NumAtomsInBound; i++) {
                sx += Array->AtomArray[Boundary.AtomsInBound[i]].Coords.x;
                sy += Array->AtomArray[Boundary.AtomsInBound[i]].Coords.y;
        }
        ss=Boundary.NumAtomsInBound;
        sxoss=sx/ss;

        for (int i=0; i < Boundary.NumAtomsInBound; i++) {
                t=Array->AtomArray[Boundary.AtomsInBound[i]].Coords.x-sxoss;
                st2 += t*t;
                line.Slope += t*Array->AtomArray[Boundary.AtomsInBound[i]].Coords.y;
        }

        line.Slope /= st2;
        line.Intercept=(sy-sx*(line.Slope))/ss;

        return line;
}
```

BoundLines.c

```
#include <stdio.h>
#include <stdlib.h>
#include "Common.h"
/*****************************************************************************
*       Creates both boundary lines for the LatticeArray Array and stores them in
*       BoundLines Boundary
*       Coded By: ©Victor C. Hayden (FLAG //GR645)
*****************************************************************************/
void CreateBound(LatticeArray *Array, BoundLines *Boundary){
        Boundary->RLine = fit(Array, Array->RBound);
        Boundary->LLine = fit(Array, Array->LBound);
}
```

BoundRotate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Common.h"
/*****************************************************************************
*       This function is responsible for the Boundary rotations, it uses the
*       boundarys located in BoundLines Boundary to do the rotations. This will
*       work irregardless of how the lines are defined, they have been defined
*       here to be between the fit of the first atom of every row
*       Coded By: ©Victor C. Hayden (FLAG //GR645)
*****************************************************************************/
Point BoundAR(LatticeArray *Array, char B,
                BoundLines *Boundary, double A1X, double A1Y){

        Point tmp;
        double x, y, x_tmp, y_tmp, Theta, Sep;

        //Small Slopes only need to be translated, not rotated
        if( Boundary->LLine.Slope - Boundary->RLine.Slope < 1.0e-9){
                Sep = fabs(Boundary->LLine.Intercept - Boundary->RLine.Intercept);
```

```
            if( B == 'R' || B == 'r' ){
                tmp.x = A1X;
                tmp.y = A1Y - Sep;
            } else {
                tmp.x = A1X;
                tmp.y = A1Y + Sep;
            }

            return tmp;

        } else {
            //Determine the Intersection Point of the two boundaries at this point
            //they should not be parallel
            x = (Boundary->LLine.Intercept - Boundary->RLine.Intercept)/
                (Boundary->RLine.Slope - Boundary->LLine.Slope);
            y = Boundary->RLine.Slope * x + Boundary->RLine.Intercept;
            //Adjust the Coordinate system to be at this intersection point to use
            //the Standard rotation matrix
            x_tmp = A1X - x;
            y_tmp = A1Y - y;
            //Calculate the angle between the lines
            Theta = fabs(atan((Boundary->RLine.Slope - Boundary->LLine.Slope) /
                                (1.0+ (Boundary->RLine.Slope * Boundary->LLine.Slope))));
            //Select the correct value of Theta
            if( B == 'L' || B == 'l' ) Theta = -1.0 * Theta;
            //Do the rotation
            tmp.x = x_tmp * cos(Theta) - y_tmp * sin(Theta) + x;
            tmp.y = y_tmp * cos(Theta) + x_tmp * sin(Theta) + y;

            return tmp;

        }
}
```

```
/*************************************************************************
*       This program is designed to determine the resulting shape of a bi-layer
*       Gold-Silicon microcantilever once it has been exposed to an induced
*       surface stress. The resultant shape of the cantilever is due to the
*       minimization of the Lattice Energy of the system.
*
*       This program uses a Metropolis-Hastings Algorithm called simulated
*       annealing to do this. There is more on this in the SimAnn.c file.
*       This File serves as the main controller for running the simulated
*       annealing algorithm for various amount of surface stretch, and handles
*       reading in the data, and creating the outputs.
*
*       There is also a short tutorial on Datastuctures in Structures.h.
*
*       Coded By: ©Victor C. Hayden (FLAG //GR645)
*
*************************************************************************/
#include <stdio.h>  //Std input output library
#include <stdlib.h> //Std c library that contains most functions
#include <time.h>   //Time Library, to access the Current Time
#include <math.h>   //Contains standard math functions
#include <stdbool.h>//Library to use Boolean Logic, i.e. true or false
#include "Common.h" //Contains required constants
extern void FileCleanup();
/*************************************************************************
*       This function groups together those functions that are required to create
*       the LatticeArray Datastructure initially, only needs to be called once
*       for each LatticeArray that you want to make. Best if this is done in
*       pre-processing.
*       Note after the data is read in from the        csv file discribed in FileInOut.c
*       that it is rotated by 90 degrees. Since the program will be finding
*       boundary lines to rotate it is best to avoid pure vertical lines. While
*       the program will happily put the inf into the array, as it is allowed.
*       It is best to avoid it, as it will cause the program to crash in places.
*************************************************************************/
```

```
void LatticeCreator(LatticeArray *Array, char FileName[],
                                                        int argCommand,
char *argv[]){
        ArrayInit(Array);                       //Array Initializer
    inputFileToArray(Array,FileName, argCommand,argv);//Reads in the file
    createSpringLattice(Array);                 //Makes the Array of springs
    MinLineInit(Array);                         //Makes the 2D array for the LineMover
    RotateEntireCantilever(Array, -M_PI_2);         //Rotates by rad degrees
    SurfaceSpringAssignment(Array,Array->NumberSurfaceAtoms);
    if (recovery) {PowerCrashRecover(Array);}
}

/*****************************************************************************
*       Steps here are explained as they appear, except where the same code is
*       repeated
*
*       WARNING : THIS FUNCTION CONTAINS AN INFINITE LOOP
*                       This Loop depends on a break in the if statement and is connected
*                       to CurStretch and AmtStretch, beware of modifications to these
*                       varibles.
*
*       argv[] accesses the commandline.
*       atof() converts the commandline string to double
*
*****************************************************************************/
int main(int argc, char * argv[]){
        //LatticeArrays for Simulated Annealing
        LatticeArray F_Array, CurrentArray, SpaceArray, BestArray;
        BoundLines BoundOut;        //Rotated version of the final boundary lines
        double CurStretch = 0.0;//Current Percent Stretch
        double NumStretch = 0.0;//Number of stretches to do
        double AmtStretch = 0.0;//Amount to Stretch each time
        //Output file name, gets altered to append current amount of stretch
        // i.e Lat10_10_4_ with 1% stretch becomes Lat10_10_4_1_000.csv
        char *FileNameString = NULL;
        char *name;   //Output file name initial string, i.e. ./Lat10_10_4_
        char *Oname; //VizOutput file name initial string, i.e. ./Lat10_10_4_
```

```
    double surfBonds = 0.0; //Surface Bond Length

    //atof() converts from string to double
    k_Surface = atof(argv[4]); //Surface Spring Constant
    SBondStretch = atof(argv[5]);//Total Percent Surface Stretch
    NumStretch = atof(argv[6]);

    //Read in the data from the input file
LatticeCreator(&F_Array, argv[1], argc,argv);
    LatticeCreator(&CurrentArray, argv[1], argc,argv);
    LatticeCreator(&SpaceArray, argv[1], argc,argv);
    LatticeCreator(&BestArray, argv[1], argc,argv);

    printf("Number of Atoms in the Array: %i    ****\n", F_Array.ArrayIndex);

    name = argv[2];
    Oname = argv[3];

    AmtStretch = SBondStretch / NumStretch;

    //Handling for both LatticeArrays with and without Gold
    surfBonds =  (F_Array.AUinX == 0|| BestArray.AUinX == 0 ||
                              SpaceArray.AUinX == 0 || CurrentArray.AUinX == 0)?
                             (SiBL):(AU_L);

    //******//WARNING UNRESTRAINED LOOP//******//
    for(;;){

            if (recovery){
                    CurStretch = CPCurStr;
            }else{
                    CurStretch += AmtStretch;
                    CPCurStr = CurStretch;
            }
            if ((double) CurStretch >= SBondStretch ){
                    break; //DO NOT REMOVE
            }
```

```
        //Stretching the surface bond lengths
        F_Array.SurfaceBLength = surfBonds*(1.0 + (CurStretch / 100.0));
        CurrentArray.SurfaceBLength = F_Array.SurfaceBLength;
        SpaceArray.SurfaceBLength = F_Array.SurfaceBLength;
        BestArray.SurfaceBLength = F_Array.SurfaceBLength;

        printf("STEP ********** -> %lf\n", CurStretch); //Where are we?
        //Running SA, May take a long time
    RunSimAnn(&F_Array, &BestArray, &CurrentArray, &SpaceArray,argc,argv);

        CreateBound(&F_Array, &BoundOut); //Rotated boundarys for output
        //Rotate Cantilever back to normal horizontal config for output
        RotateEntireCantilever(&F_Array, M_PI_2);
        //Create the output filename
        FileNameString = FileName(name,CurStretch);
        //Outputs the LatticeArray to the csv file
        arrayToOutputFile (&F_Array, &BoundOut, FileNameString);
        FileNameString = NULL;
        //Rotate Cantilever back to vertical config for further calculations
        RotateEntireCantilever(&F_Array, -M_PI_2);

}
//******//WARNING UNRESTRAINED LOOP//******//

//Evaluating last amount of Stretch
F_Array.SurfaceBLength = surfBonds*(1.0 + (SBondStretch / 100.0));
CurrentArray.SurfaceBLength = F_Array.SurfaceBLength;
SpaceArray.SurfaceBLength = F_Array.SurfaceBLength;
BestArray.SurfaceBLength = F_Array.SurfaceBLength;
printf("STEP ********** -> %lf\n", (double)SBondStretch);
//Running SA, May take a long time
RunSimAnn(&F_Array, &BestArray, &CurrentArray, &SpaceArray,argc,argv);
//Writing final atomic positions and boundary lines
//Create final Rotated boundarys for output
CreateBound(&F_Array, &BoundOut);
//Rotate Cantilever back to normal horizontal config for output
```

```
        RotateEntireCantilever(&F_Array, M_PI_2);
        //Create the output filename
        FileNameString = FileName(name,(double)SBondStretch);
        //Outputs the LatticeArray to the csv file
        arrayToOutputFile (&F_Array, &BoundOut, FileNameString);
        //Clean-Up F_Array, CurrentArray, SpaceArray, BestArray
        free(FileNameString);
        free(F_Array.AtomArray);
        free(CurrentArray.AtomArray);
        free(SpaceArray.AtomArray);
        free(BestArray.AtomArray);
        free(SpringArray);
        FileCleanup();
        return 0;
}
```

Common.c

```
#include "Common.h"
#include <stdbool.h>
//Global Variables
double k_Surface;           //Surface Spring Constant
int NumberSurface;
double SBondStretch;            //Total Percent Surface Stretch
bool recovery;              //Is this execution recovering data
double CPCurStr;            //Crash Protection Current Stretch
double CPTem;                   //Crash Protection Temperature
Spring *SpringArray;        //Array of Springs
bool Spr;
```

Common.h

```
#include "Structures.h"
#ifndef _COMMON_H_
#define _COMMON_H_
//Global Varibles
extern double k_Surface;
extern int NumberSurface;
extern double SBondStretch;
extern bool recovery;
extern double CPCurStr;
extern double CPTem;
extern bool Spr;
extern Spring *SpringArray;
//Globally Availible Functions
extern PointSlopeLine fit(LatticeArray *Array, Bound Boundary);
extern void CreateBound(LatticeArray *Array, BoundLines *Boundary);
extern Point BoundAR(LatticeArray *Array, char B, BoundLines *Boundary,
                     double A1X, double A1Y);
extern double DetermineEnergy (LatticeArray *Array);
extern void inputFileToArray (LatticeArray *, char FileName[],
                              int argCommand, char * argv[]);
extern void arrayToOutputFile (LatticeArray *,
                              BoundLines *Boundary, char FileName[]);
extern void printDerivArray (LatticeArray Array,
                              Point *DArray, char FileName[]);
extern void LatticeArrayCopier(LatticeArray *, LatticeArray *);
extern void LatticeArrayUpdater(LatticeArray *, LatticeArray *);
extern void RotateEntireCantilever(LatticeArray *, double Theta);
extern void createSpringLattice (LatticeArray *);
extern void VizOutput(LatticeArray , char InFile[], char FileName[],
                     int argc, char *argv[]);
extern void freeLatticeArray(LatticeArray );
extern void RunSimAnn (LatticeArray *Array, LatticeArray *BestArray,
                              LatticeArray *CurrentArray,
                              LatticeArray *SpaceArray, int argCommand,
                              char * argv[]);
```

```
extern void ArrayInit(LatticeArray *);
extern void MinLineInit(LatticeArray *);
extern void GoldLineInt(LatticeArray *);
extern char* FileName (char*, double);
extern void SurfaceIni(LatticeArray *);
extern void LatticeArrayCopier(LatticeArray *, LatticeArray *);
extern void LatticeArrayLine(LatticeArray *, LatticeArray *);
extern void PowerCrashRecover(LatticeArray *);
extern void CrashPowerProtect(LatticeArray *, int argCommand, char * argv[]);
extern void TUpdater();
extern bool RejectAcceptSurfaceLength(LatticeArray *);
extern void SurfaceSpringAssignment(LatticeArray *,int);
#endif

//Global Constants
#define M_PI_2 1.57079632679489661923  /* pi/2 */


//These are the natural spring lengths of the system in angstroms:
#define GG_L 2.884995        //Au-Au Nearest Neighbour Bond Length
//Au-Au Next Nearest Neighbour Bond Length ~ 4.9969
#define AU_L sqrt(2 * pow(2.04, 2) + pow(4.08, 2))
#define SS_L 3.840311        //Si-Si Nearest Neighbour Bond Length
#define SiBL 5.43       //Si-Si Next Nearest Neighbour Bond Length
#define GS_L 2.439575        //Au-Si Nearest Neighbour Bond Length
#define AUSIBL 3.43   //Au-Si Next Nearest Neighbour Bond Length
#define VSep 1.1882  //Vertical Separation between the Au and Si interface
#define k_GG 0.30608         //Gold-Gold Nearest Neighbour Spring Constant
#define k_GGs 0.21643        //Gold-Gold Next Nearest Neighbour Spring Constant
#define k_GS 0.14600         //Gold-Silicon Nearest Neighbour Spring Constant
#define k_GSs 0.10323        //Gold-Silicon Next Nearest Neighbour Spring Constant
#define k_SS 0.28584         //Silicon-Silicon Nearest Neighbour Spring Constant
//Silicon-Silicon Next Nearest Neighbour Spring Constant
#define k_SSs 0.20211
```

Energy.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include <string.h>
#include "Common.h"
//      Coded By: ©Victor C. Hayden (FLAG //GR645)
#ifdef _OPENMP
#include <omp.h>
#endif
/*****************************************************************************
*       This function determines the energy contained within the current
*       LatticeArray Array, it includes the boundary rotations at the end for
*       periodic boundary conditions.
*       The energy for the lattice is determined through the use of a slightly
*       modified Spring energy equation:
*
*       Energy = (1.0/2.0) SpringConstant (NaturalBondLength/radius)^2 *
*                                   (radius - NaturalSpringLength)^2
*
*       The additional (NaturalBondLength/radius)^2 helps the function to mimic a
*       Lennard-Jones Potential. It goes to zero for large radii and infinity
*       for very small radii.
*
*       Depending on the current spring, it selects the correct spring constant
*       for the system. There are five distinct spring types: Surface, Nearest
*       Neighbour Gold, Next Nearest Neighbour Gold, Nearest Neighbour Silicon,
*       and Next Nearest Neighbour Silicon. These are defined in the file
*       Common.h.
*****************************************************************************/

double DetermineEnergy (LatticeArray *Array){
        BoundLines Boundary;
        double Energy = 0.0;
        double k_Surf = k_Surface;
```

```
        int i =0;
        CreateBound(Array, &Boundary); //Creates the boundaries for the rotations
#pragma omp parallel for shared(Array,SpringArray) private(i)
        for(i = 0; i < (Array->NumberSprings); i++){
                double rad = 0.0;
                double A1X, A1Y, A2X, A2Y;
                Point tmp1, tmp2;

            //Here we are checking to see if something has happened to the
                        //Energy that should never happen i.e rad -> 0
                if(isnan(Energy)){
                        printf("Determine Energy has Failed with a NaN...Exiting");
                        exit(-1);} //Is Energy Not a Number(NaN)
                if(isinf(Energy)){
                        printf("Determine Energy has Failed with a inf...Exiting");
                        exit(-1);} //Is Energy Infinity (inf) c99 will do calculations
                                        //with Energy = inf, but they will not be correct
                rad = sqrt(pow((Array->AtomArray[SpringArray[i].Atom1].Coords.x -
                                        Array->AtomArray[SpringArray[i].Atom2].Coords.x),2.0) +
                                        pow((Array->AtomArray[ SpringArray[i].Atom1].Coords.y -
                                        Array->AtomArray[SpringArray[i].Atom2].Coords.y),2.0));
                if(isinf(rad)){exit(-1);} //We do not want radius to be infinite
                //Here the Current Spring has a gold atom on each of its ends
                if ((Array->AtomArray[SpringArray[i].Atom1].AtomType == 'G')&&
                                (Array->AtomArray[SpringArray[i].Atom2].AtomType == 'G')){
                        if((Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')&&
                                (Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
                                #pragma omp atomic
                                Energy += (1.0 / 2.0) * k_Surf * pow((AU_L / rad),2.0) *
                                        pow((rad - Array->SurfaceBLength),2.0); //Surface Energy
                        }else{
                                if (SpringArray[i].NNN ){
                                        #pragma omp atomic
                                        Energy += (1.0 / 2.0) * k_GGs * pow((AU_L / rad),2.0) *
                                        pow((rad - SpringArray[i].Length),2.0);
                                        //Next Nearest Neighbour Energy
                                }
```

```
                else {
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_GG * pow((GG_L / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Nearest Neighbour Energy
                }
        }
}
//Here the Current Spring has a gold atom on one end and a silicon
//one on the other
else if ( Array->AtomArray[SpringArray[i].Atom1].AtomType !=
                 Array->AtomArray[SpringArray[i].Atom2].AtomType){
        if((Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')&&
                (Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
                #pragma omp atomic
                Energy += (1.0 / 2.0) * k_Surf * pow((AUSIBL / rad),2.0) *
                                        pow((rad - Array->SurfaceBLength),2.0);
                                        //Surface Energy
        }else{
                if (SpringArray[i].NNN ){
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_GSs * pow((AUSIBL / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Next Nearest Neighbour Energy
                }
                else {
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_GS * pow((GS_L / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Nearest Neighbour Energy
                }
        }
}//Here the Current Spring has a silicon atom on each of its ends
else if ((Array->AtomArray[SpringArray[i].Atom1].AtomType == 'S')&&
                (Array->AtomArray[SpringArray[i].Atom2].AtomType == 'S')){
        if((Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')&&
                (Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
```

```
                #pragma omp atomic
                Energy += (1.0 / 2.0) * k_Surf * pow((SiBL / rad),2.0) *
                        pow((rad - Array->SurfaceBLength),2.0); //Surface Energy
        }else{
                if (SpringArray[i].NNN ){
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_SSs * pow((SiBL / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Next Nearest Neighbour Energy
                }
                else {
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_SS * pow((SS_L / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Nearest Neighbour Energy
                }
        }
    }
    else {
        printf("DetermineEnergy has failed due to
                unknown Atom type in Lattice!"); exit(-1);
    }
//Here we check to see if we need to do a rotation for the current spring
    if(((Array->AtomArray[SpringArray[i].Atom1].AtomBound == true)&&
        (Array->AtomArray[SpringArray[i].Atom2].AtomBound == false))||
        ((Array->AtomArray[SpringArray[i].Atom1].AtomBound == false)&&
        (Array->AtomArray[SpringArray[i].Atom2].AtomBound == true))){

        A1X = Array->AtomArray[SpringArray[i].Atom1].Coords.x;
        A1Y = Array->AtomArray[SpringArray[i].Atom1].Coords.y;
        A2X = Array->AtomArray[SpringArray[i].Atom2].Coords.x;
        A2Y = Array->AtomArray[SpringArray[i].Atom2].Coords.y;
        //Determine the new rotated end coordinates of the spring
        if( Array->AtomArray[SpringArray[i].Atom1].Coords.y <
                (Array->NAinX * SS_L / 2.0)){
```

```
        tmp1 = BoundAR(Array, 'L', &Boundary, A1X, A1Y);

        tmp2 = BoundAR(Array, 'L', &Boundary, A2X, A2Y);
}
else{

        tmp1 = BoundAR(Array, 'R', &Boundary, A1X, A1Y);

        tmp2 = BoundAR(Array, 'R', &Boundary, A2X, A2Y);
}

rad = sqrt(pow((tmp2.x -tmp1.x),2.0) + pow((tmp2.y -tmp1.y),2.0));

if(isinf(rad)){exit(-1);} //We do not want radius to be infinite
//Here the Current Spring has a gold atom on each of its ends
if ((Array->AtomArray[SpringArray[i].Atom1].AtomType == 'G')&&
        (Array->AtomArray[SpringArray[i].Atom2].AtomType == 'G')){
        if((Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')&&
        (Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
                #pragma omp atomic
                Energy += (1.0 / 2.0) * k_Surf * pow((AU_L / rad),2.0) *
                pow((rad - Array->SurfaceBLength),2.0); //Surface Energy
        }else{
                if (SpringArray[i].NNN ){
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_GGs *
                        pow((AU_L / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Next Nearest Neighbour Energy
                }
                else {
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_GG *
                        pow((GG_L / rad),2.0) *
                        pow((rad - SpringArray[i].Length),2.0);
                        //Nearest Neighbour Energy
                }
```

```
                    }
                }
//Here the Current Spring has a gold atom on one end and a silicon
//one on the other
                else if ( Array->AtomArray[SpringArray[i].Atom1].AtomType !=
                            Array->AtomArray[SpringArray[i].Atom2].AtomType){
                    if((Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')&&
                        (Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_Surf * pow((AUSIBL / rad),2.0) *
                        pow((rad - Array->SurfaceBLength),2.0); //Surface Energy
                    }else{
                        if (SpringArray[i].NNN ){
                            #pragma omp atomic
                            Energy+=(1.0 / 2.0) * k_GSs *
                            pow((AUSIBL / rad),2.0)*
                            pow((rad - SpringArray[i].Length),2.0);
                            //Next Nearest Neighbour Energy
                        }
                        else {
                            #pragma omp atomic
                            Energy += (1.0 / 2.0) * k_GS * pow((GS_L / rad),2.0)
                            *pow((rad - SpringArray[i].Length),2.0);
                            //Nearest Neighbour Energy
                        }
                    }
                }//Here the Current Spring has a Silicon atom on each of its ends
                else if ((Array->AtomArray[SpringArray[i].Atom1].AtomType == 'S')
                    &&(Array->AtomArray[SpringArray[i].Atom2].AtomType == 'S')){
                    if((Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')
                    &&(Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
                        #pragma omp atomic
                        Energy += (1.0 / 2.0) * k_Surf * pow((SiBL / rad),2.0) *
                        pow((rad - Array->SurfaceBLength),2.0); //Surface Energy
                    }else{
                        if (SpringArray[i].NNN ){
                            #pragma omp atomic
```

```
                                        Energy += (1.0 / 2.0) * k_SSs
                                        * pow((SiBL / rad),2.0) *
                                        pow((rad - SpringArray[i].Length),2.0);
                                        //Next Nearest Neighbour Energy
                                }
                                else {
                                        #pragma omp atomic
                                        Energy += (1.0 / 2.0) * k_SS *
                                        pow((SS_L / rad),2.0) *
                                        pow((rad - SpringArray[i].Length),2.0);
                                        //Nearest Neighbour Energy
                                }
                        }
                }
                else {
                        printf("DetermineEnergy has failed due to unknown
                                Atom type in Lattice!"); exit(-1);
                }
        }
    }
    return Energy;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include "Common.h"
#include <dirent.h> //Directory Access for remove
//      Coded By: ©Victor C. Hayden (FLAG //GR645)

/*****************************************************************************
*       This function takes the properly formatted input file and converts it to
*       an array of structures, these structures are defined in AtomInfo.h, and
*       contain all of the relevent information about the Atom in question.
*       Systems both with and without gold are handled.
*       Inputs: char FileName[] - Name of the Input File
*       Output: AtomArray - pointer to the array of structures
*                       ( i.e. the array of atoms )
*****************************************************************************/
void inputFileToArray (LatticeArray *Array, char FileName[],
                                        int argCommand,char * argv[]){
        FILE* in, *rec;
        unsigned long int AtomNum, NAinX, NAinY, ArrayIndex, LeverH, temp, AUinX;
        int scan;
        char tmp[10];
        double PosX, PosY;
        in = fopen(FileName,"r");
        if(in==NULL){printf("Error: can't open file.\n"); exit(-1);}
        if ((argCommand == 9)||(argCommand == 10)){ //Pure Silicon Lattice
                if(argCommand == 10){
                        recovery = (argv[9][0] == 't')?true:false;
                } else {
                        recovery = false;
                }
                Array->NAinX=atof(argv[7]);
                Array->NumberSurfaceAtoms=atof(argv[7]);
                Array->NAinY=atof(argv[8]);
```

```
        Array->ArrayIndex=Array->NAinX*Array->NAinY;
        Array->AtomArray = realloc(Array->AtomArray, Array->ArrayIndex *
                                                    sizeof(Atom));
        if (Array->AtomArray == NULL){
        free(Array->AtomArray);
        printf("Memory allocation failed while allocating for AtomArray.\n");
        exit(-1);}
        for(int i = 0; i<Array->ArrayIndex; i++){

        scan = fscanf(in, "%u,%lf,%lf,%s\n", &AtomNum,&PosX,&PosY,tmp);

        Array->AtomArray[i].AtomNumber = i+1;
        Array->AtomArray[i].Coords.x = PosX;
        Array->AtomArray[i].Coords.y = PosY;
        Array->AtomArray[i].AtomType = tmp[2];
        Array->AtomArray[i].AtomSB = tmp[0];
        Array->AtomArray[i].AtomVirtual = (tmp[4] == 'r')? false:true;
        }
} else if((argCommand == 11)||(argCommand == 12)){
        //Gold and Silicon Lattice
        if(argCommand == 12){
                recovery = ((argv[11][0] == 't') || (argv[11][0] == 'T'))?
                                        true:false;
        } else {
                recovery = false;
        }

        Array->NAinX=atof(argv[7]);
        Array->NAinY=atof(argv[8]);
        Array->LeverH = atof(argv[9]);
        Array->AUinX = atof(argv[10]);
        Array->NumberSurfaceAtoms=atof(argv[10]);
        Array->ArrayIndex=Array->LeverH*
                                        Array->AUinX+Array->NAinX*(Array->NAinY);
        Array->AtomArray = realloc(Array->AtomArray,
                                                Array->ArrayIndex * sizeof(Atom));
        if (Array->AtomArray == NULL) { free(Array->AtomArray);
```

```
                printf("Memory allocation failed while allocating for AtomArray.\n");
                exit(-1);}
                for(int i = 0; i<Array->ArrayIndex; i++){
                        scan = fscanf(in, "%u,%lf,%lf,%s\n", &AtomNum,&PosX,&PosY,tmp);
                        Array->AtomArray[i].AtomNumber = i+1;
                        Array->AtomArray[i].Coords.x = PosX;
                        Array->AtomArray[i].Coords.y = PosY;
                        Array->AtomArray[i].AtomType = tmp[2];
                        Array->AtomArray[i].AtomSB = tmp[0];
                        Array->AtomArray[i].AtomVirtual = (tmp[4] == 'r')? false:true;
                }



        }else{
                printf("Usage::> InputFile OutputFile VisualizationOutputFile
                SurfaceSpringConstant AmountToStretch NumberOfStretches SlinX SlinY
                [OPTIONAL:: GoldThickness AUinX, Recovery]...Exiting");
                exit(-1);
        }

        fclose(in);

}
/*****************************************************************************
*       This function takes the current Array of Atoms and writes it to a file
*       Inputs: char FileName[] - Name of the Output File
*               struct Atom *AtomArray - Array of Atoms to be written
*****************************************************************************/
void arrayToOutputFile (LatticeArray *Array, BoundLines *Boundary,
                                        char FileName[]){
        FILE *Out;
        Out = fopen(FileName, "w");
        if(Out==NULL){
                printf("Error: Can't Write to file.\n"); exit(-1);
        }
                fprintf(Out, "Atom Number, X, Y, Atom Type\n");
```

```
        for(int i = 0; i< Array->ArrayIndex; i++){
                fprintf(Out,"%u,%lf,%lf,%c \n", Array->AtomArray[i].AtomNumber,
                Array->AtomArray[i].Coords.x, Array->AtomArray[i].Coords.y,
                Array->AtomArray[i].AtomType);
        }
                fprintf(Out, "Final Boundary Line Sets: \n Right Line: Slope: %lf,
                        Intercept: %lf \n Left Line: Slope: %lf, Intercept: %lf \n",
                        Boundary->RLine.Slope, Boundary->RLine.Intercept,
                        Boundary->LLine.Slope, Boundary->LLine.Intercept);
        fclose(Out);
}

/*****************************************************************************
*       LatticeArray initializer, gives default values to the elements in
*       the LatticeArray
*****************************************************************************/

void ArrayInit(LatticeArray *Array){
        Array->NumberSprings=0;
        Array->SSpring=0;
        Array->ArrayIndex=0;
        Array->NAinX=0;
        Array->AUinX=0;
        Array->NAinY=0;
        Array->LeverH=0;
        Array->SurfaceBLength=0.0;
        Array->AtomArray=NULL;
        Array->RBound.AtomsInBound=NULL;
        Array->LBound.AtomsInBound=NULL;
        Array->Slope=NULL;
        Array->AtomsInLine=NULL;
}
/*****************************************************************************
*       This function creates a series of files that are used to Recover the
*       currently processed data in the event of a Power Failure, System Crash,
*       Program timeout (ACENET limits the amount of time a program can run
*       to about a month, this allows you to bypass this time limit), or other
```

```
*       such event. This sub-function saves the current BestArray once a new
*       one is found, thus perserving the data.
*
*****************************************************************************/
void CrashPowerProtect(LatticeArray *Array, int argCommand,char * argv[]){

        FILE *Out, *OSpr, *Consts, *Slopes, *SlopesAU, *file, *file2, *CPTS;
        Out = fopen("/globalscratch/vhayden/Sim1/CrashPowerDumpAtoms.csv", "w");
        for(int i = 0; i< Array->ArrayIndex; i++){
                fprintf(Out,"%0.14lf,%0.14lf\n", Array->AtomArray[i].Coords.x,
                                Array->AtomArray[i].Coords.y);
        }
        fclose(Out);
        /*
        if (file = fopen("./Output/CurLatticeSprings.csv", "r")){
        fclose(file);
        }
        else {
                OSpr = fopen("./Output/CurLatticeSprings.csv","w");
                for(int i = 0; i < (Array->NumberSprings); i++){
                        fprintf(OSpr,"%u,%u,%0.14lf\n", SpringArray[i].Atom1,
                                        SpringArray[i].Atom2, SpringArray[i].Length);
                }
                fclose(OSpr);
        }
        */
        if (file2 = fopen("<File Name Here>", "r")){
        fclose(file2);
        }
        else {
                Consts = fopen("<File Name Here>","w");
                for( int i = 1; i<argCommand; i++){
                        fprintf(Consts, "%s ", argv[i]);
                }
                fprintf(Consts, "\n");
                fclose(Consts);
        }
```

```
	Slopes = fopen("<File Name Here>", "w");
	for( int i = 0; i < 2*Array->NAinX; i++){
		fprintf(Slopes, "%0.14lf\n", Array->Slope[i]);
	}
	fclose(Slopes);
	SlopesAU = fopen("<File Name Here>", "w");
	for( int i = 0; i < 3*Array->AUinX; i++){
		fprintf(SlopesAU, "%0.14lf\n", Array->SlopeAU[i]);
	}
	fclose(SlopesAU);

	CPTS = fopen("<File Name Here>", "w");
	fprintf(CPTS, "%0.14lf,%lf\n", CPCurStr, CPTem);
	fclose(CPTS);
}
void TUpdater(){
	FILE *CPTS;
	CPTS = fopen("<File Name Here>", "w");
	fprintf(CPTS, "%0.14lf,%lf\n", CPCurStr, CPTem);
	fclose(CPTS);
}
void PowerCrashRecover(LatticeArray *Array){
	FILE *in, *Slopes, *SlopesAU, *Constants;
	int scan;
	double PosX, PosY, slope;
	in = fopen("<File Name Here>", "r");
	for(int i = 0; i< Array->ArrayIndex; i++){
		scan = fscanf(in, "%lf,%lf\n",&PosX,&PosY);
		Array->AtomArray[i].Coords.x = PosX;
		Array->AtomArray[i].Coords.y = PosY;
	}
	fclose(in);

	Constants = fopen("<File Name Here>", "r");
	scan = fscanf(Constants, "%lf,%lf\n", &CPCurStr, &CPTem);
	fclose(Constants);
```

```
        Slopes = fopen("<File Name Here>", "r");
        for( int i = 0; i < 2*Array->NAinX; i++){
                scan = fscanf(in, "%lf\n",&slope);
                Array->Slope[i] = slope;

        }
        fclose(Slopes);
        SlopesAU = fopen("<File Name Here>", "r");
        for( int i = 0; i < 3*Array->AUinX; i++){
                scan = fscanf(in, "%lf\n",&slope);
                Array->SlopeAU[i] = slope;
        }
        fclose(SlopesAU);

}

void FileCleanup(){
        remove("<File Name Here>/CrashPowerDumpAtoms.csv");
        remove("<File Name Here>/CrashPowerDumpIt.csv");
        remove("<File Name Here>/CrashPowerDumpSISlopes.csv");
        remove("<File Name Here>/CrashPowerDumpAUSlopes.csv");
}
```

FileNameMaker.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <stdbool.h>
#include <string.h>
#include "Common.h"
//      Coded By: ©Victor C. Hayden (FLAG //GR645)
/*****************************************************************************
*       This function creates the output file names as the program will stretch
*       stepwise up to the maximum. Since the data has been processed anyway it
*       is best not to waste the already spent processing time so we can output
*       the data in between steps without having to specify all of the
*       individual filenames.
*
*       Creates FileNames as:
*               Input: ./Output/Output_Lat10_11_4_
*               Output: ./Output/Output_Lat10_11_4_1_000.csv for a 1.000 % stretch
*                                       (decimal has been replaced with underscore)
*****************************************************************************/
char* FileName (char *FileNameString, double StretchAmt){
        char StretchAmtFile[10];
        char *NewString = NULL;


        NewString = (char *)malloc((strlen(FileNameString)+12)*sizeof(char));

        for( int i = 0; i < strlen(NewString); i++){
                NewString[i] = '\0';
        }

        sprintf(StretchAmtFile, "%.4f", StretchAmt);

        strcat(NewString, FileNameString);
        strcat(NewString, StretchAmtFile);
```

```
	for( int i = 2; NewString[i] != '\0'; i++){
		if( NewString[i] == '.') NewString[i] = '_';
	}

	strcat(NewString, ".csv");

	if(NewString[0] != '.'){
		for( int i = 0; NewString[i] != '\0'; i++){
			NewString[i] = NewString[i+1];
		}
		for( int i = 0; NewString[i] != '\0'; i++){
			if(NewString[i+1] =='\0') NewString[i] = '\0';
		}
	}
	return NewString;
}
```

LatticeCopier.c

```
/*****************************************************************************
*       Contained here are the various files for the LatticeArray Updater
*       Further discriptions of what each sub-function does is given above the
*       actual Function
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include "Common.h"
//      Coded By: ©Victor C. Hayden (FLAG //GR645)
void LatticeArrayUpdater(LatticeArray *Array, LatticeArray *cpy){
        for(int i = 0; i < 2*Array->NAinX; i++){
                cpy->Slope[i] = Array->Slope[i];
        }
        for(int i = 0; i < 3*Array->AUinX; i++){
                cpy->SlopeAU[i] = Array->SlopeAU[i];
        }
        for(int i = 0; i < Array->ArrayIndex; i++){
                cpy->AtomArray[i].Coords.x = Array->AtomArray[i].Coords.x;
                cpy->AtomArray[i].Coords.y = Array->AtomArray[i].Coords.y;
        }
}
```

```
/*****************************************************************************
*       Contained within are the functions associated with creating the various
*       arrays for the Simulated "Annealing" function located within SimAnn.c.
*       These functions are commented where necessary, however for the indexing of
*       the arrays, I would advise to try them by hand if it is not clear how
*       they work, as they contain both interger and modulus math.
*****************************************************************************/
//      Coded By: ©Victor C. Hayden (FLAG //GR645)
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <stdbool.h>
#include <limits.h>     //Access System number limits
#include "Common.h"
#include "dSFMT.h"   //Pseudo-Random Number Generator
/*****************************************************************************
*       This function creates the lines required for the GoldLineMover algorithm
*       below, this only needs to be run once for each of the LatticeArrays that
*       you create. Element 0 of the created array for each line is the total
*       number of atoms in each line. Since the lines for even versus uneven
*       numbers of rows will not be the same, they are filled with -1's otherwise.
*
*       Access the AtomInLine via  ArrayName->AtomsInLine[LineNumber][AtomNumber]
*
*       unsigned long int -> [0 -> 4294967295]
*
*       This function as well as the MinLineInit use both integer and modulus math,
*       given that, here is a breif note on how these work:
*
*****************************************************************************/
void GoldLineInt(LatticeArray *Array){
    int TotalSi;
            //Total Number of Silicon, used to maintain that the atom numbers in
    //the gold array are in the same as the main LatticeArray Array
```

```
        TotalSi = Array->NAinX*(Array->NAinY);

//Allocating the Arrays within the structure, DO NOT FREE AT END OF FUNCTION
        Array->SlopeAU = (double*)malloc((3*Array->AUinX)*sizeof(double));
        Array->GoldAtomsInLine = malloc((3*Array->AUinX)*
                sizeof(unsigned long int*)); //2D Array
        for( int i = 0; i < 3*Array->AUinX; i++){
                Array->GoldAtomsInLine[i] = malloc(((Array->LeverH / 3)+3) *
                sizeof(unsigned long int));
        }

        //Initalizing Slope Array
        for(int i = 0; i < 3*Array->AUinX; i++){
                Array->SlopeAU[i] = 0.0;
        }
        //Initalizing GoldAtomsInLine Array
        for( int i = 0; i < 3*Array->AUinX; i++){
                for( int j = 0; j < (Array->LeverH / 3)+2; j++){

                        Array->GoldAtomsInLine[i][j] = -1;
                }
        }
        //Number of Atoms in each columm
        if(Array->LeverH % 3 == 0){
                for(int i = 0; i < 3*Array->AUinX; i++){
                        Array->GoldAtomsInLine[i][0] = (Array->LeverH / 3);
                }
        }else if(Array->LeverH % 3 == 1){
                for(int i = 0; i < 3*Array->AUinX; i++){
                        if ((i % 3) == 0){
                                Array->GoldAtomsInLine[i][0] = (Array->LeverH / 3)+1;
                        }
                        else if(i % 3 == 1 ||i % 3 == 2 ){
                                Array->GoldAtomsInLine[i][0] = (Array->LeverH / 3);
                        }
                }
```

```
	}else if(Array->LeverH % 3 == 2){
		for(int i = 0; i < 3*Array->AUinX; i++){
			if ((i % 3) == 2){
				Array->GoldAtomsInLine[i][0] = (Array->LeverH / 3);
			}
			else if(i % 3 == 1 ||i % 3 == 0 ){
				Array->GoldAtomsInLine[i][0] = (Array->LeverH / 3)+1;
			}
		}
	}
	//Filling in the columms
	for( int j = 0; j < 3*Array->AUinX; j+=3){
		for( int i = 1; i <= Array->GoldAtomsInLine[j][0]; i++){
			Array->GoldAtomsInLine[j][i] = (i-1)* 3 * Array->AUinX
			+ (j/3)+TotalSi;
		}
	}
	for( int j = 1; j < 3*Array->AUinX; j+=3){
		for( int i = 1; i <= Array->GoldAtomsInLine[j][0]; i++){
			Array->GoldAtomsInLine[j][i] = (i-1)* 3 * Array->AUinX
			+ ((j-1)/3)+Array->AUinX+TotalSi;
		}
	}

	for( int j = 2; j < 3*Array->AUinX; j+=3){
		for( int i = 1; i <= Array->GoldAtomsInLine[j][0]; i++){
			Array->GoldAtomsInLine[j][i] = (i-1)* 3 * Array->AUinX
			+ ((j-2)/3)+2*Array->AUinX+TotalSi;
		}
	}
	if(true){NULL;} //GR645 Flag Line for GDB Does Nothing
}
void TmpGLine(LatticeArray *Array){
	Array->ProxArr = malloc((3*Array->AUinX)*sizeof(GSA));
	for(int h = 0; h < 3 * Array->AUinX; h++){
		Array->ProxArr[h].LineNum = 0;
		Array->ProxArr[h].PercentDiff = 0.0;
```

```
	}
	double CurSilY;
	double CurGoldY;
	for( int i = 0; i < 3*Array->AUinX; i++){
		CurGoldY = Array->AtomArray[Array->GoldAtomsInLine[i][1]].Coords.x;
		for( int j = 0; j < 2*Array->NAinX; j++){
			CurSilY = Array->AtomArray[Array->AtomsInLine[j][1]].Coords.x;
			if(CurSilY > CurGoldY){break;}
			if(fabs(CurGoldY - CurSilY) < 1e-6){
				Array->ProxArr[i].LineNum = j;
				Array->ProxArr[i].PercentDiff = 0.0;
				break;
				}
			Array->ProxArr[i].LineNum = j;
			Array->ProxArr[i].PercentDiff = (CurGoldY - CurSilY)/(SiBL / 2.0);
		}
	}
	if(true){NULL;}//GR645 Flag Line for GDB Does Nothing
}
/*****************************************************************************
*	This function creates the lines required for the LineMover algorithm below,
*	this only needs to be run once for each of the LatticeArrays that you
*	create. Element 0 of the created array for each line is the total number
*	of atoms in each line. Since the lines for even versus uneven numbers of
*	rows will not be the same, they are filled with -1's otherwise.
*
*	Access the AtomInLine via  ArrayName->AtomsInLine[LineNumber][AtomNumber]
*	Coded By: ©Victor C. Hayden (FLAG //GR645)
*****************************************************************************/
void MinLineInit(LatticeArray *Array){

	//Allocating the Arrays within the structure,
	//DO NOT FREE AT END OF FUNCTION
	Array->Slope = (double*)malloc((2*Array->NAinX)*sizeof(double));
	Array->AtomsInLine = malloc((2*Array->NAinX)*sizeof(unsigned long int*));
	for( int i = 0; i < 2*Array->NAinX; i++){
		Array->AtomsInLine[i] = malloc(((Array->NAinY / 2)+2) *
```

```
        sizeof(unsigned long int));
}

//Initalizing Slope Array
for(int i = 0; i < 2*Array->NAinX; i++){
        Array->Slope[i] = 0.0;
}
//Initalizing AtomsInLine Array
for( int i = 0; i < 2*Array->NAinX; i++){
        for( int j = 0; j < (Array->NAinY / 2)+2; j++){

                Array->AtomsInLine[i][j] = -1;
        }
}
//Number of Atoms in each columm
if(Array->NAinY % 2 == 0){
        for(int i = 0; i < 2*Array->NAinX; i++){
                Array->AtomsInLine[i][0] = (Array->NAinY / 2);
        }
}else if(Array->NAinY % 2 == 1){
        for(int i = 0; i < 2*Array->NAinX; i++){
                if ((i % 2) == 0){
                        Array->AtomsInLine[i][0] = (Array->NAinY / 2);
                }
                else if(i % 2 == 1){
                        Array->AtomsInLine[i][0] = (Array->NAinY / 2)+1;
                }
        }
}
//Filling in the columms
for( int j = 1; j < 2*Array->NAinX; j+=2){
        for( int i = 1; i <= Array->AtomsInLine[j][0]; i++){
                Array->AtomsInLine[j][i] = (i-1)* 2 * Array->NAinX + (j/2);
        }
}

for( int j = 0; j < 2*Array->NAinX; j+=2){
```

```
            for( int i = 1; i <= Array->AtomsInLine[j][0]; i++){
                Array->AtomsInLine[j][i] = ((i-1)* 2 * Array->NAinX
                + ((j-1)/2))+Array->NAinX;
                if ( j > 0) { Array->AtomsInLine[j][i]++;}
            }
        }

        if(true){NULL;} //GR645 Flag Line for GDB Does Nothing

        //Only need to do this if gold is present
        if(Array->AUinX > 0){
            GoldLineInt(Array);
            TmpGLine(Array);
        }
}

/*****************************************************************************
*       This function is responsible for moving all of the GOLD minimization lines
*       by a random small amount Randomizes the Slopes of the lines, then
*       rotates them using a rotation matrix
*       VH
*****************************************************************************/
void GoldLineMover (LatticeArray *Array){
        double Slope,x,y;
        for( int i = 1; i < 3*Array->AUinX-2; i++){
            Array->SlopeAU[i] = Array->Slope[Array->ProxArr[i].LineNum]
            + Array->ProxArr[i].PercentDiff
            * (Array->Slope[Array->ProxArr[i].LineNum+1]
            -Array->Slope[Array->ProxArr[i].LineNum]);
        }
        Array->SlopeAU[3*Array->AUinX-2] = Array->SlopeAU[3*Array->AUinX-3]
                                                    + Array->SlopeAU[1];
        Array->SlopeAU[3*Array->AUinX-1] = Array->SlopeAU[3*Array->AUinX-2]
                                                    + Array->SlopeAU[2];
        for( int j = 1; j < 3*Array->AUinX; j++){
            for( int h = 1; h <= Array->GoldAtomsInLine[j][0]; h++){
                if(Array->GoldAtomsInLine[j][h] == -1) {continue;}
```

```
                x = Array->AtomArray[Array->GoldAtomsInLine[j][h]].Coords.x*
                        cos(Array->SlopeAU[j]) -
                        Array->AtomArray[Array->GoldAtomsInLine[j][h]].Coords.y*
                        sin(Array->SlopeAU[j]);
                y = Array->AtomArray[Array->GoldAtomsInLine[j][h]].Coords.y*
                        cos(Array->SlopeAU[j]) +
                        Array->AtomArray[Array->GoldAtomsInLine[j][h]].Coords.x*
                        sin(Array->SlopeAU[j]);
                Array->AtomArray[Array->GoldAtomsInLine[j][h]].Coords.x = x;
                Array->AtomArray[Array->GoldAtomsInLine[j][h]].Coords.y = y;
            }

        }
        if(true){NULL;} //GR645 Flag Line for GDB Does Nothing
}

/*****************************************************************************
*       This function is responsible for moving all of the SILICON minimization
*       lines by a random small amount Randomizes the Slopes of the lines, then
*       rotates them using a rotation matrix
*       VH
*****************************************************************************/
void LineMover(LatticeArray *Array, uint32_t Seed){
        dsfmt_t dsfmt;
        double deg, x, y, Slope;
        deg = -4.65e-8;
        deg -= deg*((3000000.0 - CPTem)/3000000.0);
        dsfmt_init_gen_rand(&dsfmt, Seed); //Seed must be between 0 and 4294967295
        for( int i = 1; i < 2*Array->NAinX-1; i++){
                Slope = -(dsfmt_genrand_open_open(&dsfmt)-1.0)* deg;
                for( int j = i; j < 2*Array->NAinX; j++){
                        Array->Slope[j] += Slope;
                }
        }
        Array->Slope[2*Array->NAinX -1] = Array->Slope[2*Array->NAinX -2]
                                                        +Array->Slope[1];
        for( int j = 1; j < 2*Array->NAinX; j++){
```

```
            for( int h = 1; h <= Array->AtomsInLine[j][0]; h++){
                if(Array->AtomsInLine[j][h] == -1) {continue;}
                x = Array->AtomArray[Array->AtomsInLine[j][h]].Coords.x*
                    cos(Array->Slope[j]) -
                    Array->AtomArray[Array->AtomsInLine[j][h]].Coords.y*
                    sin(Array->Slope[j]);
                y = Array->AtomArray[Array->AtomsInLine[j][h]].Coords.y*
                    cos(Array->Slope[j]) +
                    Array->AtomArray[Array->AtomsInLine[j][h]].Coords.x*
                    sin(Array->Slope[j]);
                Array->AtomArray[Array->AtomsInLine[j][h]].Coords.x = x;
                Array->AtomArray[Array->AtomsInLine[j][h]].Coords.y = y;
            }

        }
        if(true){NULL;} //GR645 Flag Line for GDB Does Nothing
        //Only need to do this if gold is present
        if(Array->AUinX > 0){
            GoldLineMover(Array);
        }
}
/*****************************************************************************
 * The Algorithm as is does not allow for a large amount of motion in anything
 * but the forward direction, therefore as a result, a check is made after
 * every itterative movement such that if the top surface is longer than is
 * possible under the current total amount of maximum stretch then the
 * configuration is considered to be unacceptible and as a result it is
 * rejected.
 * VH
 *****************************************************************************/
bool RejectAcceptSurfaceLength(LatticeArray *Array){
        double Top = 0.0;
        int count = 0;
        for(int i = 0; i < Array->NumberSprings; i++){
            if(SpringArray[i].Surface == false){
                continue;
            }
```

```
            else if(SpringArray[i].Surface == true){
                Top += sqrt(pow((Array->AtomArray[SpringArray[i].Atom1].Coords.x -
                                Array->AtomArray[SpringArray[i].Atom2].Coords.x),2.0) +
                                pow((Array->AtomArray[ SpringArray[i].Atom1].Coords.y -
                                Array->AtomArray[SpringArray[i].Atom2].Coords.y),2.0));
                count++;
                continue;
            } else {
            printf("RejectAccept has Failed to Indetify the atom.... Exiting\n");
            exit(-1);}
        }

        if( Top > (Array->SurfaceBLength * (count))) return false;
        return true;
}
```

Rotate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Common.h"
//      Coded By: ©Victor C. Hayden (FLAG //GR645)
/*****************************************************************************
**
*       This function rotates the Cantilever contained within Array by theta radian
*       degrees. Since computers have difficulty in handling vertical lines as would
*       be created by the normal boundaries, infinite slope, a simple rotation by
*       90 degrees yields a cantilever that is vertical hence no more problems, just
*       keep in mind that everything is vertical so it maybe difficult to visualize.
*
*****************************************************************************/
void RotateEntireCantilever(LatticeArray *Array, double Theta){
        double x,y;
        for(int i = 0; i < Array->ArrayIndex; i++){
            x = (Array->AtomArray[i].Coords.x * cos(Theta))
                        + (Array->AtomArray[i].Coords.y * sin(Theta));
```

```
            y = (-Array->AtomArray[i].Coords.x * sin(Theta))
                        + (Array->AtomArray[i].Coords.y * cos(Theta));
            Array->AtomArray[i].Coords.x = x;
            Array->AtomArray[i].Coords.y = y;
        }

}
```

SimAnn.c

```
/*****************************************************************************
*       Modified Metropolis-Hastings Algorithm for Simulated Annealing
*
*       First off, this is not "Annealing" in the strictest form of the word. But
*       is however a computational analog to the process of Annealing. It allows
*       the system to explore regions that would be otherwise forbidden under the
*       normal minimization scheme. With most methods the object is to minimize
*       the function analytically until a minima is found. However this found
*       minimum will be a local minimum not necessarily the Global minima. So
*       inorder to avoid the possibility of getting stuck in a local minima, a
*       shallow one anyway, we can use this well documented method to find
*       a best minima for the system.
*
*       So the algorithm works like this, for each value of the control parameter
*       T we are going to select a pseudo-random collection of atomic movements
*       and calculate their collective lattice energy for each configuration.
*       This energy is to be compared to two other energies, these being the
*       current best and the current good energies. The current best energy is
*       the configuration with the lowest lattice energy overall thus far.
*       The current good energy is the configuration that is not the lowest but
*       has been accepted by the algorithm. This is howthis algorithm allows for
*       so called uphill motion in the energy, or it allows the system to move
*       to an higher energy state thus maybe getting out of a local minima and
*       then it continues the minimization. In order to tell if this good
*       energy will be accepted or not we do a comparison between the
*       AcceProb = exp(-(TempCurrent - Current)/T)
*       and a randomly choosen value between 0 and 1 if the random number is less
```

```
*       that AcceProb then take it as being the good energy otherwise continue.
*       Coded By: ©Victor C. Hayden (FLAG //GR645)
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <stdbool.h>
#include <limits.h>     //Access System number limits
#include "Common.h"
#include "dSFMT.h"
//double SIMD Fast Merseene Twister (Very Good random number generator, gives
//very good quality Random Numbers)
//Checkout documentation on pseudo random numbers to see why this is important

#define MAX_ITS 100000      //Max iterations per temperature
#define MIN_ITS 750  //Min iterations per temperature
#define MAX_T 3000000       //Max value of Control Parameter
#define MIN_T 2000000       //Min value of Control Parameter
#define MaxTempIter 200 //Max Number of Temperature iterations
//Sub-function for moving Atoms
extern void LineMover (LatticeArray *Array, uint32_t Seed);

/*****************************************************************************
*       Run the Simulated "Annealing" Algorithm
*       Variables explained where required
*****************************************************************************/
void RunSimAnn (LatticeArray *Array, LatticeArray *BestArray,
                LatticeArray *CurrentArray, LatticeArray *SpaceArray,
                int argCommand, char * argv[]){
        dsfmt_t dsfmt;          //Error Variable
        uint32_t Seed;//unsigned 32 bit integer, Random Seed Value for DSFMT
        //Energies, Best, Quasi-best, and current working
        double CurBest, Current, TempCurrent;
        double T; //Current Value of the Control Variable
        double AcceProb; //Acceptance Probability
        int TriedSoln = 0; //Number of tried Solutions
```

```
FILE *Out; //DEBUG
//Randomize Seed based on Time, only time this happens, otherwise Seed is
//determined from a random from the previous seed
srand(time(NULL));
Seed = (rand() % UINT_MAX);
//dSFMT Initializer
dsfmt_init_gen_rand(&dsfmt, Seed); //Seed must be between 0 and 4294967295

//Determining Initial Values for Energies
CurBest = Current = DetermineEnergy(CurrentArray);//Starting lattice Energy
//Randomizing Seed Value
Seed = dsfmt_genrand_open_open(&dsfmt) * UINT_MAX;
if(recovery){
        T = CPTem;
} else {
        T = MAX_T;
        CPTem = T;
}

for( int Tindex = 0; Tindex < MaxTempIter; Tindex++){
        for( int iter = 0; iter < MAX_ITS; iter++){

                LatticeArrayUpdater(CurrentArray, SpaceArray);
                LineMover(SpaceArray, Seed);
                //New Random seed created for each iteration
                Seed = dsfmt_genrand_open_open(&dsfmt) * UINT_MAX;
                TempCurrent = DetermineEnergy(SpaceArray);
                //if the temp current energy is lower that the current
                //good energy so far then accept as is
                if((TempCurrent < Current) &&
                        RejectAcceptSurfaceLength(SpaceArray)){
                        LatticeArrayUpdater(SpaceArray, CurrentArray);
                        Current = TempCurrent;
                }

                if(TempCurrent > Current){TriedSoln++;}
                //if the current energy is lower that the best so far then accept
```

```
                if((Current < CurBest) && RejectAcceptSurfaceLength(SpaceArray)){
                        LatticeArrayUpdater(SpaceArray, BestArray);
                        CurBest = Current;
                        CrashPowerProtect(BestArray, argCommand, argv);
                        TriedSoln = 0;
                }else{ //Might still accept this higher energy state
                        AcceProb = exp(-(TempCurrent - Current)/T);
                        if ((dsfmt_genrand_open_open(&dsfmt) < AcceProb) &&
                                        RejectAcceptSurfaceLength(SpaceArray)){
                                LatticeArrayUpdater(SpaceArray, CurrentArray);
                                Current = TempCurrent;
                                TriedSoln = 0;
                        }
                }
                //Stop Criteria / Check to break early
                if (TriedSoln > 1e4) {
                        TriedSoln = 0;
                        break;
                }
                //Check for Restart if the energy is very different then restart
                //at a better position
                if (TempCurrent > 1.25*CurBest){
                        LatticeArrayUpdater(BestArray, CurrentArray);
                        Current = DetermineEnergy(CurrentArray);
                        TriedSoln = 0;
                }
                TriedSoln++;
        }
//Mod the control Parameter, 2 options Simple and Exponential Cooling Sched
        T *= 0.85;//Simple Cooling Schedule 1
        CPTem = T;//Crash Protect Tempterature
        TUpdater();//Update the Crash Protect File
        //T = MAX_T * exp((-0.05)*(Tindex)); //Exponential Cooling Sched

        //Check for early T breakout
        //Break if T low
        if (T < 1) break;
```

```
	}
	//Copy the Best Array configuration found into F_Array
	//(renamed here to Array just in this Local scope)
	LatticeArrayUpdater(BestArray, Array);}
```

SpringLatCreate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include "Common.h"
//	Coded By: ©Victor C. Hayden (FLAG //GR645)
/*****************************************************************************
*	This function creates the spring lattice that is used for the determine
*	energy function. The created lattice is that of nearest and next nearest
*	neighbours, and works with and without the presence of gold.
*****************************************************************************/
void createSpringLattice (LatticeArray *Array){
	double rad;
	Array->SSpring = 0;
	Array->NumberSprings = 0;
	Array->LBound.AtomsInBound = NULL;

	Array->LBound.NumAtomsInBound = 1;
	Array->LBound.AtomsInBound = realloc(Array->LBound.AtomsInBound,
				(Array->LBound.NumAtomsInBound)*sizeof(unsigned long
int));
	Array->LBound.AtomsInBound[0] = 0;
	Array->RBound.NumAtomsInBound = 0;
	Array->RBound.AtomsInBound = NULL;
	if (SpringArray != NULL){free(SpringArray); SpringArray = NULL;}
	for(int i = 0; i < (Array->ArrayIndex); i++){
		//Assume by default that the atom is not in a boundary
		Array->AtomArray[i].AtomBound = false;
		if((i+1)< Array->ArrayIndex){
			if(fabs(Array->AtomArray[i].Coords.y -
```

```
                    Array->AtomArray[i+1].Coords.y) > 1e-3){
            Array->LBound.AtomsInBound = realloc(Array->LBound.AtomsInBound,
            (Array->LBound.NumAtomsInBound + 1) * sizeof(unsigned long int));
            Array->LBound.AtomsInBound[Array->LBound.NumAtomsInBound] = i+1;

                Array->LBound.NumAtomsInBound += 1;

            Array->RBound.AtomsInBound = realloc(Array->RBound.AtomsInBound,
            (Array->RBound.NumAtomsInBound + 1) * sizeof(unsigned long int));
            Array->RBound.AtomsInBound[Array->RBound.NumAtomsInBound] = i;

                Array->RBound.NumAtomsInBound += 1;
            }
}
            //Here it is not necessary to got though the entire array
            for(int j = i+1; (j < i + 2.2 * Array->NAinX) &&
                        (j< Array->ArrayIndex); j++){
                //Find the radius between the atoms
                rad = sqrt(pow((Array->AtomArray[i].Coords.x -
                Array->AtomArray[j].Coords.x),2) +
                pow((Array->AtomArray[i].Coords.y -
                Array->AtomArray[j].Coords.y),2));
                //Here we handle each of the possible cases
            if (Array->AtomArray[i].AtomType == Array->AtomArray[j].AtomType){
                        if(Array->AtomArray[i].AtomType == 'S'){
                            if (rad <= (1.75*SS_L)){

                                SpringArray = (Spring *)realloc(SpringArray,
                                (Array->NumberSprings+1)*sizeof(Spring));
                                //Must be a Next Nearest Neighbour
                                if (rad > 1.2*SS_L){
                            SpringArray[Array->NumberSprings].NNN = true;
                                }else{
                            SpringArray[Array->NumberSprings].NNN = false;
                                }
                                //Setting SpringArray Values
                            SpringArray[Array->NumberSprings].Length = rad;
```

```
                SpringArray[Array->NumberSprings].Atom1 = i;
                SpringArray[Array->NumberSprings].Atom2 = j;
                        //Which of the springs are on the surface
                        if ((Array->AtomArray[i].AtomSB == 's')&&
                        (Array->AtomArray[j].AtomSB == 's')&&
                        ( i == j - 1)){
                SpringArray[Array->NumberSprings].Length = rad;
                                Array->SurfaceBLength = SiBL;
                                Array->SSpring+=1;
                        }
                        Array->NumberSprings+=1;
                }
        }
        else {
                if (rad <= (1.75*GG_L)){
                        SpringArray = (Spring *)realloc(SpringArray,
                        (Array->NumberSprings+1)*sizeof(Spring));
                        //Must be a Next Nearest Neighbour
                        if (rad > 1.2*GG_L){
                SpringArray[Array->NumberSprings].NNN = true;
                        }else{
                SpringArray[Array->NumberSprings].NNN = false;
                        }
                        //Setting SpringArray Values
                SpringArray[Array->NumberSprings].Length = rad;
                SpringArray[Array->NumberSprings].Atom1 = i;
                SpringArray[Array->NumberSprings].Atom2 = j;
                        //Which of the springs are on the surface
                        if ((Array->AtomArray[i].AtomSB == 's')&&
                        (Array->AtomArray[j].AtomSB == 's')&&
                        ( i == j - 1)){
        SpringArray[Array->NumberSprings].Length = AU_L;
                                Array->SSpring+=1;
                        }
                        Array->NumberSprings+=1;
                }
        }
```

```
                    }
                    else {
                        if (rad <= (1.75*GS_L)){
                            SpringArray = (Spring *)realloc(SpringArray,
                            (Array->NumberSprings+1)*sizeof(Spring));
                        if (rad > 1.2*GS_L){//Must be a Next Nearest Neighbour
                            SpringArray[Array->NumberSprings].NNN = true;
                            }else{
                            SpringArray[Array->NumberSprings].NNN = false;
                            }
                            //Setting SpringArray Values
                            SpringArray[Array->NumberSprings].Length = rad;
                            SpringArray[Array->NumberSprings].Atom1 = i;
                            SpringArray[Array->NumberSprings].Atom2 = j;
                            Array->NumberSprings+=1;
                        }
                    }
                }
        }

if ((Array->AtomArray[Array->ArrayIndex -1].AtomType == 'G')&&
(Array->AtomArray[Array->ArrayIndex -2].AtomType == 'G')){
        Array->SurfaceBLength = AU_L;
}
else if ((Array->AtomArray[Array->ArrayIndex -1].AtomType == 'S')&&
(Array->AtomArray[Array->ArrayIndex -2].AtomType == 'S')){
        Array->SurfaceBLength = SiBL;
}

Array->RBound.AtomsInBound = realloc(Array->RBound.AtomsInBound,
(Array->RBound.NumAtomsInBound + 1) * sizeof(unsigned long int));
Array->RBound.AtomsInBound[Array->RBound.NumAtomsInBound] =

Array->ArrayIndex-1;
Array->LBound.NumAtomsInBound -= 1;
//Setting the last of the Spring values
for( int i = 0; i <= Array->LBound.NumAtomsInBound; i++) {
```

```
                Array->AtomArray[Array->LBound.AtomsInBound[i]].AtomBound = true;
                Array->AtomArray[Array->LBound.AtomsInBound[i]].First = true;
        }
        for( int i = 0; i <= Array->RBound.NumAtomsInBound; i++) {
                Array->AtomArray[Array->RBound.AtomsInBound[i]].AtomBound = true;
                Array->AtomArray[Array->RBound.AtomsInBound[i]].First = false;
        }

}
void SurfaceSpringAssignment(LatticeArray *Array, int NumSur){
        for(int i = 0; i < Array->ArrayIndex; i++){
                if( Array->AtomArray[i].AtomVirtual ){
                        Array->AtomArray[i-NumSur].AtomSB = 's';
                        Array->AtomArray[i-NumSur-1].AtomSB = 's';
                }
        }
        for(int i = 0; i < Array->NumberSprings; i++){
                if((fabs(Array->AtomArray[SpringArray[i].Atom1].Coords.x -
                Array->AtomArray[SpringArray[i].Atom2].Coords.x) < 1e-6)&&
                (Array->AtomArray[SpringArray[i].Atom1].AtomSB == 's')&&
                (Array->AtomArray[SpringArray[i].Atom2].AtomSB == 's')){
                        SpringArray[i].Surface = true;
                }
                else {
                        SpringArray[i].Surface = false;
                }
        }
}
```

Structures.h

```
/*****************************************************************************
*       Contained here are most of the custom datatypes that are used in this
*       program with an explanation of what each varible inside the structure is.
*       Coded By: ©Victor C. Hayden (FLAG //GR645)
*****************************************************************************/
#include <stdbool.h>
//2D point to avoid 2d arrays or 2* indexing
typedef struct {double x,y;}Point;
//Information movement of gold
typedef struct {
        int LineNum;
        double PercentDiff;
}GSA;
//Everything that an atom needs to know about itself
typedef struct{
        unsigned long int AtomNumber; //Atom Number
        Point Coords; //Location in Real Space
        char AtomSB; //Surface VS Bulk
        char AtomType; //Si VS Au
        bool AtomBound; //Atom In Boundary
        bool First; //First in the row?
        bool AtomVirtual; //Is the Atom Real, Virtual Atoms DNE
}Atom;
//Everything that an Spring needs to know about itself
typedef struct{
        unsigned long int Atom1; //Which atoms it this spring connected too
        unsigned long int Atom2;
        double Length; //Natural spring length
        bool NNN; //Next Nearest Neighbour? (to know the correct spring constant)
        bool Surface; //True for Actual Surface Springs
}Spring;
//Defines the bounsary
typedef struct{
        unsigned long int NumAtomsInBound; //Number of Atoms in the Boundary
        unsigned long int *AtomsInBound; //Which atoms are in the boundary
```

```
}Bound;
//Point Slope Line (helps keep everything together)
typedef struct{
        double Slope;
        double Intercept;
}PointSlopeLine;
//This is the main datatype and contains all info about the lattice
typedef struct{
        unsigned long int NumberSprings;//How many Springs are in the lattice
        unsigned long int SSpring;          //How many surface Springs are there
        unsigned long int ArrayIndex;       //Number of Atoms in the Array
        unsigned long int NAinX;            //Number of Silicon Atoms in X
        unsigned long int AUinX;            //Number of Gold Atoms in X
        unsigned long int NAinY;            //Total Number of rows of atoms
        unsigned long int LeverH;           //Number of rows of gold
   unsigned long int NumberSurfaceAtoms; //Total Number of Surface Atoms
        double SurfaceBLength;                  //Desired Surface bondlength
        Atom *AtomArray;                        //Array of Atoms (Atom defined above)
                Bound RBound;                           //Right Boundary
                Bound LBound;                           //Left Boundary
                double *Slope;                          //Slopes of the silicon minimization
lines
        double *SlopeAU;                        //Slopes of the Gold minimization lines
        long int **AtomsInLine;                 //Atoms in each silicon minimization lines
        long int **GoldAtomsInLine;         //Atoms in each Gold minimization lines
        GSA *ProxArr;
}LatticeArray;
//Grouping the boundary lines
typedef struct{
        PointSlopeLine RLine;       //Right
        PointSlopeLine LLine;       //Left
}BoundLines;
```

Appendix F: Graphical User Interface Source Code (Java, Bash, Expect, C99)

```
public class Atom{
        public Atom(long Index, double x, double y, char BulkSurface,
                                                    char SilGold, char rv){
     Ind = Index;
    X = x;
    Y = y;
    BS = BulkSurface;
    SG = SilGold;
    RV = rv;
       }

    public long getIndex(){
      return Ind;
    }
    public double getX(){
      return X;
    }
    public double getY(){
      return Y;
    }
    public char getBS(){
      return BS;
    }
    public char getSG(){
      return SG;
    }
    public char getRV(){
      return RV;
    }
    public boolean getBound(){
      return Bound;
    }
    public String getInStr(){
      return Long.toString(Ind);
```

```
}
public String getXStr(){
   return Double.toString(X);
}
public String getYStr(){
   return Double.toString(Y);
}
public String getBSstr(){
   return Character.toString(BS);
}
public String getSGstr(){
   return Character.toString(SG);
}
public String getRVstr(){
   return Character.toString(RV);
}
public void setBound(){
   Bound = !Bound;
}
public void setIndex(long I){
   Ind = I;
}
public void setX(double x){
   X = x;
}
public void setY(double y){
   Y = y;
}
public void setBS(char bs){
   BS = bs;
}
public void setSG(char sg){
   SG = sg;
}
public void setRV(char rv){
   RV = rv;
}
```

```
private long Ind = 0;
private double X=0;
private double Y=0;
private char BS = 'b';
private char SG = 'S';
private char RV = 'r';
private boolean Bound = false;
}
```

MyCustomFilter.java

```
import java.io.File;

public class MyCustomFilter extends javax.swing.filechooser.FileFilter {

    @Override
    public boolean accept(File file) {
      // Allow only directories, or files with ".csv" extension
     return file.isDirectory() || file.getAbsolutePath().endsWith(".csv");
    }
    @Override
    public String getDescription() {
      return "Cantilever Lattice Files (*.csv)";
    }
}
```

SAfilter.java

```
import java.io.File;

public class SAfilter extends javax.swing.filechooser.FileFilter {
```

```
    @Override
    public boolean accept(File file) {
      // Allow only directories, or files with ".jpg" extension
     return file.isDirectory() || file.getAbsolutePath().endsWith(".jpg");
    }
    @Override
    public String getDescription() {
     return "Image Files (*.jpg)";
    }
}
```

LatticePlot.java

```
import java.awt.Color;
import javax.swing.JFrame;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.DefaultXYDataset;
import org.jfree.chart.renderer.AbstractRenderer;

  public class LatticePlot{
    public LatticePlot(String title, Atom[] AtomArray,int AUinX, int AUinY
              ,int NAinX,int NAinY, Color Silicon, Color Gold, boolean DisSurLine,
              boolean DisSubLine, double SurAmp, double SurFreq, double SubAmp,
              double SubFreq, double Height, double Length, boolean ZG) {

      JFrame frame = new JFrame(title);
      frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
      DefaultXYDataset dataset1 = new DefaultXYDataset();
      double[][] Sil;
      double[][] Gol;
      Total = NAinX*NAinY+AUinX*AUinY;
```

```
VirtCounts(AtomArray);
if(Total <= 250000 ){
Sil = new double[2][RSIL];
Gol = new double[2][RGOL];
int SilCount = 0;
int GolCount = 0;
for(int i = 0; i < Total; i++){
  if(AtomArray[i].getRV()=='v'){
    continue;
  }
  if(AtomArray[i].getSG()=='S'){
    Sil[0][SilCount] = AtomArray[i].getX();
    Sil[1][SilCount] = AtomArray[i].getY();
    SilCount++;
  }else if(AtomArray[i].getSG()=='G'){
    Gol[0][GolCount] = AtomArray[i].getX();
    Gol[1][GolCount] = AtomArray[i].getY();
    GolCount++;
  }
}
} else if(ZG){
  int plotSil = 3*NAinX;
  Sil = new double[2][plotSil];
  Gol = new double[2][RGOL];
  int SilCount = 0;
  int GolCount = 0;
  for(int i = 0; i < Total; i++){
  if(AtomArray[i].getRV()=='v'){
    continue;
  }
  if(AtomArray[i].getSG()=='S'&&
                                ((i > (NAinX*NAinY-3*NAinX-1)))){
    Sil[0][SilCount] = AtomArray[i].getX();
    Sil[1][SilCount] = AtomArray[i].getY();
    SilCount++;
  }else if(AtomArray[i].getSG()=='G'){
    Gol[0][GolCount] = AtomArray[i].getX();
```

```
      Gol[1][GolCount] = AtomArray[i].getY();
      GolCount++;
    }
    }
  }else{
    int plotSil = 6*NAinX + 2*NAinY - 12;
    Sil = new double[2][plotSil];
    Gol = new double[2][RGOL];
    int SilCount = 0;
    int GolCount = 0;
    for(int i = 0; i < Total; i++){
    if(AtomArray[i].getRV()=='v'){
      continue;
    }
    if(AtomArray[i].getSG()=='S'&& ((i < 3*NAinX)||
                                (i > (NAinX*NAinY-3*NAinX))||
                                (AtomArray[i].getBound()))){
      Sil[0][SilCount] = AtomArray[i].getX();
      Sil[1][SilCount] = AtomArray[i].getY();
      SilCount++;
    }else if(AtomArray[i].getSG()=='G'){
      Gol[0][GolCount] = AtomArray[i].getX();
      Gol[1][GolCount] = AtomArray[i].getY();
      GolCount++;
    }
  }
  }

  dataset1.addSeries("Silicon",Sil);
  dataset1.addSeries("Gold",Gol);
  XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
  renderer.setSeriesLinesVisible(0, false);
  renderer.setSeriesShapesVisible(0, true);
  renderer.setSeriesLinesVisible(1, false);
  renderer.setSeriesShapesVisible(1, true);
  if(DisSurLine){
   double[][] surline = new double[2][10000];
```

```
        for(int i =0; i<10000;i++){
          surline[0][i]=(double)(i*(Length/10000));
          surline[1][i]=Fun(surline[0][i],SurAmp,SurFreq,Length,Height);
        }
        dataset1.addSeries("SurLine",surline);
        renderer.setSeriesLinesVisible(2, true);
        renderer.setSeriesShapesVisible(2, false);
      }

      NumberAxis xAxis = new NumberAxis("X");
      xAxis.setAutoRangeIncludesZero(false);
      xAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
      NumberAxis yAxis = new NumberAxis("Y");
      yAxis.setAutoRangeIncludesZero(false);
      yAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
      renderer.setSeriesPaint(0, Silicon);
      renderer.setSeriesPaint(1, Gold);


      XYPlot plot = new XYPlot(dataset1,xAxis,yAxis,renderer);

     chart = new JFreeChart(plot);
     ChartPanel cpanel = new ChartPanel(chart);
     frame.getContentPane().add(cpanel);
     frame.pack();
     frame.setVisible(true);

}

    public static JFreeChart getchart(){
       return chart;
    }
     public double Fun(double x,double Amp,double Freq,
                 double length,double Height){
        return 0.5*Amp*Math.cos(2*Math.PI*x*Freq/length)+Height-Amp/2.0;
     }
    private void VirtCounts(Atom[] Atoms){
```

```
        for(int i = 0; i < Total; i++){
            if(Atoms[i].getRV()=='v'){
              continue;
            }else if(Atoms[i].getRV()=='r'){
              if(Atoms[i].getSG()=='S'){
                RSIL++;
              }else if(Atoms[i].getSG()=='G'){
                RGOL++;
              }
            }
          }
        }
        private int RGOL = 0;
        private int RSIL = 0;
        private int Total = 0;
        private static JFreeChart chart;

    }
```

LatticePlotFromBackup.java

```
import java.awt.Color;
import javax.swing.JFrame;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.DefaultXYDataset;
import org.jfree.chart.renderer.AbstractRenderer;

    public class LatticePlotFromBackup{
        public LatticePlotFromBackup(String title, Atom[] AtomArray,int AUinX,
                int AUinY,int NAinX,int NAinY, Color Silicon, Color Gold,
                boolean DisSurLine, boolean DisSubLine, double SurAmp,
```

```
            double SurFreq, double SubAmp, double SubFreq,
            double Height, double Length) {

JFrame frame = new JFrame(title);
frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
DefaultXYDataset dataset1 = new DefaultXYDataset();

Total = NAinX*NAinY+AUinX*AUinY;
VirtCounts(AtomArray);

double[][] Sil = new double[2][RSIL];
double[][] Gol = new double[2][RGOL];
int SilCount = 0;
int GolCount = 0;
for(int i = 0; i < Total; i++){
  if(AtomArray[i].getRV()=='v'){
    continue;
  }
  if(AtomArray[i].getSG()=='S'){
    Sil[0][SilCount] = AtomArray[i].getX();
    Sil[1][SilCount] = AtomArray[i].getY();
    SilCount++;
  }else if(AtomArray[i].getSG()=='G'){
    Gol[0][GolCount] = AtomArray[i].getX();
    Gol[1][GolCount] = AtomArray[i].getY();
    GolCount++;
  }
}

dataset1.addSeries("Silicon",Sil);
dataset1.addSeries("Gold",Gol);
XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
renderer.setSeriesLinesVisible(0, false);
renderer.setSeriesShapesVisible(0, true);
renderer.setSeriesLinesVisible(1, false);
renderer.setSeriesShapesVisible(1, true);
if(DisSurLine){
```

```
        double[][] surline = new double[2][10000];
        for(int i =0; i<10000;i++){
          surline[0][i]=(double)(i*(Length/10000));
          surline[1][i]=Fun(surline[0][i],SurAmp,
                                                    SurFreq,Length,Height);
        }
        dataset1.addSeries("SurLine",surline);
        renderer.setSeriesLinesVisible(2, true);
        renderer.setSeriesShapesVisible(2, false);
       }

       NumberAxis xAxis = new NumberAxis("X");
       xAxis.setAutoRangeIncludesZero(false);
       xAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

       renderer.setSeriesPaint(0, Silicon);
       renderer.setSeriesPaint(1, Gold);


       XYPlot plot = new XYPlot(dataset1,xAxis,new
                                                    NumberAxis("Y"),renderer);

      chart = new JFreeChart(plot);
      ChartPanel cpanel = new ChartPanel(chart);
      frame.getContentPane().add(cpanel);
      frame.pack();
      frame.setVisible(true);

}

    public static JFreeChart getchart(){
      return chart;
    }
     public double Fun(double x,double Amp,double Freq,
                double length,double Height){
       return 0.5*Amp*Math.cos(2*Math.PI*x*Freq/length)+Height-Amp/2.0;
     }
```

```
    private void VirtCounts(Atom[] Atoms){
      for(int i = 0; i < Total; i++){
        if(Atoms[i].getRV()=='v'){
          continue;
        }else if(Atoms[i].getRV()=='r'){
          if(Atoms[i].getSG()=='S'){
            RSIL++;
          }else if(Atoms[i].getSG()=='G'){
            RGOL++;
          }
        }
      }
    }
    private int RGOL = 0;
    private int RSIL = 0;
    private int Total = 0;
    private static JFreeChart chart;

  }
```

InputLatticeViz.java

```
import javax.swing.JFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.FastScatterPlot;
import org.jfree.data.xy.DefaultXYDataset;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

public class InputLatticeViz {
  public InputLatticeViz(String[][] SiliconLattice, String[][] GoldLattice,
                         int NAinX, int NAinY, int AUinY, int AUinX){
    XYSeriesCollection dataset = new XYSeriesCollection();
```

```
    XYSeries Silicon = new XYSeries("Silicon");
    XYSeries Gold = new XYSeries("Gold");
    if(AUinX > 0 && AUinY > 0){
      float [][] GoldLat = new float[2][AUinY*AUinX];
    }

    JFrame frame = new JFrame("Input Lattice");
    frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

    for(int i =0; i< NAinX*NAinY;i++){
      if(SiliconLattice[4][i].equals("r")&&
                    SiliconLattice[3][i].equals("S")){
        Silicon.add(Float.parseFloat(SiliconLattice[0][i]),
                    Float.parseFloat(SiliconLattice[1][i]));
      }
    }
    dataset.addSeries(Silicon);
    for(int i =0; i< AUinX*AUinY;i++){
      if(GoldLattice[4][i].equals("r")&&GoldLattice[3][i].equals("G")){
        Gold.add(Float.parseFloat(GoldLattice[0][i]),
                    Float.parseFloat(GoldLattice[1][i]));
      }
    }
    dataset.addSeries(Gold);
  }

}
```

MainWindow.java

```
import java.awt.Color;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Reader;
import java.io.StringWriter;
import java.io.Writer;
import javax.swing.JColorChooser;
import javax.swing.JFileChooser;
import javax.swing.UIManager;
import org.jfree.chart.ChartUtilities;


public class MainWindow extends javax.swing.JFrame {

  /** Creates new form MainWindow */
  public MainWindow() {
    initComponents();
    IniSetup();

  }

  @SuppressWarnings("unchecked")

  private void initComponents() {

    HoleSelect = new javax.swing.ButtonGroup();
    OpenLatticeFile = new javax.swing.JFileChooser();
    SAFileChooser = new javax.swing.JFileChooser();
    VizOutFile = new javax.swing.JFileChooser();
    buttonGroup1 = new javax.swing.ButtonGroup();
    SaveVizOut = new javax.swing.JFileChooser();
    SiliconColourChange = new javax.swing.JColorChooser();
    GoldColourChanger = new javax.swing.JColorChooser();
    buttonGroup2 = new javax.swing.ButtonGroup();
    buttonGroup3 = new javax.swing.ButtonGroup();
    jPanel1 = new javax.swing.JPanel();
```

```
OpenLat = new javax.swing.JButton();
MkANscr = new javax.swing.JButton();
QuiT = new javax.swing.JButton();
OpenLatCheck = new javax.swing.JCheckBox();
jSeparator1 = new javax.swing.JSeparator();
SelSaveLoc = new javax.swing.JButton();
SaveLatCheck = new javax.swing.JCheckBox();
jSeparator2 = new javax.swing.JSeparator();
VizInLat = new javax.swing.JButton();
jPanel2 = new javax.swing.JPanel();
ReSeT = new javax.swing.JButton();
jPanel4 = new javax.swing.JPanel();
SimName = new javax.swing.JTextField();
SimNumber = new javax.swing.JTextField();
jLabel6 = new javax.swing.JLabel();
jLabel7 = new javax.swing.JLabel();
Update_ACENET_Script = new javax.swing.JButton();
Email = new javax.swing.JTextField();
jLabel8 = new javax.swing.JLabel();
jPanel5 = new javax.swing.JPanel();
jScrollPane1 = new javax.swing.JScrollPane();
ACENET_Script_Field = new javax.swing.JTextArea();
jPanel6 = new javax.swing.JPanel();
ProgressBar = new javax.swing.JProgressBar();
jPanel7 = new javax.swing.JPanel();
InputFileNameDisplay = new javax.swing.JTextField();
jLabel10 = new javax.swing.JLabel();
DisplayOutName = new javax.swing.JTextField();
jLabel17 = new javax.swing.JLabel();
VizOutputPic = new javax.swing.JTextField();
jLabel26 = new javax.swing.JLabel();
jPanel8 = new javax.swing.JPanel();
jLabel14 = new javax.swing.JLabel();
jLabel15 = new javax.swing.JLabel();
jLabel16 = new javax.swing.JLabel();
Recov = new javax.swing.JCheckBox();
SurSprConst = new javax.swing.JTextField();
```

```
MaxStretch = new javax.swing.JTextField();
NumSteps = new javax.swing.JTextField();
jPanel9 = new javax.swing.JPanel();
jLabel18 = new javax.swing.JLabel();
jLabel19 = new javax.swing.JLabel();
SurAmp = new javax.swing.JTextField();
SurFreq = new javax.swing.JTextField();
jPanel10 = new javax.swing.JPanel();
jLabel22 = new javax.swing.JLabel();
jLabel23 = new javax.swing.JLabel();
SubAmp = new javax.swing.JTextField();
SubFreq = new javax.swing.JTextField();
jPanel11 = new javax.swing.JPanel();
jLabel9 = new javax.swing.JLabel();
NAinX = new javax.swing.JTextField();
jLabel11 = new javax.swing.JLabel();
NAinY = new javax.swing.JTextField();
jLabel13 = new javax.swing.JLabel();
AUinY = new javax.swing.JTextField();
AUinX = new javax.swing.JTextField();
jLabel12 = new javax.swing.JLabel();
MkNLat = new javax.swing.JButton();
jSeparator3 = new javax.swing.JSeparator();
jPanel13 = new javax.swing.JPanel();
SelVizOut = new javax.swing.JButton();
jPanel15 = new javax.swing.JPanel();
CompRun = new javax.swing.JRadioButton();
VizBack = new javax.swing.JRadioButton();
VizOutLat = new javax.swing.JButton();
SavOutLatViz = new javax.swing.JButton();
DisSubLine = new javax.swing.JCheckBox();
DisSurLine = new javax.swing.JCheckBox();
ZoomGoldLat = new javax.swing.JButton();
SavOutLatVizZoomGold = new javax.swing.JButton();
jPanel14 = new javax.swing.JPanel();
AppSurRough = new javax.swing.JButton();
AppSubRough = new javax.swing.JButton();
```

```
jPanel17 = new javax.swing.JPanel();
RunOnLocal = new javax.swing.JButton();
RunOnOther = new javax.swing.JButton();
RunOnACENET = new javax.swing.JButton();
jPanel16 = new javax.swing.JPanel();
jLabel27 = new javax.swing.JLabel();
jLabel28 = new javax.swing.JLabel();
ChangeSilColour = new javax.swing.JButton();
ChangeGoldColour = new javax.swing.JButton();
SiliconColorField = new javax.swing.JPanel();
GoldColorField = new javax.swing.JPanel();
jPanel3 = new javax.swing.JPanel();
jLabel1 = new javax.swing.JLabel();
jLabel2 = new javax.swing.JLabel();
UNAME = new javax.swing.JTextField();
UPASS = new javax.swing.JPasswordField();
jLabel3 = new javax.swing.JLabel();
SACE = new javax.swing.JRadioButton();
BSERV = new javax.swing.JRadioButton();
SaveCurLat = new javax.swing.JButton();

OpenLatticeFile.setFileFilter(new MyCustomFilter());

SAFileChooser.setAcceptAllFileFilterUsed(false);
SAFileChooser.setCurrentDirectory(new java.io.File("/home"));
SAFileChooser.setDialogType(javax.swing.JFileChooser.SAVE_DIALOG);
SAFileChooser.setFileSelectionMode(

    javax.swing.JFileChooser.DIRECTORIES_ONLY);

VizOutFile.setFileFilter(new MyCustomFilter());

SaveVizOut.setDialogType(javax.swing.JFileChooser.SAVE_DIALOG);
SaveVizOut.setFileFilter(new SAfilter());

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Gold / Silicon System Control Interface");
```

```
jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Operations:"));

OpenLat.setText("Open Lattice");
OpenLat.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    OpenLatActionPerformed(evt);
  }
});

MkANscr.setText("Make ACENET Script");
MkANscr.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    MkANscrActionPerformed(evt);
  }
});

QuiT.setText("Exit");
QuiT.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    QuiTActionPerformed(evt);
  }
});

OpenLatCheck.setFont(new java.awt.Font("DejaVu Sans", 0, 31));

jSeparator1.setBorder(javax.swing.BorderFactory.createEtchedBorder());

SelSaveLoc.setText("Select Save Location");
SelSaveLoc.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    SelSaveLocActionPerformed(evt);
  }
});

SaveLatCheck.setFont(new java.awt.Font("DejaVu Sans", 0, 31));
```

```
    jSeparator2.setBorder(javax.swing.BorderFactory.createEtchedBorder());

    VizInLat.setText("Visualize Current Input Lattice");
    VizInLat.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        VizInLatActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
        .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addComponent(VizInLat, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE)
          .addGroup(jPanel1Layout.createSequentialGroup()
            .addComponent(OpenLat, javax.swing.GroupLayout.DEFAULT_SIZE, 288,
Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(OpenLatCheck))
          .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
            .addComponent(SelSaveLoc, javax.swing.GroupLayout.DEFAULT_SIZE, 288,
Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(SaveLatCheck))
          .addComponent(QuiT, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE)
          .addComponent(jSeparator2, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE)
          .addComponent(MkANscr, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE)
```

```
            .addComponent(jSeparator1, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 314, Short.MAX_VALUE))
            .addContainerGap())
    );
    jPanel1Layout.setVerticalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(OpenLatCheck, javax.swing.GroupLayout.PREFERRED_SIZE, 27,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(OpenLat))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(SelSaveLoc)
            .addComponent(SaveLatCheck, javax.swing.GroupLayout.PREFERRED_SIZE, 27,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGap(8, 8, 8)
          .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addGap(8, 8, 8)
          .addComponent(VizInLat, javax.swing.GroupLayout.PREFERRED_SIZE, 35,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
          .addComponent(MkANscr)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jSeparator2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
          .addComponent(QuiT)
          .addContainerGap())
    );

    jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder("New Lattice or Modifiy
Current:"));
```

```
        ReSeT.setText("Create New Modified Lattice (Reset Based on Current Lattice)");
        ReSeT.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ReSeTActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(ReSeT, javax.swing.GroupLayout.DEFAULT_SIZE, 568,
Short.MAX_VALUE)
                .addContainerGap())
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addComponent(ReSeT)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        jPanel4.setBorder(javax.swing.BorderFactory.createTitledBorder("Set System Variables:"));

        SimName.setText("0");

        SimNumber.setText("0");

        jLabel6.setText("Sim Name:");

        jLabel7.setText("Sim Number:");

        Update_ACENET_Script.setText("Update ACENET Script");
        Update_ACENET_Script.addActionListener(new java.awt.event.ActionListener() {
```

```
        public void actionPerformed(java.awt.event.ActionEvent evt) {
          Update_ACENET_ScriptActionPerformed(evt);
        }
      });

      Email.setText("XX@MAIL.com");

      jLabel8.setText("E-Mail Address:");

      javax.swing.GroupLayout jPanel4Layout = new javax.swing.GroupLayout(jPanel4);
      jPanel4.setLayout(jPanel4Layout);
      jPanel4Layout.setHorizontalGroup(
        jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel4Layout.createSequentialGroup()
          .addContainerGap()

.addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
            .addComponent(jLabel8, javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel4Layout.createSequentialGroup()
              .addGap(12, 12, 12)
              .addComponent(Update_ACENET_Script, javax.swing.GroupLayout.DEFAULT_SIZE,
186, Short.MAX_VALUE))
            .addComponent(Email, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 198, Short.MAX_VALUE)
            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel4Layout.createSequentialGroup()

.addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel6)
                .addComponent(jLabel7))
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(SimName, javax.swing.GroupLayout.DEFAULT_SIZE, 102,
Short.MAX_VALUE)
```

```
                .addComponent(SimNumber, javax.swing.GroupLayout.PREFERRED_SIZE, 102,
javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addContainerGap())
    );
    jPanel4Layout.setVerticalGroup(
      jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel4Layout.createSequentialGroup()

.addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(SimName, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel6))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(SimNumber, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 19,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jLabel8)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(Email, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
          .addComponent(Update_ACENET_Script)
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );

    jPanel5.setBorder(javax.swing.BorderFactory.createTitledBorder("AceNet Script
Preview:"));

    ACENET_Script_Field.setColumns(20);
    ACENET_Script_Field.setRows(5);
    ACENET_Script_Field.setText("#$ -S /bin/bash\n#$ -cwd\n#$ -N <NAME>\n#$ -j y\n#$ -o
<NAME>.out\n#$ -l h_vmem=5120M\n#$ -l h_rt=720:00:00\n#$ -m eas\n#$ -M
<EMAIL>\n./Sim ./Input/Lat<NAinX>_<NAinY>_<GoldY>_<GoldX>_<NAME>.csv ./Output/o_
```

```
./b_ <SurfaceSpringConst> <MaxStretch> <NumberStretch> <NAinX> <NAinY> <AUinY>
<AUinX> <Recovery?>");
    jScrollPane1.setViewportView(ACENET_Script_Field);

    javax.swing.GroupLayout jPanel5Layout = new javax.swing.GroupLayout(jPanel5);
    jPanel5.setLayout(jPanel5Layout);
    jPanel5Layout.setHorizontalGroup(
      jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel5Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 568,
Short.MAX_VALUE)
        .addContainerGap())
    );
    jPanel5Layout.setVerticalGroup(
      jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel5Layout.createSequentialGroup()
        .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 303,
Short.MAX_VALUE)
        .addContainerGap())
    );

    jPanel6.setBorder(javax.swing.BorderFactory.createTitledBorder("File Creation
Progress:"));

    ProgressBar.setStringPainted(true);

    javax.swing.GroupLayout jPanel6Layout = new javax.swing.GroupLayout(jPanel6);
    jPanel6.setLayout(jPanel6Layout);
    jPanel6Layout.setHorizontalGroup(
      jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel6Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(ProgressBar, javax.swing.GroupLayout.DEFAULT_SIZE, 808,
Short.MAX_VALUE)
        .addContainerGap())
```

```
        );
        jPanel6Layout.setVerticalGroup(
            jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel6Layout.createSequentialGroup()
                .addComponent(ProgressBar, javax.swing.GroupLayout.PREFERRED_SIZE, 29,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        jPanel7.setBorder(javax.swing.BorderFactory.createTitledBorder("Lattice Properties:"));

        InputFileNameDisplay.setEditable(false);

        jLabel10.setText("Lattice File Name:");

        DisplayOutName.setEditable(false);

        jLabel17.setText("Output Lattice File:");

        VizOutputPic.setEditable(false);

        jLabel26.setText("Output File Name:");

        javax.swing.GroupLayout jPanel7Layout = new javax.swing.GroupLayout(jPanel7);
        jPanel7.setLayout(jPanel7Layout);
        jPanel7Layout.setHorizontalGroup(
            jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel7Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel10)
                    .addComponent(jLabel17)
                    .addComponent(jLabel26))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addComponent(VizOutputPic, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 435, Short.MAX_VALUE)
            .addComponent(DisplayOutName, javax.swing.GroupLayout.DEFAULT_SIZE, 435,
Short.MAX_VALUE)
            .addComponent(InputFileNameDisplay, javax.swing.GroupLayout.DEFAULT_SIZE,
435, Short.MAX_VALUE))
          .addContainerGap())
    );
    jPanel7Layout.setVerticalGroup(
      jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel7Layout.createSequentialGroup()

.addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(InputFileNameDisplay, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel10))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(DisplayOutName, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel17))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(VizOutputPic, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel26)))
    );

    jPanel8.setBorder(javax.swing.BorderFactory.createTitledBorder("Simulation Settings:"));

    jLabel14.setText("Surface Spring Constant (1e9):");

    jLabel15.setText("Maximum Stretch:");
```

```
        jLabel16.setText("Number of Steps to Maximum:");

        Recov.setText("Recovery?");

        SurSprConst.setText("0");

        MaxStretch.setText("0");

        NumSteps.setText("0");

        javax.swing.GroupLayout jPanel8Layout = new javax.swing.GroupLayout(jPanel8);
        jPanel8.setLayout(jPanel8Layout);
        jPanel8Layout.setHorizontalGroup(
            jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel8Layout.createSequentialGroup()

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel8Layout.createSequentialGroup()
                        .addContainerGap()

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel16)

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                                .addComponent(jLabel15)
                                .addComponent(jLabel14)))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(SurSprConst, javax.swing.GroupLayout.DEFAULT_SIZE, 104,
Short.MAX_VALUE)
                            .addComponent(MaxStretch, javax.swing.GroupLayout.DEFAULT_SIZE, 104,
Short.MAX_VALUE)
                            .addComponent(NumSteps, javax.swing.GroupLayout.DEFAULT_SIZE, 104,
Short.MAX_VALUE)))
                    .addGroup(jPanel8Layout.createSequentialGroup()
                        .addGap(83, 83, 83)
```

```
                .addComponent(Recov)))
            .addContainerGap())
        );
        jPanel8Layout.setVerticalGroup(
          jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(jPanel8Layout.createSequentialGroup()

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel14)
                .addComponent(SurSprConst, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel15)
                .addComponent(MaxStretch, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel8Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel16)
                .addComponent(NumSteps, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(Recov)
            .addGap(28, 28, 28))
        );

        jPanel9.setBorder(javax.swing.BorderFactory.createTitledBorder("Surface Sine Wave
Variation:"));

        jLabel18.setText("Amplitude:");

        jLabel19.setText("Frequency:");

        javax.swing.GroupLayout jPanel9Layout = new javax.swing.GroupLayout(jPanel9);
```

```
        jPanel9.setLayout(jPanel9Layout);
        jPanel9Layout.setHorizontalGroup(
            jPanel9Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel9Layout.createSequentialGroup()
                .addContainerGap(43, Short.MAX_VALUE)

.addGroup(jPanel9Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel18)
                    .addComponent(jLabel19))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel9Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(SurAmp, javax.swing.GroupLayout.DEFAULT_SIZE, 199,
Short.MAX_VALUE)
                    .addComponent(SurFreq, javax.swing.GroupLayout.DEFAULT_SIZE, 199,
Short.MAX_VALUE))
                .addContainerGap())
        );
        jPanel9Layout.setVerticalGroup(
            jPanel9Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel9Layout.createSequentialGroup()

.addGroup(jPanel9Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel18)
                    .addComponent(SurAmp, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel9Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel19)
                    .addComponent(SurFreq, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        jPanel10.setBorder(javax.swing.BorderFactory.createTitledBorder("Vary Substrate Surface
Roughness:"));
```

```
        jLabel22.setText("Amplitude:");

        jLabel23.setText("Frequency:");

        javax.swing.GroupLayout jPanel10Layout = new javax.swing.GroupLayout(jPanel10);
        jPanel10.setLayout(jPanel10Layout);
        jPanel10Layout.setHorizontalGroup(
            jPanel10Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel10Layout.createSequentialGroup()
                .addGap(45, 45, 45)

.addGroup(jPanel10Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel22)
                    .addComponent(jLabel23))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel10Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(SubAmp, javax.swing.GroupLayout.DEFAULT_SIZE, 197,
Short.MAX_VALUE)
                    .addComponent(SubFreq, javax.swing.GroupLayout.DEFAULT_SIZE, 197,
Short.MAX_VALUE))
                .addContainerGap())
        );
        jPanel10Layout.setVerticalGroup(
            jPanel10Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel10Layout.createSequentialGroup()

.addGroup(jPanel10Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel22)
                    .addComponent(SubAmp, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel10Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel23)
```

```
            .addComponent(SubFreq, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        jPanel11.setBorder(javax.swing.BorderFactory.createTitledBorder("Lattice Properties:"));

        jLabel9.setText("Silicon Atoms in X:");

        NAinX.setText("0");

        jLabel11.setText("Silicon Atoms in Y:");

        NAinY.setText("0");

        jLabel13.setText("Gold Atoms in Y:");

        AUinY.setText("0");

        AUinX.setText("0");

        jLabel12.setText("Gold Atoms in X:");

        MkNLat.setText("Make New Lattice");
        MkNLat.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                MkNLatActionPerformed(evt);
            }
        });

        jSeparator3.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        javax.swing.GroupLayout jPanel11Layout = new javax.swing.GroupLayout(jPanel11);
        jPanel11.setLayout(jPanel11Layout);
        jPanel11Layout.setHorizontalGroup(
            jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel11Layout.createSequentialGroup()
```

```
            .addContainerGap()

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel11Layout.createSequentialGroup()

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(jLabel11)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel12)
                            .addComponent(jLabel13)))
                    .addComponent(jLabel9))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(NAinX, javax.swing.GroupLayout.DEFAULT_SIZE, 67,
Short.MAX_VALUE)
                    .addComponent(AUinY, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 67, Short.MAX_VALUE)
                    .addComponent(NAinY, javax.swing.GroupLayout.DEFAULT_SIZE, 67,
Short.MAX_VALUE)
                    .addComponent(AUinX, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 67, Short.MAX_VALUE))
                .addContainerGap())
            .addComponent(jSeparator3, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 210, Short.MAX_VALUE))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel11Layout.createSequentialGroup()
            .addComponent(MkNLat)
            .addGap(54, 54, 54))))
    );
    jPanel11Layout.setVerticalGroup(
      jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGroup(jPanel11Layout.createSequentialGroup()

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel9)
            .addComponent(NAinX, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel11)
            .addComponent(NAinY, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel13)
            .addComponent(AUinY, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel11Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel12)
            .addComponent(AUinX, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jSeparator3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(MkNLat))
      );

      jPanel13.setBorder(javax.swing.BorderFactory.createTitledBorder("Visualize Output
Lattice:"));

      SelVizOut.setText("Select Output File");
      SelVizOut.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            SelVizOutActionPerformed(evt);
        }
    });

    jPanel15.setBorder(javax.swing.BorderFactory.createTitledBorder("Completed Run or
Backup?"));

    buttonGroup1.add(CompRun);
    CompRun.setSelected(true);
    CompRun.setText("Completed");

    buttonGroup1.add(VizBack);
    VizBack.setText("Backup");

    javax.swing.GroupLayout jPanel15Layout = new javax.swing.GroupLayout(jPanel15);
    jPanel15.setLayout(jPanel15Layout);
    jPanel15Layout.setHorizontalGroup(
        jPanel15Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel15Layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(CompRun)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(VizBack)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    jPanel15Layout.setVerticalGroup(
        jPanel15Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel15Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(CompRun)
            .addComponent(VizBack))
    );

    VizOutLat.setText("VisCurOutLat");
    VizOutLat.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            VizOutLatActionPerformed(evt);
```

```
        }
    });

    SavOutLatViz.setText("Save Output");
    SavOutLatViz.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            SavOutLatVizActionPerformed(evt);
        }
    });

    DisSubLine.setText("Display Substrate Line");

    DisSurLine.setText("Display Surface Line");

    ZoomGoldLat.setText("ZoomGold");
    ZoomGoldLat.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            ZoomGoldLatActionPerformed(evt);
        }
    });

    SavOutLatVizZoomGold.setText("Save Zoom Gold");
    SavOutLatVizZoomGold.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            SavOutLatVizZoomGoldActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanel13Layout = new javax.swing.GroupLayout(jPanel13);
    jPanel13.setLayout(jPanel13Layout);
    jPanel13Layout.setHorizontalGroup(
        jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel13Layout.createSequentialGroup()

.addGroup(jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
                .addGroup(jPanel13Layout.createSequentialGroup()
                  .addComponent(SelVizOut)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(VizOutLat, javax.swing.GroupLayout.PREFERRED_SIZE, 106,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(ZoomGoldLat, javax.swing.GroupLayout.PREFERRED_SIZE, 102,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel13Layout.createSequentialGroup()
                  .addContainerGap()
                  .addComponent(jPanel15, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(SavOutLatVizZoomGold,
javax.swing.GroupLayout.PREFERRED_SIZE, 125, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(SavOutLatViz, javax.swing.GroupLayout.PREFERRED_SIZE,
125, javax.swing.GroupLayout.PREFERRED_SIZE))))
              .addGroup(jPanel13Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(DisSubLine)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(DisSurLine)))
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        jPanel13Layout.setVerticalGroup(
          jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(jPanel13Layout.createSequentialGroup()

.addGroup(jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(SelVizOut)
              .addComponent(VizOutLat)
              .addComponent(ZoomGoldLat))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
.addGroup(jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel15, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGroup(jPanel13Layout.createSequentialGroup()
              .addComponent(SavOutLatViz, javax.swing.GroupLayout.PREFERRED_SIZE, 17,
javax.swing.GroupLayout.PREFERRED_SIZE)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
              .addComponent(SavOutLatVizZoomGold,
javax.swing.GroupLayout.PREFERRED_SIZE, 17, javax.swing.GroupLayout.PREFERRED_SIZE)))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel13Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(DisSubLine, javax.swing.GroupLayout.DEFAULT_SIZE, 28,
Short.MAX_VALUE)
            .addComponent(DisSurLine, javax.swing.GroupLayout.PREFERRED_SIZE, 16,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap())
    );

    jPanel14.setBorder(javax.swing.BorderFactory.createTitledBorder("Apply Roughness:"));

    AppSurRough.setText("Apply Surface Roughness");
    AppSurRough.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        AppSurRoughActionPerformed(evt);
      }
    });

    AppSubRough.setText("Apply Substrate Roughness");
    AppSubRough.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        AppSubRoughActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel14Layout = new javax.swing.GroupLayout(jPanel14);
```

```
        jPanel14.setLayout(jPanel14Layout);
        jPanel14Layout.setHorizontalGroup(
            jPanel14Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel14Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel14Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                    .addComponent(AppSurRough, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(AppSubRough, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                .addContainerGap(21, Short.MAX_VALUE))
        );
        jPanel14Layout.setVerticalGroup(
            jPanel14Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel14Layout.createSequentialGroup()
                .addComponent(AppSurRough)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(AppSubRough)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        jPanel17.setBorder(javax.swing.BorderFactory.createTitledBorder("Auto Run Options:"));

        RunOnLocal.setText("Run on Local");
        RunOnLocal.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                RunOnLocalActionPerformed(evt);
            }
        });

        RunOnOther.setText("Run on Other");
        RunOnOther.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
        RunOnOtherActionPerformed(evt);
      }
    });

    RunOnACENET.setText("Run on ACENET");
    RunOnACENET.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        RunOnACENETActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel17Layout = new javax.swing.GroupLayout(jPanel17);
    jPanel17.setLayout(jPanel17Layout);
    jPanel17Layout.setHorizontalGroup(
      jPanel17Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel17Layout.createSequentialGroup()
        .addContainerGap()

.addGroup(jPanel17Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(RunOnOther, javax.swing.GroupLayout.DEFAULT_SIZE, 139,
Short.MAX_VALUE)
          .addComponent(RunOnACENET, javax.swing.GroupLayout.DEFAULT_SIZE, 139,
Short.MAX_VALUE)
          .addComponent(RunOnLocal, javax.swing.GroupLayout.DEFAULT_SIZE, 139,
Short.MAX_VALUE))
        .addContainerGap())
    );
    jPanel17Layout.setVerticalGroup(
      jPanel17Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel17Layout.createSequentialGroup()
        .addGap(9, 9, 9)
        .addComponent(RunOnLocal)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(RunOnACENET)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(RunOnOther)
        .addGap(20, 20, 20))
```

```
        );

        jPanel16.setBorder(javax.swing.BorderFactory.createTitledBorder("Select Colours:"));

        jLabel27.setText("Silicon:");

        jLabel28.setText("Gold:");

        ChangeSilColour.setText("Change");
        ChangeSilColour.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ChangeSilColourActionPerformed(evt);
            }
        });

        ChangeGoldColour.setText("Change");
        ChangeGoldColour.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ChangeGoldColourActionPerformed(evt);
            }
        });


SiliconColorField.setBorder(javax.swing.BorderFactory.createEtchedBorder(javax.swing.border.
EtchedBorder.RAISED));

        javax.swing.GroupLayout SiliconColorFieldLayout = new
javax.swing.GroupLayout(SiliconColorField);
        SiliconColorField.setLayout(SiliconColorFieldLayout);
        SiliconColorFieldLayout.setHorizontalGroup(

SiliconColorFieldLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 133, Short.MAX_VALUE)
        );
        SiliconColorFieldLayout.setVerticalGroup(

SiliconColorFieldLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGap(0, 29, Short.MAX_VALUE)
    );


GoldColorField.setBorder(javax.swing.BorderFactory.createEtchedBorder(javax.swing.border.Et
chedBorder.RAISED));

    javax.swing.GroupLayout GoldColorFieldLayout = new
javax.swing.GroupLayout(GoldColorField);
    GoldColorField.setLayout(GoldColorFieldLayout);
    GoldColorFieldLayout.setHorizontalGroup(

GoldColorFieldLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 133, Short.MAX_VALUE)
    );
    GoldColorFieldLayout.setVerticalGroup(

GoldColorFieldLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 29, Short.MAX_VALUE)
    );

    javax.swing.GroupLayout jPanel16Layout = new javax.swing.GroupLayout(jPanel16);
    jPanel16.setLayout(jPanel16Layout);
    jPanel16Layout.setHorizontalGroup(
      jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel16Layout.createSequentialGroup()
        .addContainerGap()

.addGroup(jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
            .addComponent(jLabel27)
            .addComponent(jLabel28))
        .addGap(18, 18, 18)

.addGroup(jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
            .addComponent(ChangeGoldColour, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
            .addComponent(ChangeSilColour, javax.swing.GroupLayout.DEFAULT_SIZE, 90,
Short.MAX_VALUE))
          .addGap(6, 6, 6)

.addGroup(jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(SiliconColorField, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(GoldColorField, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
          .addGap(30, 30, 30))
      );
      jPanel16Layout.setVerticalGroup(
        jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel16Layout.createSequentialGroup()

.addGroup(jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel16Layout.createSequentialGroup()
              .addComponent(SiliconColorField, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addGap(4, 4, 4))
            .addComponent(ChangeSilColour, javax.swing.GroupLayout.DEFAULT_SIZE, 37,
Short.MAX_VALUE)
            .addComponent(jLabel27))
          .addGap(10, 10, 10)

.addGroup(jPanel16Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
            .addComponent(GoldColorField, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(ChangeGoldColour, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(jLabel28))
          .addContainerGap())
      );

      jPanel3.setBorder(javax.swing.BorderFactory.createTitledBorder("UserName &
Password:"));
```

```
        jLabel1.setText("User:");

        jLabel2.setText("Password:");

        jLabel3.setText("Server:");

        buttonGroup3.add(SACE);
        SACE.setSelected(true);
        SACE.setText("ACENET");

        buttonGroup3.add(BSERV);
        BSERV.setText("Baskes");

        javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(
            jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel3Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel3)
                    .addComponent(jLabel1)
                    .addComponent(jLabel2))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(BSERV)
                    .addComponent(UPASS, javax.swing.GroupLayout.DEFAULT_SIZE, 151,
Short.MAX_VALUE)
                    .addComponent(UNAME, javax.swing.GroupLayout.DEFAULT_SIZE, 151,
Short.MAX_VALUE)
                    .addComponent(SACE))
                .addContainerGap())
        );
        jPanel3Layout.setVerticalGroup(
```

```
        jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel3Layout.createSequentialGroup()
          .addContainerGap()

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
            .addComponent(UNAME, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGap(18, 18, 18)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
            .addComponent(UPASS, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel3)
            .addComponent(SACE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
          .addComponent(BSERV))
      );

      SaveCurLat.setText("Save Current Lattice to Selected File");
      SaveCurLat.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
          SaveCurLatActionPerformed(evt);
        }
      });

      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addContainerGap()
```

```
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                  .addComponent(jPanel8, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                  .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addComponent(jPanel9, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                  .addComponent(jPanel10, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                  .addComponent(jPanel16, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
              .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addComponent(SaveCurLat)
                .addGap(57, 57, 57)))
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                  .addComponent(jPanel14, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                  .addComponent(jPanel4, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                  .addComponent(jPanel11, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                  .addComponent(jPanel7, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
                .addComponent(jPanel5, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)))
            .addGroup(layout.createSequentialGroup()
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                  .addComponent(jPanel13, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(jPanel3, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(jPanel17, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addGap(12, 12, 12))
                .addComponent(jPanel6, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
            .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addComponent(jPanel10, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
```

```
                        .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jPanel8, 0, 148, Short.MAX_VALUE))
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                        .addComponent(jPanel4, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jPanel11, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanel9, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jPanel14, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jPanel5, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jPanel7, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGap(2, 2, 2)
            .addComponent(jPanel6, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(jPanel3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
            .addComponent(jPanel17, javax.swing.GroupLayout.PREFERRED_SIZE, 167,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                .addComponent(jPanel13, javax.swing.GroupLayout.Alignment.LEADING, 0, 154,
Short.MAX_VALUE)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                    .addComponent(jPanel16, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(SaveCurLat))))
            .addContainerGap())
    );

    pack();
  }// </editor-fold>//GEN-END:initComponents

  private void Update_ACENET_ScriptActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_Update_ACENET_ScriptActionPerformed
    ACENET_Script_Field.setText("#$ -S /bin/bash\n#$ -cwd\n#$ -N
"+SimName.getText()+"\n#$ -j y\n#$ -o "+SimName.getText()+".out\n#$ -l
h_vmem=5000M\n#$ -pe openmp 8\n#$ -l h_rt=720:00:00\n#$ -m eas\n#$ -M
"+Email.getText()+"\nexport OMP_NUM_THREADS=$NSLOTS\n./Sim
~/scratch/Sim"+SimNumber.getText()+"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY.
getText()+"_"+AUinX.getText()+"_"+SimName.getText()+".csv
~/scratch/Sim"+SimNumber.getText()+"/o_ ./b_ "+SurSprConst.getText()+"
"+MaxStretch.getText()+" "+NumSteps.getText()+" "+NAinX.getText()+" "+NAinY.getText()+"
"+AUinY.getText()+" "+AUinX.getText()+"\n");

    if(Recov.isSelected()){
      ACENET_Script_Field.setText("#$ -S /bin/bash\n#$ -cwd\n#$ -N
"+SimName.getText()+"\n#$ -j y\n#$ -o "+SimName.getText()+".out\n#$ -l
h_vmem=2970M\n#$ -pe openmp 4\n#$ -l h_rt=720:00:00\n#$ -m eas\n#$ -M
"+Email.getText()+"\nexport OMP_NUM_THREADS=$NSLOTS\n./Sim
~/scratch/Sim"+SimNumber.getText()+"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY.
getText()+"_"+AUinX.getText()+"_"+SimName.getText()+".csv
```

```
~/scratch/Sim"+SimNumber.getText()+"/o_ ./b_ "+SurSprConst.getText()+"
"+MaxStretch.getText()+" "+NumSteps.getText()+" "+NAinX.getText()+" "+NAinY.getText()+"
"+AUinY.getText()+" "+AUinX.getText()+" t\n");
    }
  }//GEN-LAST:event_Update_ACENET_ScriptActionPerformed

  private void QuiTActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_QuiTActionPerformed
    System.exit(0);
  }//GEN-LAST:event_QuiTActionPerformed

  private void OpenLatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_OpenLatActionPerformed
    int returnVal = OpenLatticeFile.showOpenDialog(this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
    InputFileNameDisplay.setText(OpenLatticeFile.getSelectedFile().getName());
    OpenLatCheck.setSelected(true);
    String str = OpenLatticeFile.getSelectedFile().getName();
    String delimiters = "[ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz._]+";
    String[] tokens = str.split(delimiters);
    NAinX.setText(tokens[1]);
    NAinY.setText(tokens[2]);
    AUinY.setText(tokens[3]);
    AUinX.setText(tokens[4]);
    try{
    String [] command = new String[]{"mkdir","./Scratch/Sim"+SimNumber.getText()};
    Process chi = Runtime.getRuntime().exec(command);
    command = new String[]{"cp", OpenLatticeFile.getSelectedFile().toString(),
"./Scratch/Sim"+SimNumber.getText()+"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY
.getText()+"_"+AUinX.getText()+".csv"};
    chi = Runtime.getRuntime().exec(command);
    } catch (Exception e){

    }
    OpenLatticeFile.setSelectedFile(new
File(System.getProperty("user.dir")+"/Scratch/Sim"+SimNumber.getText()+
```

```
"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY.getText()+"_"+AUinX.getText()+".csv"))
;
      InputFileNameDisplay.setText(OpenLatticeFile.getSelectedFile().getName());
      OpenLatCheck.setSelected(true);
      ReadIn();
    } else {
      ProgressBar.setString("File access cancelled by user.");
    }
    }//GEN-LAST:event_OpenLatActionPerformed

    private void MkANscrActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_MkANscrActionPerformed

      if(SaveLatCheck.isSelected()){
        try{
          WriteScriptFile();
        }
        catch (IOException ex) {
          ProgressBar.setString("Cannot Write Script File");
        }
      } else {
        ProgressBar.setString("Must Select Save Location...");
      }

    }//GEN-LAST:event_MkANscrActionPerformed

    private void SelSaveLocActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_SelSaveLocActionPerformed
      int returnVal = SAFileChooser.showSaveDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
      SaveLatCheck.setSelected(true);
    } else {
      ProgressBar.setString("File access cancelled by user.");
    }
    }//GEN-LAST:event_SelSaveLocActionPerformed
```

```
    private void ReSeTActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_ReSeTActionPerformed
        SurSprConst.setText("1e9");
        MaxStretch.setText("0");
        NumSteps.setText("0");
        Recov.setSelected(false);
        SurAmp.setText("0");
        SurFreq.setText("0");
        SubAmp.setText("0");
        SubFreq.setText("0");
        SimName.setText("<NAME>");
        SimNumber.setText("<Number>");
        SiliconColorField.setBackground(SiliconColor);
        GoldColorField.setBackground(GoldColor);
        if(Lattice.length > 1){
          ReadIn();
        }
}//GEN-LAST:event_ReSeTActionPerformed
    private void IniSetup(){
        ReSeT.doClick();
        NAinX.setText("0");
        NAinY.setText("0");
        AUinY.setText("0");
        AUinX.setText("0");
        VizOutputPic.setText("Select Output File");
        InputFileNameDisplay.setText("Unmodified Lattice Name");
        DisplayOutName.setText("Modified Lattice Name");
        Email.setText("<E-Mail>");
    }
    private void MkNLatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_MkNLatActionPerformed
        String[] command = new String[]{};
        try {
          if(Integer.parseInt(AUinY.getText())>0){
            double layer =
Double.parseDouble(NAinY.getText())+Double.parseDouble(AUinY.getText());
```

```
            command = new String[]{"./GeneratorAUSI" ,NAinX.getText() , Double.toString(layer) ,
AUinY.getText() , "Lat.csv"};
            Process chi = Runtime.getRuntime().exec(command);
            InputStream in = chi.getInputStream();
            String CommandOut = convertStreamToString(in);
            String [] Oline = CommandOut.split(",");
            Oline[4] = Oline[4].replaceAll("\\n", "");
            AUinX.setText(Oline[4]);
            command = new String[]{"mkdir","./Scratch/Sim"+SimNumber.getText()};
            chi = Runtime.getRuntime().exec(command);
            command = new String[]{"mv", "./Lat.csv",
"./Scratch/Sim"+SimNumber.getText()+"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY
.getText()+"_"+AUinX.getText()+".csv"};
            chi = Runtime.getRuntime().exec(command);
            OpenLatticeFile.setSelectedFile(new
File(System.getProperty("user.dir")+"/Scratch/Sim"+SimNumber.getText()+
"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY.getText()+"_"+AUinX.getText()+".csv"))
;
            InputFileNameDisplay.setText(OpenLatticeFile.getSelectedFile().getName());
            OpenLatCheck.setSelected(true);

         } else{
            command = new String[]{"./Generator" ,NAinX.getText() , NAinY.getText() , "Lat.csv"};
            Process chi = Runtime.getRuntime().exec(command);
            command = new String[]{"mv", "./Lat.csv", SAFileChooser.getSelectedFile()+
"/Lat"+NAinX.getText()+"_"+NAinY.getText()+".csv"};
            chi = Runtime.getRuntime().exec(command);
            OpenLatticeFile.setSelectedFile(new File(SAFileChooser.getSelectedFile()+
"/Lat"+NAinX.getText()+"_"+NAinY.getText()+".csv"));
            InputFileNameDisplay.setText(OpenLatticeFile.getSelectedFile().getName());
            OpenLatCheck.setSelected(true);
         }

      } catch (IOException e) {
         ProgressBar.setString("Error in Lattice Creation");
      }
      ReadIn();
```

```
  }//GEN-LAST:event_MkNLatActionPerformed

  private void VizInLatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_VizInLatActionPerformed
    demo = new LatticePlot("Current
Lattice",Lattice,getGoldinX(),getGoldinY(),getSilinX(),getSilinY(),
        SiliconColor,GoldColor, DisSurLine.isSelected(),DisSubLine.isSelected(),getSurAmp(),
        getSurFreq(),getSubAmp(),getSubFreq(),GoldSurfaceHeight,GoldLength,false);
  }//GEN-LAST:event_VizInLatActionPerformed

  private void AppSurRoughActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_AppSurRoughActionPerformed
    if (Lattice.length < 1 ){
      ProgressBar.setString("Please Enter Lattice From File or Make New");
      return;
    }
    SurfaceCoSineRemover();
  }//GEN-LAST:event_AppSurRoughActionPerformed

  private void AppSubRoughActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_AppSubRoughActionPerformed
    if (Lattice.length < 1 ){
      ProgressBar.setString("Please Enter Lattice From File or Make New");
      return;
    }
    SubstrateCoSineRemover();
  }//GEN-LAST:event_AppSubRoughActionPerformed

  private void SelVizOutActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_SelVizOutActionPerformed
    int returnVal = VizOutFile.showOpenDialog(this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {

  } else {
    ProgressBar.setString("File access cancelled by user.");
```

```
    }
    }//GEN-LAST:event_SelVizOutActionPerformed

    private void ChangeSilColourActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_ChangeSilColourActionPerformed
        try{
        SiliconColor = JColorChooser.showDialog(this,"Choose Desired Silicon Colour",
SiliconColor);
        }catch(Exception e){
            ProgressBar.setString("Cannot Change Colour");
        }
        SiliconColorField.setBackground(SiliconColor);
    }//GEN-LAST:event_ChangeSilColourActionPerformed

    private void ChangeGoldColourActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_ChangeGoldColourActionPerformed
        try{
        GoldColor = JColorChooser.showDialog(this,"Choose Desired Silicon Colour", GoldColor);
        }catch(Exception e){
            ProgressBar.setString("Cannot Change Colour");
        }
        GoldColorField.setBackground(GoldColor);
    }//GEN-LAST:event_ChangeGoldColourActionPerformed

    private void VizOutLatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_VizOutLatActionPerformed
        ReadInFromBackup();
        demo = new LatticePlot("Current
Lattice",Lattice,getGoldinX(),getGoldinY(),getSilinX(),getSilinY(),
            SiliconColor,GoldColor, DisSurLine.isSelected(),DisSubLine.isSelected(),getSurAmp(),
            getSurFreq(),getSubAmp(),getSubFreq(),GoldSurfaceHeight,GoldLength,false);
    }//GEN-LAST:event_VizOutLatActionPerformed

    private void SavOutLatVizActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_SavOutLatVizActionPerformed
        int returnVal = SaveVizOut.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
```

```
    try {
      ChartUtilities.saveChartAsJPEG(SaveVizOut.getSelectedFile(), LatticePlot.getchart(),
1000,600);
            } catch (Exception e) {
              ProgressBar.setString("Problem occurred creating chart.");
            }
  } else {
    ProgressBar.setString("File access cancelled by user.");
  }
  }//GEN-LAST:event_SavOutLatVizActionPerformed

  private void RunOnLocalActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_RunOnLocalActionPerformed
    // TODO add your handling code here:
  }//GEN-LAST:event_RunOnLocalActionPerformed

  private void RunOnACENETActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_RunOnACENETActionPerformed
    if(SACE.isSelected()){
        Update_ACENET_Script.doClick();
        FixFileIO();
        WriteTransferPrepScript("acenet");
        try{
          WriteScriptFile();
          WriteUpLoadScript();
          WriteSubJob(ACENETServer);
          String [] command = new String[]{"sh" ,"SimTrans"+SimNumber.getText()+".sh"};
          Process chi = Runtime.getRuntime().exec(command);
        } catch (Exception e){

        }
    }else{return;}

  }//GEN-LAST:event_RunOnACENETActionPerformed
  private void WriteSubJob(String Serv){
    try{
```

```
        Writer out = new OutputStreamWriter(new
FileOutputStream(".//SimSub"+SimNumber.getText()+".exp"));
      try {
        if(SACE.isSelected()){
        out.write("#!/usr/bin/expect -f\n");
        out.write("set timeout -1\n");
        out.write("spawn ssh "+UNAME.getText()+"@"+Serv+" \"\n");
        out.write("unzip Sim"+SimNumber.getText()+".zip;\n");
        out.write("rm Sim"+SimNumber.getText()+".zip;\n");
        out.write("cd Sim"+SimNumber.getText()+";\n");
        out.write("gcc4 -std=c99 -ggdb -msse2 -DHAVE_SSE2 -DMEXP=19937 -lm -fopenmp -o
Sim *.c;\n");
        out.write("mv ./Sim ./Compiled/Sim;\n");
        out.write("cd ~/scratch/;\n");
        out.write("mkdir Sim"+SimNumber.getText()+";\n");
        out.write("cd -;\n");
        out.write("cd Compiled/;\n");
        out.write("mv ./Input/* ~/scratch/Sim"+SimNumber.getText()+"/;\n");
        out.write("qsub Subm.sh;\n");
        out.write("\"\n");
        out.write("expect {\n");
        out.write("      password: {send \""+new String(UPASS.getPassword())+"\\r\" ;
exp_continue}\n");
        out.write("      eof exit\n");
        out.write("}\n");
        } else if(BSERV.isSelected()){
          String C = "./Sim
./Input/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY.getText()+"_"+AUinX.getText()+"
_"+SimName.getText()+".csv ./Output/o_ ./b_ "+SurSprConst.getText()+"
"+MaxStretch.getText()+" "+NumSteps.getText()+" "+NAinX.getText()+" "+NAinY.getText()+"
"+AUinY.getText()+" "+AUinX.getText()+";";
          if(Recov.isSelected()){
            C += " t";
          }
          out.write("#!/usr/bin/expect -f\n");
          out.write("spawn ssh "+UNAME.getText()+"@"+Serv+" \"\n");
          out.write("cd Sim"+SimNumber.getText()+";\n");
```

```
            out.write("gcc -std=c99 -ggdb -lm -o Sim *.c -march=core2 -mtune=core2;\n");
            out.write("mv ./Sim ./Compiled/Sim;\n");
            out.write("cd Compiled/;\n");
            out.write("screen -tdm 'Sim"+SimNumber.getText()+"' bash -c 'gdb -ex run --args
"+C+"';\n");
            out.write(C);
            out.write("\"\n");
            out.write("expect {\n");
            out.write("       password: {send \""+new String(UPASS.getPassword())+"\\r\" ;
exp_continue}\n");
            out.write("       eof exit\n");
            out.write("}\n");
          }
        }
        finally {
          out.close();
        }
        } catch (IOException e){

        }
    }
    private void FixFileIO(){
        try{
        FileReader fr = new FileReader("./CFiles/ACENETFileInOut.c");
        FileWriter writer = new FileWriter("./CFiles/acenet/FileInOut.c");
        BufferedReader br = new BufferedReader(fr);
        String s;
        String[] splitString;
        while((s = br.readLine()) != null) {
          try{
          splitString = s.split("<UNAME>");
          s = splitString[0] + UNAME.getText() + splitString[1];
          splitString = s.split("<SimName>");
          s = splitString[0] + "Sim" + SimNumber.getText() + splitString[1];
          } catch(Exception e){

          }
```

```
            writer.append(s);
            writer.append('\n');
        }
        writer.flush();
        writer.close();
        fr.close();
        } catch (IOException e){
            ProgressBar.setString("Problem in FileInOut.c or FixFileIO");
        }
    }
    private void WriteTransferPrepScript(String Serv){
        try{//
        Writer out = new OutputStreamWriter(new
FileOutputStream(".//SimTrans"+SimNumber.getText()+".sh"));
        try {
        out.write("#!/bin/bash\n");
        out.write("cd CFiles/"+Serv+"/\n");
        out.write("cp *.c ../../Scratch/Sim"+SimNumber.getText()+"/\n");
        out.write("cp *.h ../../Scratch/Sim"+SimNumber.getText()+"/\n");
        out.write("cd ../..\n");
        out.write("cd Scratch/Sim"+SimNumber.getText()+"/\n");
        out.write("mkdir Sim"+SimNumber.getText()+"\n");
        out.write("mv *.c ./Sim"+SimNumber.getText()+"/\n");
        out.write("mv *.h ./Sim"+SimNumber.getText()+"/\n");
        out.write("cd Sim"+SimNumber.getText()+"/\n");
        out.write("mkdir Compiled\n");
        out.write("cd Compiled/\n");
        out.write("mkdir Input\n");
        out.write("mkdir Output\n");
        out.write("cd ../..\n");
        out.write("mv ./"+SAFileChooser.getSelectedFile().getName()+"
./Sim"+SimNumber.getText()+"/Compiled/Input/"+SAFileChooser.getSelectedFile().getName()+
"\n");
        out.write("cd ~/NetBeansProjects/SurfaceRougher/Scratch/\n");
        if(Serv.compareTo("acenet")==0){
            out.write("mv ./Subm.sh
./Sim"+SimNumber.getText()+"/Sim"+SimNumber.getText()+"/Compiled/Subm.sh\n");
```

```
        }
      out.write("cd Sim"+SimNumber.getText()+"/\n");
      out.write("zip -9 Sim"+SimNumber.getText()+".zip Sim"+SimNumber.getText()+"/\n");
      out.write("zip -9 Sim"+SimNumber.getText()+".zip Sim"+SimNumber.getText()+"/*\n");
      out.write("zip -9 Sim"+SimNumber.getText()+".zip Sim"+SimNumber.getText()+"/*/*\n");
      out.write("zip -9 Sim"+SimNumber.getText()+".zip Sim"+SimNumber.getText()+"/*/*/*\n");
      out.write("cd ~/NetBeansProjects/SurfaceRougher/\n");
      out.write("expect SimUp"+SimNumber.getText()+".exp\n");
      out.write("expect SimSub"+SimNumber.getText()+".exp\n");
      //out.write("rm -r *.exp\n");
      //out.write("rm -r *.exp*\n");
      }
      finally {
      out.close();
      }
      } catch (IOException e){

      }

   }
   private void WriteUpLoadScript(){
      try{
         Writer out = new OutputStreamWriter(new
FileOutputStream(".//SimUp"+SimNumber.getText()+".exp"));
      try {
         String Serv;
         if(SACE.isSelected()){
            Serv = ACENETServer;
         } else {
            Serv = LocalServer;
         }
      out.write("#!/usr/bin/expect -f\n");
      out.write("set timeout -1\n");
      out.write("spawn scp -C
Scratch/Sim"+SimNumber.getText()+"/Sim"+SimNumber.getText()+".zip
"+UNAME.getText()+"@"+Serv+":~/\n");
      out.write("expect {\n");
```

```
    out.write("      password: {send \""+ new String(UPASS.getPassword())+"\\r\" ;
exp_continue}\n");
    out.write("      eof exit\n");
    out.write("}");

    }
    finally {
      out.close();
    }
    } catch (IOException e){

    }
  }
  private void RunOnOtherActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_RunOnOtherActionPerformed
    if(BSERV.isSelected()){
        FixFileIO();
        WriteTransferPrepScript("Baskes");
        try{
          String [] command = new String[]{"sh" ,"SimTrans"+SimNumber.getText()+".sh"};
          Process chi = Runtime.getRuntime().exec(command);
          command = new String[]{"rm","SimTrans"+SimNumber.getText()+".sh"};
          chi = Runtime.getRuntime().exec(command);
          WriteUpLoadScript();
          command = new String[]{"expect" ,"SimUp"+SimNumber.getText()+".exp"};
          chi = Runtime.getRuntime().exec(command);
          command = new String[]{"rm","SimUp"+SimNumber.getText()+".exp"};
          chi = Runtime.getRuntime().exec(command);
          WriteSubJob(LocalServer);
          command = new String[]{"expect" ,"SimSub"+SimNumber.getText()+".exp"};
          chi = Runtime.getRuntime().exec(command);
          command = new String[]{"rm","SimSub"+SimNumber.getText()+".exp"};
          chi = Runtime.getRuntime().exec(command);
        } catch (Exception e){

        }
    }else{return;}
```

```
    }//GEN-LAST:event_RunOnOtherActionPerformed

    private void SaveCurLatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_SaveCurLatActionPerformed
        try{
          FileWriter writer;
          int total =
Integer.parseInt(AUinX.getText())*Integer.parseInt(AUinY.getText())+Integer.parseInt(NAinX.get
Text())*Integer.parseInt(NAinY.getText());
          if(SaveLatCheck.isSelected()){
           writer = new FileWriter(SAFileChooser.getSelectedFile());
          } else {
            SAFileChooser.setSelectedFile(new
File(System.getProperty("user.dir")+"/Scratch/Sim"+SimNumber.getText()+
"/Lat"+NAinX.getText()+"_"+NAinY.getText()+"_"+AUinY.getText()+"_"+AUinX.getText()+"_"+Si
mName.getText()+".csv"));
            writer = new FileWriter(SAFileChooser.getSelectedFile());
          }

          for(int i = 0; i < total; i++){
            writer.append(Lattice[i].getInStr());
            writer.append(',');
            writer.append(Lattice[i].getXStr());
            writer.append(',');
            writer.append(Lattice[i].getYStr());
            writer.append(',');
            writer.append(Lattice[i].getBSstr());
            writer.append(',');
            writer.append(Lattice[i].getSGstr());
            writer.append(',');
            writer.append(Lattice[i].getRV());
            writer.append('\n');
            ProgressBar.setValue((int)(100.0*i/total));
            ProgressBar.setStringPainted(true);
          }
            writer.flush();
            writer.close();
```

```
            ProgressBar.setValue(100);
            ProgressBar.setStringPainted(true);
          }catch (Exception e){
            System.err.println("Error: " + e.getMessage());
          }
    }//GEN-LAST:event_SaveCurLatActionPerformed

    private void SavOutLatVizZoomGoldActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_SavOutLatVizZoomGoldActionPerformed
      int returnVal = SaveVizOut.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
      try {
        ChartUtilities.saveChartAsJPEG(SaveVizOut.getSelectedFile(), LatticePlot.getchart(),
1000,600);
              } catch (Exception e) {
                ProgressBar.setString("Problem occurred creating chart.");
              }
    } else {
      ProgressBar.setString("File access cancelled by user.");
    }
    }//GEN-LAST:event_SavOutLatVizZoomGoldActionPerformed

    private void ZoomGoldLatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_ZoomGoldLatActionPerformed
      ReadInFromBackup();
      ZoomGoldLatPlot = new LatticePlot("Current
Lattice",Lattice,getGoldinX(),getGoldinY(),getSilinX(),getSilinY(),
          SiliconColor,GoldColor, DisSurLine.isSelected(),DisSubLine.isSelected(),getSurAmp(),
          getSurFreq(),getSubAmp(),getSubFreq(),GoldSurfaceHeight,GoldLength,true);
    }//GEN-LAST:event_ZoomGoldLatActionPerformed
private void ReadInFromBackup(){
      int i = 0;
      try {
        CSVReader reader = new CSVReader(new FileReader(VizOutFile.getSelectedFile()));
        String [] nextLine;
          if(VizBack.isSelected()){
          try {
```

```
            while ((nextLine = reader.readNext()) != null) {
              Lattice[i].setX(Double.parseDouble(nextLine[1]));
              Lattice[i].setY(-1.0 * Double.parseDouble(nextLine[0]));
              i++;
            }
          } catch (IOException ex) {
            ProgressBar.setString("File access error.");

          }
        }else if (CompRun.isSelected()){
          try {

            while ((nextLine = reader.readNext()) != null) {
              Lattice[i].setX(Double.parseDouble(nextLine[1]));
              Lattice[i].setY(Double.parseDouble(nextLine[2]));
              i++;
            }
          } catch (IOException ex) {
            ProgressBar.setString("File access error.");

          }
        }else{ProgressBar.setString("Error: ReadInFromBackup Viz Radio group not working");}
      } catch (FileNotFoundException e) {
      }
    }
    private void ReadIn(){
      int total =
Integer.parseInt(AUinX.getText())*Integer.parseInt(AUinY.getText())+Integer.parseInt(NAinX.get
Text())*Integer.parseInt(NAinY.getText());
      System.gc();
      Lattice = new Atom[total];
      int i = 0;
      try {
        CSVReader reader = new CSVReader(new FileReader(OpenLatticeFile.getSelectedFile()));
        String [] nextLine;
          try {
```

```
            while ((nextLine = reader.readNext()) != null) {
              Lattice[i]=new
Atom(Integer.parseInt(nextLine[0]),Double.parseDouble(nextLine[1]),Double.parseDouble(next
Line[2]),nextLine[3].charAt(0),nextLine[4].charAt(0),nextLine[5].charAt(0));
              if((Double.parseDouble(nextLine[1])>GoldLength) && (nextLine[4].charAt(0) ==
'G')){
                GoldLength = Double.parseDouble(nextLine[1]);
              }
              if((Double.parseDouble(nextLine[2])>GoldSurfaceHeight)&&
(nextLine[4].charAt(0) == 'G')){
                GoldSurfaceHeight = Double.parseDouble(nextLine[2]);
                Lattice[i-1].setBound();
                Lattice[i].setBound();
              }
              if((Double.parseDouble(nextLine[1])>SiliconLength) && (nextLine[4].charAt(0) ==
'S')){
                SiliconLength = Double.parseDouble(nextLine[1]);
              }
              if((Double.parseDouble(nextLine[2])>SiliconSubHeight)&& (nextLine[4].charAt(0)
== 'S')){
                SiliconSubHeight = Double.parseDouble(nextLine[2]);
                Lattice[i-1].setBound();
                Lattice[i].setBound();
              }
              i++;
            }
          } catch (IOException ex) {
            ProgressBar.setString("File access error.");

          }

      } catch (FileNotFoundException e) {
      }
      Lattice[0].setBound();
      Lattice[i-1].setBound();
    }
```

```
private static <T> T[] concat(T[] a, T[] b) {
final int alen = a.length;
final int blen = b.length;
if (alen == 0) {
  return b;
}
if (blen == 0) {
  return a;
}
final T[] result = (T[]) java.lang.reflect.Array.
     newInstance(a.getClass().getComponentType(), alen + blen);
System.arraycopy(a, 0, result, 0, alen);
System.arraycopy(b, 0, result, alen, blen);
return result;
}

private void SurfaceCoSineRemover(){
  int total = getGoldinX() * getGoldinY() + getSilinX() * getSilinY();
  for(int i = 0; i < total; i++){
    ProgressBar.setValue((int)(100.0*i/total));
    ProgressBar.setStringPainted(true);
    if(Lattice[i].getY()>Fun(Lattice[i].getX())){
      if(Lattice[i].getBound()){
        continue;
      }
      Lattice[i].setRV('v');
    }
  }
  ProgressBar.setValue(100);
  ProgressBar.setStringPainted(true);

}
public double Fun(double x){
  return
0.5*Double.parseDouble(SurAmp.getText())*Math.cos(2*Math.PI*x*Double.parseDouble(SurFr
eq.getText())/GoldLength)+GoldSurfaceHeight-Double.parseDouble(SurAmp.getText())/2.0;
```

```
    }
    private void SubstrateCoSineRemover(){//fix boundaries and mixing
      int total = getGoldinX() * getGoldinY() + getSilinX() * getSilinY();
      for(int i = 0; i < total; i++){
        if(Lattice[i].getSG() == 'S'){
          continue;
        }
        if(Lattice[i].getSG() == 'G'){
          Lattice[i].setY(Lattice[i].getY()-Double.parseDouble(SubAmp.getText()));
        }
      }
      for(int i = 0; i < total; i++){
        ProgressBar.setValue((int)(100.0*i/total));
        ProgressBar.setStringPainted(true);
        if(Lattice[i].getY()>FunSub(Lattice[i].getX())&&Lattice[i].getSG() == 'S'){
          if(Lattice[i].getBound()){
            continue;
          }
          Lattice[i].setRV('v');
        }
        if(Lattice[i].getY()<FunSub(Lattice[i].getX())&& Lattice[i].getSG() == 'G'){
          if(Lattice[i].getBound()){
            continue;
          }
          Lattice[i].setRV('v');
        }
      }
      ProgressBar.setValue(100);
      ProgressBar.setStringPainted(true);
    }
    public double FunSub(double x){
      return
0.5*Double.parseDouble(SubAmp.getText())*Math.cos(2*Math.PI*x*Double.parseDouble(SubF
req.getText())/SiliconLength)+SiliconSubHeight-Double.parseDouble(SubAmp.getText())/2.0;
    }
    private void WriteScriptFile() throws IOException{
      String filename;
```

```
    if(SaveLatCheck.isSelected()){
      filename = SAFileChooser.getSelectedFile() + "/Subm.sh";
    }else {
      filename = ".//Scratch/Subm.sh";
    }
    Writer out = new OutputStreamWriter(new FileOutputStream(filename));
    try {
    out.write(ACENET_Script_Field.getText());
    }
    finally {
    out.close();
    }

}
public String convertStreamToString(InputStream is) throws IOException {
    /*
     * To convert the InputStream to String we use the
     * Reader.read(char[] buffer) method. We iterate until the
     * Reader return -1 which means there's no more data to
     * read. We use the StringWriter class to produce the string.
     */
    if (is != null) {
      Writer writer = new StringWriter();

      char[] buffer = new char[1024];
      try {
        Reader reader = new BufferedReader(
            new InputStreamReader(is, "UTF-8"));
        int n;
        while ((n = reader.read(buffer)) != -1) {
          writer.write(buffer, 0, n);
        }
      } finally {
        is.close();
      }
      return writer.toString();
    } else {
```

```
        return "";
    }
}
public static void setLookAndFeel(){
    try{
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }catch(Exception e){

    }
}
public int getGoldinX(){
    return Integer.parseInt(AUinX.getText());
}
public int getGoldinY(){
    return Integer.parseInt(AUinY.getText());
}
public int getSilinX(){
    return Integer.parseInt(NAinX.getText());
}
public int getSilinY(){
    return Integer.parseInt(NAinY.getText());
}
public double getSurAmp(){
    return Double.parseDouble(SurAmp.getText());
}
public double getSurFreq(){
    return Double.parseDouble(SurFreq.getText());
}
public double getSubAmp(){
    return Double.parseDouble(SubAmp.getText());
}
public double getSubFreq(){
    return Double.parseDouble(SubFreq.getText());
}
/**
* @param args the command line arguments
*/
```

```
    public static void main(String args[]) {
        setLookAndFeel();

        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new MainWindow().setVisible(true);

            }
        });
    }
    private double TotalNumberAtoms = 0.0;
    private double GoldLength = 0.0;
    private double GoldSurfaceHeight = 0.0;
    private double SiliconLength = 0.0;
    private double SiliconSubHeight = 0.0;
    private String LocalServer = "baskes.physics.mun.ca";
    private String ACENETServer = "placentia.ace-net.ca"; //add 3 if they break head node
placentia3.ace-net.ca
    private Color SiliconColor = Color.BLACK;
    private Color GoldColor = Color.RED;
    private Atom[] Lattice = new Atom[0];
    public LatticePlot demo;
    public LatticePlot ZoomGoldLatPlot;

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JTextArea ACENET_Script_Field;
    private javax.swing.JTextField AUinX;
    private javax.swing.JTextField AUinY;
    private javax.swing.JButton AppSubRough;
    private javax.swing.JButton AppSurRough;
    private javax.swing.JRadioButton BSERV;
    private javax.swing.JButton ChangeGoldColour;
    private javax.swing.JButton ChangeSilColour;
    private javax.swing.JRadioButton CompRun;
    private javax.swing.JCheckBox DisSubLine;
    private javax.swing.JCheckBox DisSurLine;
    private javax.swing.JTextField DisplayOutName;
```

```
private javax.swing.JTextField Email;
private javax.swing.JPanel GoldColorField;
private javax.swing.JColorChooser GoldColourChanger;
private javax.swing.ButtonGroup HoleSelect;
private javax.swing.JTextField InputFileNameDisplay;
private javax.swing.JTextField MaxStretch;
private javax.swing.JButton MkANscr;
private javax.swing.JButton MkNLat;
private javax.swing.JTextField NAinX;
private javax.swing.JTextField NAinY;
private javax.swing.JTextField NumSteps;
private javax.swing.JButton OpenLat;
private javax.swing.JCheckBox OpenLatCheck;
private javax.swing.JFileChooser OpenLatticeFile;
private javax.swing.JProgressBar ProgressBar;
private javax.swing.JButton QuiT;
private javax.swing.JButton ReSeT;
private javax.swing.JCheckBox Recov;
private javax.swing.JButton RunOnACENET;
private javax.swing.JButton RunOnLocal;
private javax.swing.JButton RunOnOther;
private javax.swing.JRadioButton SACE;
private javax.swing.JFileChooser SAFileChooser;
private javax.swing.JButton SavOutLatViz;
private javax.swing.JButton SavOutLatVizZoomGold;
private javax.swing.JButton SaveCurLat;
private javax.swing.JCheckBox SaveLatCheck;
private javax.swing.JFileChooser SaveVizOut;
private javax.swing.JButton SelSaveLoc;
private javax.swing.JButton SelVizOut;
private javax.swing.JPanel SiliconColorField;
private javax.swing.JColorChooser SiliconColourChange;
private javax.swing.JTextField SimName;
private javax.swing.JTextField SimNumber;
private javax.swing.JTextField SubAmp;
private javax.swing.JTextField SubFreq;
public javax.swing.JTextField SurAmp;
```

```
private javax.swing.JTextField SurFreq;
private javax.swing.JTextField SurSprConst;
private javax.swing.JTextField UNAME;
private javax.swing.JPasswordField UPASS;
private javax.swing.JButton Update_ACENET_Script;
private javax.swing.JRadioButton VizBack;
private javax.swing.JButton VizInLat;
private javax.swing.JFileChooser VizOutFile;
private javax.swing.JButton VizOutLat;
private javax.swing.JTextField VizOutputPic;
private javax.swing.JButton ZoomGoldLat;
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.ButtonGroup buttonGroup2;
private javax.swing.ButtonGroup buttonGroup3;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel18;
private javax.swing.JLabel jLabel19;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel22;
private javax.swing.JLabel jLabel23;
private javax.swing.JLabel jLabel26;
private javax.swing.JLabel jLabel27;
private javax.swing.JLabel jLabel28;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JPanel jPanel1;
```

```
    private javax.swing.JPanel jPanel10;
    private javax.swing.JPanel jPanel11;
    private javax.swing.JPanel jPanel13;
    private javax.swing.JPanel jPanel14;
    private javax.swing.JPanel jPanel15;
    private javax.swing.JPanel jPanel16;
    private javax.swing.JPanel jPanel17;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JPanel jPanel7;
    private javax.swing.JPanel jPanel8;
    private javax.swing.JPanel jPanel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JSeparator jSeparator3;
    // End of variables declaration//GEN-END:variables

}
```

Appendix G: Lattice Input File Creation Code (C99)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#define SiBL 5.43
#define VAu ( sqrt( pow(4.08,2)*3.0 ) ) / 3.0
#define HAu sqrt( 2 * pow(2.04,2) + pow(4.08,2) )
#define VSep 1.1882
typedef struct{
    double x;
    double y;
    char SB;
    char SilGold;
    char AtomRV;
}Atom;
int main (int argc, char * argv[]){
    long int NAinX, NAinY, GoldThickness, AuinX, TotalAtoms, AtomIndex;
    double TLength, GoldLatticeSep, yoff;
    FILE *Out;
    Atom *Array;
    AtomIndex = 0;
    NAinX = atoi(argv[1]);
    NAinY = atoi(argv[2]);
    GoldThickness = atoi(argv[3]);
    GoldLatticeSep = HAu;
    TLength = (NAinX-1) * SiBL;
    AuinX = (long int) ((TLength/HAu)+0.5);
    TLength = TLength - AuinX * GoldLatticeSep;
    GoldLatticeSep += TLength / AuinX;
    AuinX++;
    TotalAtoms = NAinX * (NAinY - GoldThickness) + (AuinX)*GoldThickness;
    Array = (Atom *) malloc( TotalAtoms * sizeof(Atom));
    for(int j = 0; j < (NAinY - GoldThickness); j++){
        for(int i = 0; i < NAinX; i++){
            if ( j % 2 == 0){
                Array[AtomIndex].x = (SiBL / 2.0) + i * SiBL;
                Array[AtomIndex].y = j * SiBL/2.0;
```

```
                Array[AtomIndex].SB = 'b';
                Array[AtomIndex].SilGold = 'S';
            }
            else{
                Array[AtomIndex].x = i * SiBL;
                Array[AtomIndex].y = j * SiBL/2.0;
                Array[AtomIndex].SB = 'b';
                Array[AtomIndex].SilGold = 'S';
            }
            AtomIndex++;
        }
    }
    yoff = Array[AtomIndex-1].y;
    for(int j = 0; j< GoldThickness; j++){
        for (int i = 0; i < AuinX; i++){
            Array[AtomIndex].x =( j % 3 )*GoldLatticeSep / 3.0 + i * GoldLatticeSep;
            Array[AtomIndex].y = j * VAu + VSep + yoff;
            Array[AtomIndex].SB = 'b';
            Array[AtomIndex].SilGold = 'G';
            AtomIndex++;
        }
    }
    for ( int i = 0; i<=AuinX; i++){ Array[AtomIndex - i].SB = 's'; }

    Out = fopen(argv[4], "w");
    if(Out==NULL){
        printf("Error: Can't Write to file.\n"); exit(-1);
    }
    for(int i = 0; i< AtomIndex; i++){
        fprintf(Out,"%u,%lf,%lf,%c,%c,r\n", i+1, Array[i].x,
                                    Array[i].y, Array[i].SB, Array[i].SilGold);
    }
    fclose(Out);
    free(Array);
    return 0;
}
```