MEASUREMENT OF THE ADHESION FORCE BETWEEN A CARBON NANOTUBE AND A CURVED SILICON SURFACE

by

Daniel Anderson

Advisor: Bret C. Hess Ph.D.

Physics 492R Capstone Project Report submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics and Astronomy

Brigham Young University

August 2009

ABSTRACT

MEASUREMENT OF THE ADHESION FORCE BETWEEN A CARBON NANOTUBE AND A CURVED SILICON SURFACE

Daniel Anderson

Department of Physics and Astronomy

Bachelor of Science

We calculate the adhesion force between a carbon nanotube and differently shaped surfaces of a silicon block. We use the empirical molecular dynamics program “GULP” to test this. We vary the length of the adhesion contact surface and measure the respective adhesive force. We also vary the curvature of the surface and measure the adhesive force. We find that for contact surfaces between 5nm and 35nm the adhesive force varies linearly with the contact length. We also find that for a (7, 0) carbon nanotube which is kept at approximately 1.9Å from a curved silicon surface, the adhesion force increases rapidly for small radii as the number of nearest neighbor Si-C bonds increases and the nanotube becomes fully embedded in silicon.

ACKNOWLEDGMENTS

I would like to acknowledge and thank Dr. Hess for being my advisor and mentor over the last year and a half. I sincerely appreciate all of his help. I would also like to acknowledge Dr. Julian D. Gale of the Curtin University of Technology in Perth, Australia, for developing the GULP program. I would also like to thank Dr. Jeffrey T. Frey for sending me his code for the program TubeGen, which allowed me to create nanotube coordinate files quickly and easily. Lastly, I would like to thank the BYU Physics Department for providing what I believe is the best undergraduate physics program in the nation.

Introduction

Our understanding of the physics of nanostructures is affecting our technological planet. In a recent article in the July 13th, 2009 issue of EE Times it states “The semiconductor memory industry is about to experience major technological changes as three-dimensional multi-gate structures push transistors and memory architectures forward”[1] The theoretical structures referenced in this article were multilevel nanowire composites.

In 2003, a fully synthetic nanoscale electromechanical (NEM) actuator (~300 nm) was constructed and tested by researchers at U.C. Berkeley[2]. This nanostructure is of particular interest to our study as it utilized a very similar structure to that which our paper studies. The U.C. Berkeley device incorporated a rotatable metal plate attached to a multi-walled carbon nanotube, which served as the key motion-enabling element, which was attached to a silicon substrate. The researchers at UC Berkeley found that “the unusual mechanical and electronic properties of carbon and boron-nitride nanotubes (including favorable elastic modulus and tensile strength, high thermal and electrical conductivity, and low inter-shell friction of the atomically smooth surfaces) suggest that nanotubes may serve as important NEMS enabling materials.” [3]

The two nanostructures described above are but two examples of the hundreds of theoretical and experimental devices that will someday improve our world. Clearly, a thorough understanding of the mechanical, electrical and quantum properties of these devices is essential to their development and future success. Many nanostructures like microelectromechanical systems (MEMS) require the understanding of the relative motion of objects in contact. This understanding is also essential for controlling macroscopic lubrication and adhesion.[4]

Similar to our study here of a carbon nanotube in close proximity to a silicon substrate, the physics department at Vanderbilt University has done a study modeling carbon nanotubes encapsulated in a silicon dioxide block. They were particularly interested in a situation where the nanotube could be made to freely float in this structure and where the nanotube in this configuration would exhibit electronic properties similar to a pristine nanotube in a vacuum. [5]

Another related study was performed here at BYU. Dr. Robert Davis created a novel approach for the *Measurement of the Adhesion Force between Carbon Nanotubes and a Silicon Dioxide Substrate*.[6] In this study, carbon nanotubes were suspended over a trench that was etched into a flat silicon dioxide substrate. A vertical force was applied downward at the center of the tube using an atomic force microscope. The study quantitatively measured how much force was required to cause the nanotubes to slip, thereby measuring the adhesion force between the nanotube and the substrate. Dr. Davis' group showed in their experiments that an applied vertical force from an atomic force microscope, in force distance mode, caused the tubes to slip across the 250-nm-wide silicon dioxide trench tops with an axial tension of 8 nN.[7] One interesting conclusion reached by Dr. Davis was that a simple force per unit length friction model was insufficient for describing the nanotube adhesion to a silicon dioxide surface at their experimental contact lengths.

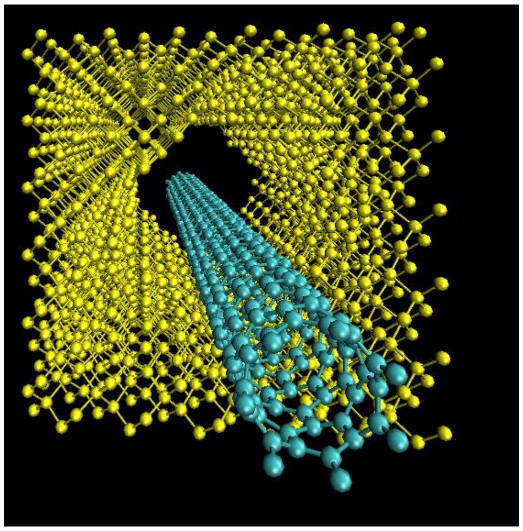
Figure 1 - Perspective picture of one of our simulations. Depicts a carbon nanotube encased inside of a silicon block

It is our purpose here to contribute research to this rapidly growing world of nanostructures. In particular, we will study a nanostructure consisting of a carbon nanotube attached to a curved silicon surface. We will calculate the adhesion force between the carbon nanotube and the curved silicon surface.

We chose carbon nanotubes because they have been shown to exhibit many surprising and impressive properties and are among the most promising structures for a range of applications that exploit the novel properties of materials at the nanoscale. Of necessity, the carbon nanotubes are in proximity and interact with other constructed materials.[8] The control of the position and shape of nanotubes is also essential if the nanotubes are to be used in nanostructures.[9]

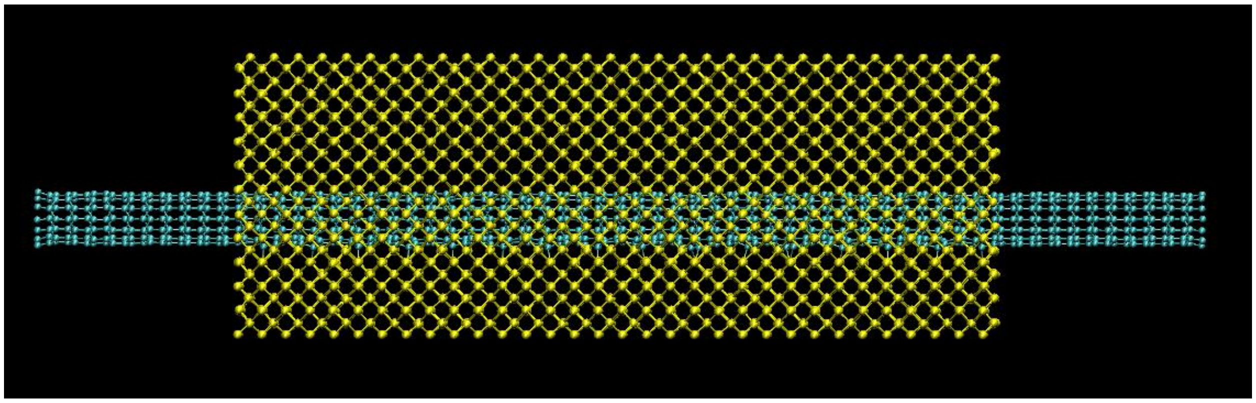

Figure 2 - Side view of a carbon nanotube encased inside of a silicon block

Silicon is the primary material used in construction of nanomechanical systems such as the NEMs and MEMS devices. Silicon can be used both to create mechanical structures and electronic circuits and circuit elements. The silicon structures are relatively easy to design and construct using photo lithographic and material deposition techniques. These techniques were developed and refined for the manufacturing of semiconductors and integrated circuits. Silicon also exhibits many properties similar to carbon and their bonding and other interactions have been studied extensively.

It is for these reasons that we have chosen to study the interaction forces between a carbon nanotube and silicon surfaces.

The study herein uses the empirical molecular simulation software called GULP[10] to simulate an experiment like unto the work of Dr. Davis. However, our simulation investigates a silicon substrate rather than silicon dioxide. The carbon nanotube is adhered to a curved surface (rather than a flat surface) of varying radii and varying lengths. A force counteracting adhesion is applied to axially pull the tube until it is displaced two carbon nanotube lattice cell lengths along the inside wall of the silicon hole. We chose two lattice cell lengths to demonstrate that the adhesion force had been overcome because of the periodicity of both the carbon nanotube and the silicon block. Essentially, once the nanotube is displaced two lattice cell lengths, all atoms in the system are the same distances from their nearest neighbors as they were in the initial condition and therefore we conclude that the nanotube would continue outward through the block. Before continuing to our simulations, I would like to discuss GULP and how it works.

<u>GULP</u>

GULP performs molecular dynamic simulations using empirically-derived, material-specific force field methods.[11] GULP iteratively processes total energy calculations of a given molecular system beginning with an initial determination of the internal energy. [12]

As GULP proceeds to iterate its calculations in time, it accounts for many different forces in the given molecular system. The forces and bonding mechanisms considered especially important to our study that GULP employs are as follows:

1. Bond order potentials – describes the strength of a bond as dependent on the environment of the bond. Uses bond length, number of bonds and angle of bonds along with empirical data to determine bond strength.[13] GULP uses specifically the Brenner model. [14]

2. Covalent bonding - two-body (atomic) short-range interactions due to atoms sharing electrons.
3. Coulomb (electric field) interactions.
4. Polarisability – describes the distortion of an electron cloud from its normal shape caused by the presence of a nearby electric field or dipole. [15]

As stated above, the GULP simulation is based on calculations of the internal energy of a system of atoms. First, GULP assumes that most of the effects of the electrons of an atom will be subsumed into that atom. GULP then decomposes the internal energy of the system into an expansion of terms of interactions between different groupings of the total number of atoms, N:

$$U = \sum_{i=1}^{N} U_i + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} U_{ij} + \frac{1}{6}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} U_{ijk} + ...$$

The first term here represents the self energies of the atoms, the second term represents 2-atom interactions, etc. If performed to a great enough order this decomposition will be exact. However, because this could be very computationally time consuming, GULP imposes computational limitations (truncation of terms) and compensates for this with reasonable parameterizations to make the problem more reasonably calculable. [16]

Modeling for the internal energy is materials-specific. Based on the system of atoms under consideration GULP will employ different force field evaluations. For example, we may wish to consider ionic interactions. Coulomb interaction can represent up to 90% of the total energy in an ionic system. [17] For two charged atoms, Coulomb's law is given by:

$$U_{ij}^{Coulomb} = \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}}$$

Again, GULP must make simplifying approximations when considering large systems of atoms, even for the simple case of ionic interactions. From the above formula we see that energy decays as the inverse power of r, but the number of interacting ions in a system of ions would increase with the surface area of a sphere at $4\pi r^2$. Thus, energy density of interactions actually amplifies with distance instead of diminishing. [18]

As we stated previously, the Brenner model implemented in GULP is of particular interest to us. The Brenner model is evaluated with bond order potentials. Bond order potentials are empirical potentials that are advantageous because they describe several different bonding states of an atom and can effectively describe chemical reactions. [19] Binding energy for bond order potentials is expressed as follows:

$$U^{BO} = \sum_{i=2}^{N} \sum_{j=1}^{i-1} \left[U^{repulsive}\left(r_{ij}\right) - B_{ij} U^{attractive}\left(r_{ij}\right) \right]$$

where B_{ij} is the bond order between the atoms i and j. The bond order depends on the local situation of both atoms i and j, and by these means converts a two-body interaction into a many-body one. [20] Note that the bond order potential is a summation of both repulsive and attractive forces.

Using all of the tools described above, GULP calculates the forces on each atom at a particular instant of time. GULP calculates the reaction of each atom to the forces applied to each atom over the short time interval using Newton's second law: $\vec{F} = \frac{\mathrm{d}}{\mathrm{d}t}(m\vec{v})$. The subsequent positions of the atoms are determined as resulting from these forces applied in the time interval. This creates a new picture of the assembled atoms. The forces and energies are then recalculated based on the new positions and energies and the process is repeated. This process creates a sequence of frames which are then viewed as a "movie".

For more information on GULP's equations, conditions, and definitions of terms and parameters, please refer to the most current version of General Utility Lattice Program[21], by Julian D. Gale, Nanochemistry Research Institute, Department of Applied Chemistry, Curtin University of Technology, P.O. Box U1987, Perth 6845, Western Australia, email: gulp@ivec.org. This paper references the original source material for calculations, models and descriptions.

Method

The coupled molecular dynamic simulation capabilities of GULP have enabled us to develop a method to measure adhesion between a carbon nanotube and a silicon surface. To do this we employ GULP's ability to tether some atoms and apply vectored forces to other atoms. Tethered atoms remain fixed in space relative to the observer's coordinate system. Atoms which receive vectored forces may move if the combined forces are sufficient.

Because GULP requires an input file describing all the initial conditions and parameters of an experiment, we had to write a program that would create these conditions efficiently. This program is attached as Appendix A. We spent over 120 hours writing the software which does the following: Prompts the user for the size of the silicon block, the radius and location of the hole to be "drilled" into the block, the type, size, and location of the nanotube to be placed in the hole, the amount of force to be applied to the nanotube, and other various parameters relating to experimental conditions such as: tethered atoms, ambient temperature, run time, step size, etc.

Upon receiving all of the necessary initial conditions from the user, the program creates a file written in GULP readable code which describes the x, y and z coordinates of every atom in the system, a list of all tethered atoms, a list of all atoms on which a force is to be applied and all other required parameters for the simulation. This file can be “fed” directly into GULP and executed as is. Along with this file, the program creates a companion file which contains instructions to the BYU Marylou Supercomputer which can also be submitted directly as is. Although writing this software took considerable time, the project would have been impossible without it. It is our hope that this software will be used by future students who want to use GULP to study molecular systems involving either nanotubes and/or crystalline structures.

Results and Discussion

For our first set of simulations we chose to calculate the adhesion force between a carbon nanotube and varying lengths of a silicon surface. We chose, based on the similar studies mentioned in the introduction, to use a (7, 0) carbon nanotube in all of our simulations. Choosing the (7, 0) nanotube also gave us the advantage of reducing the total number of atoms in our system as it is a nanotube with a somewhat small radius. We chose the following dimensions of silicon blocks to experiment upon:

Silicon block unit cell dimensions (x,y,z)	Silicon block dimensions in Å (x,y,z)
(6, 6, 8)	(32.6, 32.6, 43.5)
(6, 6, 16)	(32.6, 32.6, 86.9)
(6, 6, 32)	(32.6, 32.6, 173.8)
(6, 6, 64)	(32.6, 32.6, 347.6)

All blocks were drilled down the z-axis with an x-y plane centered hole having a 7.5 Å radius. A (7, 0) carbon nanotube was then inserted into the hole centered in the x-axis and laying 1.9 Å from the nearest silicon atom (see figures 1 and 2). We note that the usual Si-C bond length is 1.90 Å[22] which is the arithmetic mean of C-C in diamond (1.54 Å) and the Si-Si bond length in crystalline silicon (2.34 Å). We also performed tests to show that this distance was typically achieved after relaxation. For each experiment the nanotube was centered in the z-axis with respect to the block and we allowed for approximately 20 Å of nanotube to extend beyond the ends of the silicon crystal on both sides (See figure 2). For each configuration we typically ran 10 to 15 experiments of varying forces applied to the nanotube to determine the adhesive force. Below is a picture representing a before and after shot of a nanotube in which the force exceeded the adhesion force:

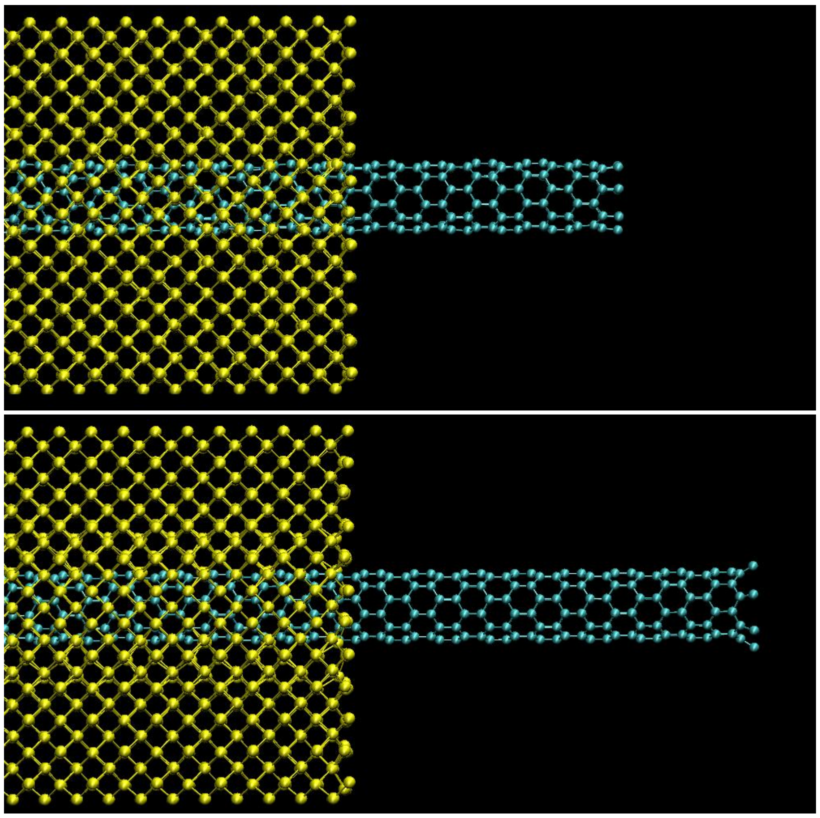

Figure 3 – The top picture shows the initial state of the carbon nanotube within the silicon block. The bottom picture shows a later state of the carbon nanotube after an axial force was applied. Note the displacement of the carbon nanotube with respect to the face of the block.

After examining closely the GULP movies, the following results were obtained:

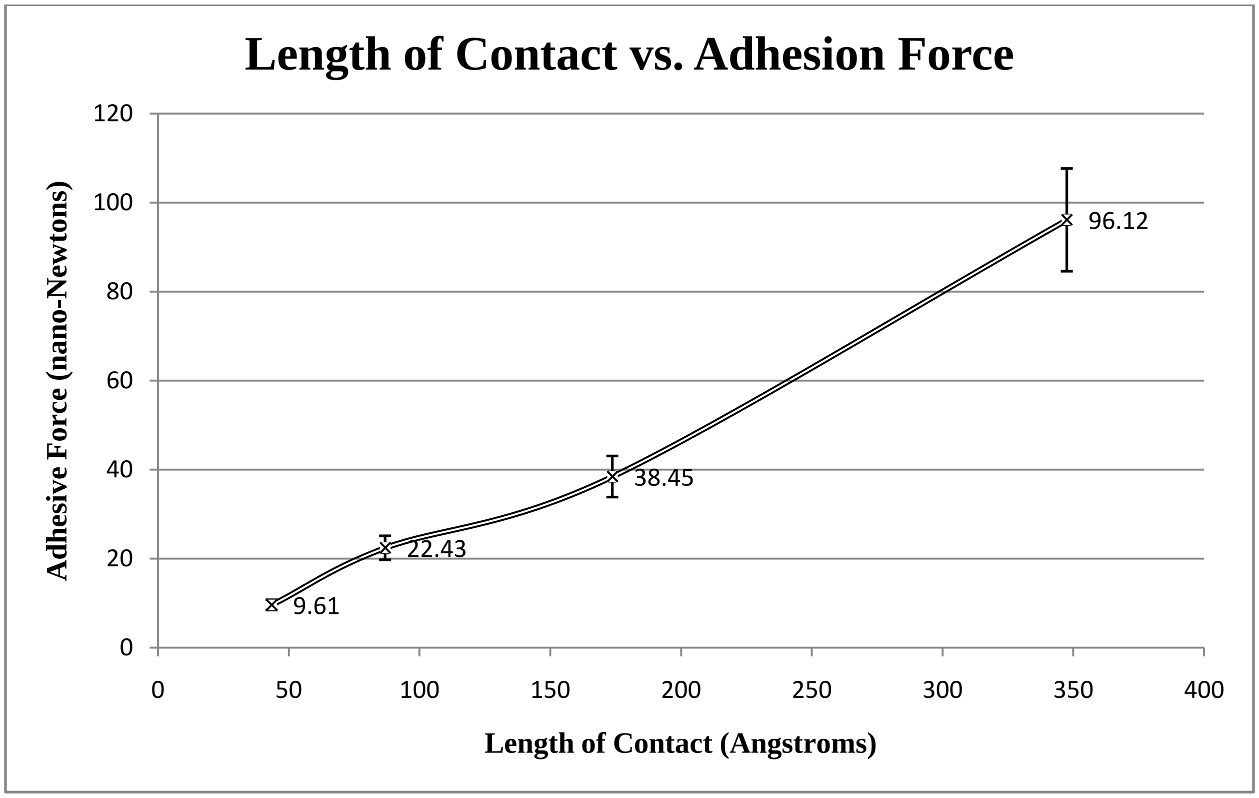

Figure 4 - The above figure represents the adhesive force as a function of the silicon - carbon contact length.

The molecular modeling demonstates a fairly linear relationship. The approximate slope of the line is .284 nano-Newtons / Å of contact. I would like to note that as the size of the atomic system increased, the time it took to run a single simulation increased seemingly exponentially. We were thus limited in the number of simulations we could perform on the larger atomic systems. The error bars on the above graph are representative of this, and are also representative of the difficulty in determining the appropriate force at which the nanotube broke free. The decimal numbers of precision in the above figure stem from the conversion of our input force, which was always an integer value of eV/Å, to nano-Newtons.

I should mention here that in our initial test simulations, we applied the total force in GULP to only the "end atoms" of the carbon nanotube. Unfotunately, this led to the end ring being ripped from the rest of the nanotube and out into space. To remedy this, we reprogrammed our software to divide the sum force we wanted to apply by the total number of atoms in the carbon nanotube and then applied the resulting dividend force to each individual atom in the nanotube. While this did fix the problem of the nanotube being ripped apart, it created a lesser problem of an asymetry in the force being applied. This occurred because the bottom surface of the nanotube is more adheared to the silicon then the top surface, the resultant force then became a type of torque pivoting around the bottom of the nanotube. This created a significant asymmetry in our force causing our applied force to not act as efficiently as possible, and thus some degree of error is introduced into our results. One possible solution to this problem would be to use a much bigger carbon nanotube, or say, one with a much larger radius. As the Si-C bond is apparently much stronger than the C-C bond, a nanotube with a larger radius would have many more total C-C bonds per ring than Si-C bonds per ring. This would hopefully result in a sum C-C bond that would be strong enough to hold the tube together and we could go back to applying the axial force only to the "end atoms" thus creating a more symetrical force to the nanotube. Unfortunately, this would also create a problem in that the system would contain many more atoms, which would result in much longer computation times. As of this writing, we have spent over 500 processor hours on the BYU supercomputer in order to achieve our results.

In our second set of simulations, we held as constant the length of the silicon-carbon contact, but varyied the radius of curvature of the hole in the silicon, beginning at a fully encapsulated nanotube and extending to a radius of about 7 times larger than the radius of the nanotube itself. This essentially represented a flat surface. The following are the results we obtained:

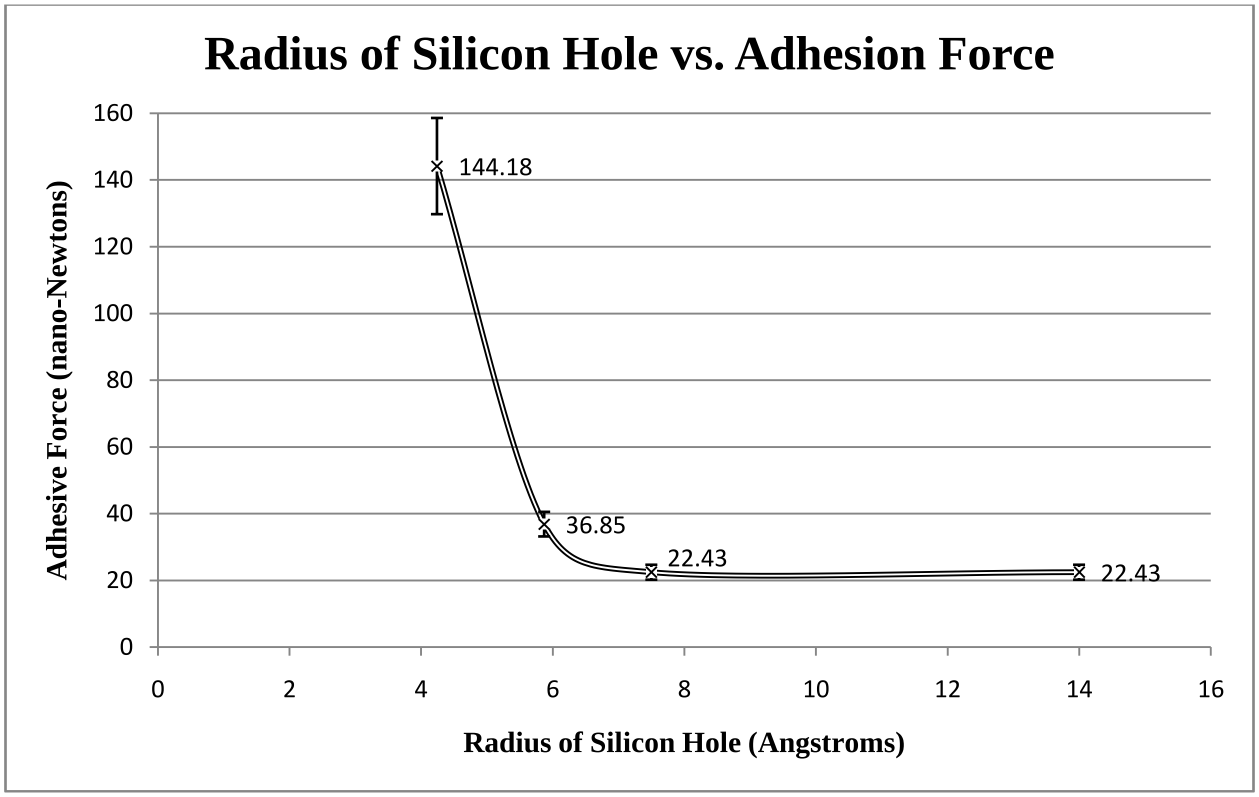

Figure 5 - This figure represents the adhesive force as a function of the radius of the silicon hole enclosing the carbon nanotube. The length of Si-C contact in these simulations was held constant.

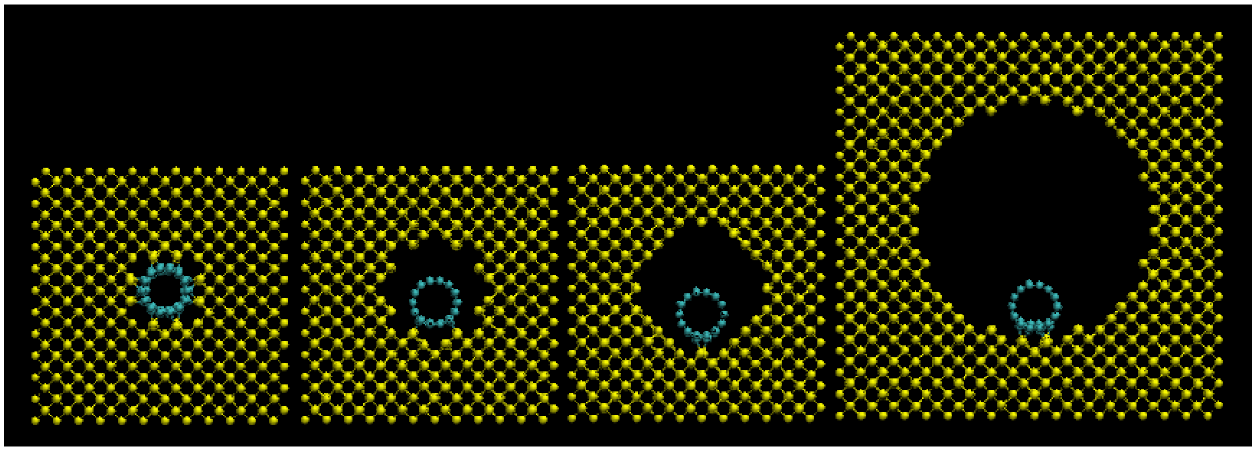

Figure 6 - This figure demonstrates the relative sizes of the silicon holes and carbon nanotubes in the above results.

Interested by these results, we notice that the adhesive force spikes rapidly as the radius of the hole approaches full encapsulation of the carbon nanotube in the bulk silicon. We decided to count the number of nearest neighbor bonds, defined as the number of Si-C bonds shorter than 1.9Å, and then divide this number into the total adhesion force to hopefully see a flat linear relationship. The following is the result:

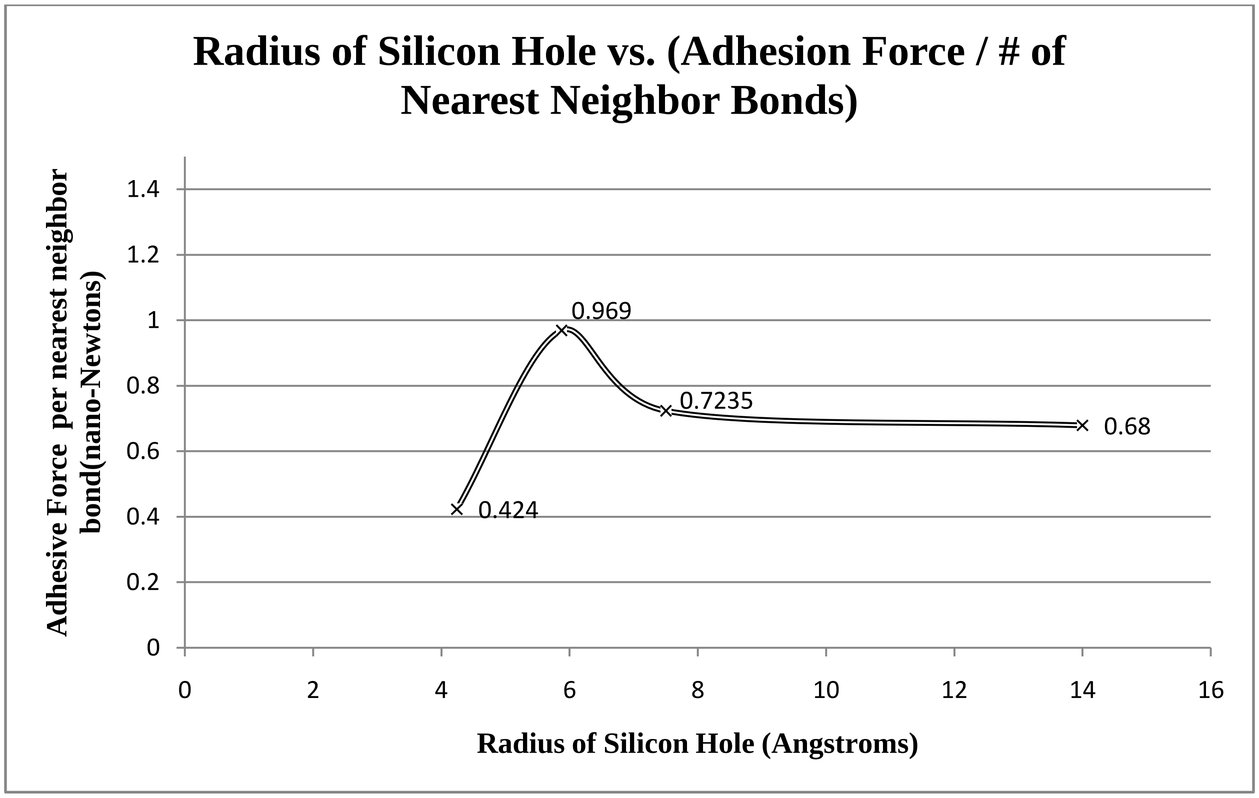

Figure 7 - This figure represents the adhesion force to number of nearest neighbor bonds for each of the different simulations

Although not a flat line as we had hoped, the numbers are at least somewhat close to each other (same order of magnitude). We propose that the "force per nearest neighbor bond" is most accurately described by the first test of hole radius 4.242 Å (coincidently corresponding to .424 nano-Newtons per nearest neighbor bond). The adhesive force in this particular configuration is most likely made up of almost exclusively nearest neighbor bonds. In the next simulation (hole radius 5.9 Å) there are significantly less nearest neighbor bonds (38 compared to the previous 340 nearest neighbor bonds), but there are more atoms that are "close to" but do not quite fit under our description of nearest neighbor. It appears that these atoms are contributing significantly to the adhesive force as demonstrated by the sharp, unexpected increase in "adhesive force per nearest neighbor" figure. If our supposition is correct then we should see a decrease in the next simulation due to the fact that although the number of nearest neighbor bonds does not decrease much (38 to 31), the number of "almost" nearest neighbor bonds decreases significantly. This ends up being the case. Continuing then as expected, the force per nearest neighbor bond decreases again with the decrease in "almost" nearest neighbor bonds for the last simulation of hole radius 14 Å. We had not expected that the Si-C interactions longer that 1.9 Å would have contributed as much as they apparently did. I find this to be a notable result.

Conclusions

In conclusion, in our study of carbon nanotubes attached to silicon surfaces, we found that for contact surfaces between 5nm and 35nm the adhesive force varies linearly with the contact length. We also found that for a (7, 0) carbon nanotube which is kept at approximately 1.9 Å from a curved silicon surface, the adhesion force increases rapidly for small radii as the number of nearest neighbor Si-C bonds increases and the nanotube becomes fully embedded in silicon. We also suspect that other Si-C bonds slightly longer than 1.9 Å can play a significant role in adhesion.

We propose for a future research direction that other molecular dynamic modeling programs such as Fireball, which uses "first principles" calculations rather than empirical measurements, be used to repeat these experiments. Additionally, as GULP is still an evolving program and future releases already anticipate refinements to the Brenner model and other models which will consider additional polymorphisms of Si-C bonds, these simulations could be repeated. Gulp could also be used in the future to calculate the binding energy by first calculating the potential energy of a system in which the carbon nanotube is extremely far away from the silicon block. Then, by calculating the potential energy in systems like those in this study, the difference between the two potential energies should yield the binding energy of the system which could be used to calculate the adhesive force.

References

[1] Pele, A.; EE Times, Issue 1565, (Monday, July 13, 2009), p. 28; *Multilevel Nanowires – Memory set to rise to 3-D challenges*.
[2] Fennimore, A.; Yuzvinsky, T.; Han, W.; Fuhrer, M.; Cumings, J.; Zettl, A.; Nature, Vol. 424 (24 July 2003), p. 408-410; *Rotational actuators based on carbon nanotubes.*
[3] Fennimore: ibidem; p. 408.
[4] Falvo, M.; Taylor, R. Helser; Chi, V.; Brooks Jr., F.; Wasburn, S.; Superfine, R.; Nature, Vol 397, (1999); *Nanometre-scale rolling and sliding of carbon nanotubes.*
[5] Tsetseris, L.; Pantelides, S.; Physical Review Letters, (2006), PRL 97, 266805; *Encapsulation of Floating Carbon Nanotubes in SiO_2.*
[6] Whittaker, J; Minot, E.; Tanenbaum, D.; McEuen, P.; Davis, R.; Nano Letters/American Chemical Society, (2006) Vol. 6, No. 5, 953-957; *Measurement of the Adhesion Force between Carbon Nanotubes and a Silicon Dioxide Substrate*
[7] Whittaker: ibidem.
[8] Tsetseris: ibidem.
[9] Hertel, T.; Martel, R.; Avouris, P; J. Phys. Chem. B (1998), 102, 910-915; *Manipulation of Individual Carbon Nanotubes and Their Interaction with Surfaces*.
[10] GULP (General Utility Lattice Program); developed by Julian Gale; https://www.ivec.org/gulp/help/gulp3.4_manual.pdf
[11] Gale: ibidem. p. 2.
[12] Gale: ibidem. p. 11.
[13] "Bond order potential." *Wikipedia, The Free Encyclopedia*. 26 Jul 2009, 13:44 UTC. 26 Jul 2009 <http://en.wikipedia.org/w/index.php?title=Bond_order_potential&oldid=304293224>.
[14] Gale: ibidem. p. 27.
[15] "Polarizability." *Wikipedia, The Free Encyclopedia*. 5 Jul 2009, 21:27 UTC. 5 Jul 2009 <http://en.wikipedia.org/w/index.php?title=Polarizability&oldid=300476948>.
[16] Gale: ibidem. p. 11.
[17] Gale: ibidem. p. 11.
[18] Gale: ibidem. p. 12.
[19] "Bond order potential." *Wikipedia, The Free Encyclopedia*. 26 Jul 2009, 13:44 UTC. 26 Jul 2009 <http://en.wikipedia.org/w/index.php?title=Bond_order_potential&oldid=304293224>.
[20] Gale: ibidem. p. 27.
[21] Gale: ibidem. p. 12.
[22] Wiberg, E.; Wiberg, N.; Holleman, A.; (2001) ; *Inorganic Chemistry*; Science; p. 861.

Appendix A

Considerable time was spent during this project to develop a C++ program to create GULP readable files which establish our initial atomic model configurations. The source code to generate the initial atomic models and GULP readable files is included in this appendix. The source code was written and executed using the free version of Microsoft® Visual C++®. It is hoped that this source code will be of value to other students modifying or extending these simulations or any other simulations done using the GULP software.

Upon compiling and running the code below, the program will successively prompt the user the following questions:

```
    Enter the name of your output file:
    How many silicon unit cells in the x-direction:
    How many silicon unit cells in the y-direction:
    How many silicon unit cells in the z-direction:
    Would you like to tether exterior silicon surface atoms? (“yes” or “no”):
    Would you like to drill a hole? (“yes” or “no”):
        If yes,
        Enter the x-coordinate of the hole in angstroms:
        Enter the y-coordinate of the hole in angstroms:
        Enter the radius of the hole in angstroms:
    Would you like to add a carbon nanotube file?
        If yes,
        Enter the name of the nanotube file to insert:   (note: the first line of the nanotube file should be the
                                                          number of atoms in the nanotube, followed by a list of all of
                                                          the atoms in the tube. This is the basic form of an .xyz file.
                                                          The following is the example form:

                                                          # of atoms

                                                          Atomic-symbol   x-coord.   y-coord.  z-coord.
                                                          Atomic-symbol   x-coord.   y-coord.  z-coord.
                                                          …               …          …         …
                                                          Atomic-symbol   x-coord.   y-coord.  z-coord.  )

        Enter the x-coordinate of where you would like the center of the tube in angstroms:
        Enter the y-coordinate of where you would like the center of the tube in angstroms:
        Would you like to apply external forces to the atoms of the CNT? (“yes” or “no”):
            If yes,
            There are a total of:  ###   carbon atoms in the CNT.
            Enter the z-axis external force to be applied to each carbon atom in eV/Angstrom:
    Enter the “Endforce”:          (this describes in picoseconds the amount of time the force will be applied)
    Enter the timestep:            (this describes the timestep GULP will use in its calculations)
    Enter the Equilibriation:      (this describes in picoseconds the amount of time GULP will allow the system to
                                   “relax” before applying the external force)
    Enter the production time:     (this describes in picoseconds the total run-time of the GULP simulation)
    Enter the Movie Sample rate:   (this describes in picoseconds the time between each GULP output to the movie
                                   file)
    Enter the hours of wall time for MaryLou:   (this is your estimated calculation time for MaryLou to run your
                                                GULP simulation)
```

```cpp
// Daniel Anderson
// August 2009
// BYU Physics Capstone Project
// Silicon Builder
//
/***************************************************************************
This program is designed to build a user-defined silicon block, then at the users discretion, drill a hole in the block
of user defined dimensions. The program then reads in a user chosen nanotube file and translates the nanotube in
accordance to the user's chosen location in the hole and with respect to the silicon crystal itself. The user is also
given the option to tether the face atoms of the silicon crystal(excluding the 2 hole faces). The program also allows the
user to apply a force to the carbon nanotube.  The force is applied to all atoms of the carbon nanotube. The program
propts the user for other information needed to create the Gulp readable file i.e. production time, step size. The
program then creates both a gulp readable input file for the user created experiment and also creates a UNIX readable
file useful for batch job submission to the BYU MARYLOU SUPERCOMPUTER
***************************************************************************/

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <string>
using namespace std;

int main()
{
        char AtomName;
        int TetheredAtoms[10000];
        int TetheredCount = 0;
        int DrilHole = false;
        int TetherSurfaces = false;
        int AddCNT = false;
        int xCells, yCells, zCells, Total_Si_Atoms = 0, index, xCount = 0, yCount = 0, zCount = 0;
        int TotalCells, AtomsInHole = 0;
        int lines,counter = 1;
        int TotalAtoms, Total_C_Atoms = 0;
        int WallTime;
        double LatticeSpace = 5.431, xHole, yHole, radius, xCoord, yCoord;
        double xPos1 = 0, yPos1 = 0, zPos1 = 0, xPos2 = 0, yPos2 = 0, zPos2 = 0;
        double XExternalForce = 0.0;
        double YExternalForce = 0.0;
        double ZExternalForce = 0.0;
        double EndForce = 25.0;
        double TimeStep = 0.001;
        double Equilibration = 0.01;
        double MovieSampleRate = 0.01;
        double TotalProductionTime = 15.0;
        string filename;
        string filename1;
        string tubename;
        string Response;
        // ifstream == Input file stream class -- provides an interface to read data
        // from files as input streams
        ifstream fin;
        // ofstream == Output file stream class -- provides an interface to write data
        // to files as output streams.
        ofstream fout;
        //-------------------------------------------------------------------------------
        // Put spaces in anywhere you need to fill in blanks...
        cout << setfill(' ');
        fout << setfill(' ');
        //-------------------------------------------------------------------------------
        // Get the file name we are going to send to GULP...
        cout << "Enter the name for your output file (e.g. sil3x3x3): " << endl;
        cin >> filename;
        fout.open(filename.c_str());
        cout << endl;
        //-------------------------------------------------------------------------------
        // Get user defined number of lattice cells...
        cout << "How many silicon unit cells in the x-direction ( >=1 ): ";
        cin >> xCells;
        cout << "How many silicon unit cells in the y-direction ( >=1 ): ";
        cin >> yCells;
        cout << "How many silicon unit cells in the z-direction ( >=1 ): ";
        cin >> zCells;
        cout << endl;
        //-------------------------------------------------------------------------------
        // Determine if we should tether surfaces...
        cout << "Would you like to tether exterior silicon surface atoms on the planer"  << endl;
        cout << "surfaces where x = 0, x = maximum, y = 0, and y = maximum?"  << endl;
        cout << "(\"yes\" or \"no\"): ";
        cin >> Response;
        if( (Response == "yes") || (Response == "y") || (Response == "YES")
                       || (Response == "Y") || (Response == "Yes") || (Response == "ye")
                       || (Response == "Ye") || (Response == "YE") )
                TetherSurfaces = true;
        else TetherSurfaces = false;
        cout << endl;
        //-------------------------------------------------------------------------------
        // IF WE ARE TETHERING ATOMS...
        // Note that Atom #1 and Atom #5 will need to be tethered if xCount==0
        // Note that Atom #1 and Atom #4 will need to be tethered if yCount==0
        // Note that Atom #6 and Atom #7 will need to be tethered if xCount==xCells-1
        // Note that Atom #6 and Atom #8 will need to be tethered if yCount==yCells-1
        // The TetheredAtoms[]array is used to store the first 10000 tethered atoms...
        // We load the array with -1 to be used latter to determine a number was stored...
        for( index = 0; index < 10000; index++ ) TetheredAtoms[index] = -1;
        //-------------------------------------------------------------------------------
        // Find out if we are going to drill a hole...
        cout << "Would you like to drill a hole? (\"yes\" or \"no\"): ";
        cin >> Response;
        if( (Response == "yes") || (Response == "y") || (Response == "YES")
                       || (Response == "Y") || (Response == "Yes") || (Response == "ye")
                       || (Response == "Ye") || (Response == "YE") )
                DrilHole = true;
        else DrilHole = false;
        cout << endl;
        //-------------------------------------------------------------------------------
        // If we are going to drill a hole we need to find out where to drill the hole...
        if( DrilHole )
                {
                cout << "Enter the x-coordinate of the hole in angstroms: ";
                cin >> xHole;
                cout << "Enter the y-coordinate of the hole in angstroms: ";
                cin >> yHole;
                cout << "Enter the radius of the hole in angstroms: ";
                cin >> radius;
                cout << endl;
                //-------------------------------------------------------------------------------
                // Does the user want to add a nanotube to the build...
                cout << "Would you add a carbon nanotube file? (\"yes\" or \"no\"): ";
                cin >> Response;
                if( (Response == "yes") || (Response == "y") || (Response == "YES")
                               || (Response == "Y") || (Response == "Yes") || (Response == "ye")
                               || (Response == "Ye") || (Response == "YE") )
                       AddCNT = true;
                else AddCNT = false;
                // Add a line space on the screen...
                cout << endl;
                //-------------------------------------------------------------------------------
                // If the user wants to add a carbon nanotube file...
                if( AddCNT )
                       {
                       // Get the name of the nanotube file...
                       cout << "Enter the name of your nanotube file to insert (e.g. cnt7x0x4.xyz) : " << endl;
                       cin >> tubename;
                       // Add a line space on the screen...
                       cout << endl;
                       // Get the coordinates of the center of the tube in X-Y:
                       cout << "Enter the x-coordinate of where you would like the center of the tube" << endl;
                       cout << "in angstroms: ";
                       cin >> xCoord;
                       cout << "Enter the y-coordinate of where you would like the center of the tube" << endl;
                       cout << "in angstroms: ";
                       cin >> yCoord;
                       // Add a line space on the screen...
                       cout << endl;
                       } // End - adding the CNT..
                //-------------------------------------------------------------------------------
                } // End - we wanted a hole...
        //-------------------------------------------------------------------------------
        //-------------------------------------------------------------------------------
        //-------------------------------------------------------------------------------
        // We include comments at the beginning of the file so that the user can read
        // about how the file was generated...
        fout << "# Important information about this GULP file:" << endl;
        fout << "# Number of silicon unit cells in the x-direction: " << setw(3) << xCells << endl;
        fout << "# Number of silicon unit cells in the y-direction: " << setw(3) << yCells << endl;
        fout << "# Number of silicon unit cells in the z-direction: " << setw(3) << zCells << endl;
        if( DrilHole ) // If the user requested a hole to be drilled in the silicon...
                {
                fout << "# The user requested a hole in the silicon: " << endl;
                fout << "#   x-coordinate of the hole in angstroms: " << setw(8) << xHole << endl;
                fout << "#   y-coordinate of the hole in angstroms: " << setw(8) << yHole << endl;
                fout << "#   radius of the hole in angstroms: " << setw(8) << radius << endl;
                // If the user wants to add a carbon nanotube file...
                if( AddCNT )
                       {
                       // Name of nanotube file...
                       fout << "# Name of nanotube file used: " + tubename << endl;
                       // Coordinates of CNT...
                       fout << "#   CNT centered at x-coordinate: "  << setw(8) << xCoord << endl;
                       fout << "#   CNT centered at y-coordinate: "  << setw(8) << yCoord << endl;
                       }
                else fout << "# No CNT was added to this build... " << endl;
                }
        else fout << "# No hole was requested by the user in the silicon... " << endl;
        //--------------------------------------------------------------------------------
        //--------------------------------------------------------------------------------
        //--------------------------------------------------------------------------------
        // Add these lines to the beginning of the GULP file...
        fout << endl; // A blank line added before the first GULP instruction...
        fout << "conv md" << endl;
        fout << "cartesian" << endl;
        fout << endl; // A blank line between these lines and the atom locations...
        //--------------------------------------------------------------------------------
        // TotalCells needs to be fixed so as to include hole drilling and additional
        // nanotube atoms
        TotalCells = xCells*yCells*zCells;
        //--------------------------------------------------------------------------------
        // This portion of code creates the Silicon crystal block based on Silicon
        // "diamond" unit cell configuration...
        if( DrilHole == false )
                {
                for( zCount=0; zCount<zCells; zCount++ )
                       {
                       for( yCount=0; yCount<yCells; yCount++ )
                               {
                               for( xCount=0; xCount<xCells; xCount++ )
                                       {
                                       //----------------------------------------------------------------
                                       // NOTE THAT 8 ATOMS ARE ADDED FOR EACH UNIT CELL THAT IS ADDED
                                       // TO THE STRUCTURE...(8 IS THE EXACT NUMBER PER CELL)
                                       //----------------------------------------------------------------
                                       // Atom #1 of unit cell... (the only corner atom added)
                                               // Note: setw(8) == Sets the number of characters to be used
                                               // as the field width for the next insertion operation.
                                       fout << "Si " << setw(8) << xCount*LatticeSpace;
                                       fout << setw(8) << yCount*LatticeSpace;
                                       fout << setw(8) << zCount*LatticeSpace << endl;
                                                               // Note: endl== terminates the line and flushes buffer...
                                       Total_Si_Atoms++;
                                       // For tethered silicon surfaces...
                                       if( TetherSurfaces && ( xCount==0 || yCount==0 ) )
                                               {
                                               // Store the current atom number...
                                               TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                               // We don't want more than 10000 tethered atoms...
                                               if( TetheredCount < 9999 ) TetheredCount++;
                                               }
                                       // Atom #2 of unit cell... (an atom on the interior of the cell)
                                       fout << "Si " << setw(8) << xCount*LatticeSpace + .25*LatticeSpace;
                                       fout << setw(8) << yCount*LatticeSpace + .25*LatticeSpace;
                                       fout << setw(8) << zCount*LatticeSpace + .25*LatticeSpace << endl;
                                       Total_Si_Atoms++;
                                       // Atom #3 of unit cell... (an atom on one face of the cell)
                                       fout << "Si " << setw(8) << xCount*LatticeSpace + .5*LatticeSpace;
                                       fout << setw(8) << yCount*LatticeSpace + .5*LatticeSpace;
                                       fout << setw(8) << zCount*LatticeSpace << endl;
                                       Total_Si_Atoms++;
                                       // Atom #4 of unit cell... (an atom on another face of the cell)
                                       fout << "Si " << setw(8) << xCount*LatticeSpace + .5*LatticeSpace;
                                       fout << setw(8) << yCount*LatticeSpace;
                                       fout << setw(8) << zCount*LatticeSpace + .5*LatticeSpace << endl;
                                       Total_Si_Atoms++;
                                       if( TetherSurfaces && yCount==0 )
                                               {
```

```
                                        TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                        if( TetheredCount < 9999 ) TetheredCount++;
                                        }
                                // Atom #5 of unit cell... (an atom on third face of the cell)
                                fout << "Si " << setw(8) << xCount*LatticeSpace;
                                fout << setw(8) << yCount*LatticeSpace + .5*LatticeSpace;
                                fout << setw(8) << zCount*LatticeSpace + .5*LatticeSpace << endl;
                                Total_Si_Atoms++;
                                if( TetherSurfaces && xCount==0 )
                                        {
                                        TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                        if( TetheredCount < 9999 ) TetheredCount++;
                                        }
                                // Atom #6 of unit cell... (an atom on the interior of the cell)
                                fout << "Si " << setw(8) << xCount*LatticeSpace + .75*LatticeSpace;
                                fout << setw(8) << yCount*LatticeSpace + .75*LatticeSpace;
                                fout << setw(8) << zCount*LatticeSpace + .25*LatticeSpace << endl;
                                Total_Si_Atoms++;
                                if( TetherSurfaces && ( (xCount==(xCells-1)) || (yCount==(yCells-1)) ) )
                                        {
                                        TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                        if( TetheredCount < 9999 ) TetheredCount++;
                                        }
                                // Atom #7 of unit cell... (an atom on the interior of the cell)
                                fout << "Si " << setw(8) << xCount*LatticeSpace + .75*LatticeSpace;
                                fout << setw(8) << yCount*LatticeSpace + .25*LatticeSpace;
                                fout << setw(8) << zCount*LatticeSpace + .75*LatticeSpace << endl;
                                Total_Si_Atoms++;
                                if( TetherSurfaces && (xCount==(xCells-1)) )
                                        {
                                        TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                        if( TetheredCount < 9999 ) TetheredCount++;
                                        }
                                // Atom #8 of unit cell... (an atom on the interior of the cell)
                                fout << "Si " << setw(8) << xCount*LatticeSpace + .25*LatticeSpace;
                                fout << setw(8) << yCount*LatticeSpace + .75*LatticeSpace;
                                fout << setw(8) << zCount*LatticeSpace + .75*LatticeSpace << endl;
                                Total_Si_Atoms++;
                                if( TetherSurfaces && (yCount==(yCells-1)) )
                                        {
                                        TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                        if( TetheredCount < 9999 ) TetheredCount++;
                                        }
                                }
                        }
                }
        }
else // A hole is drilled...
        {
        // If a hole is drilled, then this part creates the SI crystal,
        // eliminating all "hole" atoms
        for( zCount=0; zCount<zCells; zCount++ )
                {
                for( yCount=0; yCount<yCells; yCount++ )
                        {
                        for( xCount=0; xCount<xCells; xCount++ )
                                {
                                //-----------------------------------------------------------------
                                // NOTE THAT 8 ATOMS ARE ADDED FOR EACH UNIT CELL THAT IS ADDED
                                // TO THE STRUCTURE...UNLESS THE ATOMS ARE IN THE SPACE RESERVED
                                // FOR THE HOLE...IN WHICH CASE THESE ATOMS ARE EXCLUDED...
                                //-----------------------------------------------------------------
                                // Atom #1 of unit cell...
                                if( sqrt(pow(xCount*LatticeSpace-xHole,2)
                                                + pow(yCount*LatticeSpace-yHole,2)) > radius )
                                        {
                                        fout << "Si " << setw(8) << xCount*LatticeSpace;
                                        fout << setw(8) << yCount*LatticeSpace;
                                        fout << setw(8) << zCount*LatticeSpace << endl;
                                        Total_Si_Atoms++;
                                        // For tethered silicon surfaces...
                                        if( TetherSurfaces && ( xCount==0 || yCount==0 ) )
                                                {
                                                // Store the current atom number...
                                                TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                                // We don't want more than 10000 tethered atoms...
                                                if( TetheredCount < 9999 ) TetheredCount++;
                                                }
                                        }
```

```
else // We count up the number of atoms in the hole...
        AtomsInHole++;
//--------------------------------------------------------------------
// Atom #2 of unit cell...
if( sqrt(pow((xCount*LatticeSpace + .25*LatticeSpace)-xHole,2)
                + pow((yCount*LatticeSpace+.25*LatticeSpace)-yHole,2)) > radius )
        {
        fout << "Si " << setw(8) << xCount*LatticeSpace + .25*LatticeSpace;
        fout << setw(8) << yCount*LatticeSpace + .25*LatticeSpace;
        fout << setw(8) << zCount*LatticeSpace + .25*LatticeSpace << endl;
        Total_Si_Atoms++;
        }
else
        AtomsInHole++;
//--------------------------------------------------------------------
// Atom #3 of unit cell...
if( sqrt(pow((xCount*LatticeSpace + .5*LatticeSpace)-xHole,2)
                + pow((yCount*LatticeSpace+.5*LatticeSpace)-yHole,2)) > radius )
        {
        fout << "Si " << setw(8) << xCount*LatticeSpace + .5*LatticeSpace;
        fout << setw(8) << yCount*LatticeSpace + .5*LatticeSpace;
        fout << setw(8) << zCount*LatticeSpace << endl;
        Total_Si_Atoms++;
        }
else
        AtomsInHole++;
//--------------------------------------------------------------------
// Atom #4 of unit cell...
if( sqrt(pow((xCount*LatticeSpace + .5*LatticeSpace)-xHole,2)
                + pow(yCount*LatticeSpace-yHole,2)) > radius )
        {
        fout << "Si " << setw(8) << xCount*LatticeSpace + .5*LatticeSpace;
        fout << setw(8) << yCount*LatticeSpace;
        fout << setw(8) << zCount*LatticeSpace + .5*LatticeSpace << endl;
        Total_Si_Atoms++;
        if( TetherSurfaces && yCount==0 )
                {
                TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                if( TetheredCount < 9999 ) TetheredCount++;
                }
        }
else
        AtomsInHole++;
//--------------------------------------------------------------------
// Atom #5 of unit cell...
if( sqrt(pow(xCount*LatticeSpace-xHole,2)
                + pow((yCount*LatticeSpace +.5*LatticeSpace)-yHole,2)) > radius )
        {
        fout << "Si " << setw(8) << xCount*LatticeSpace;
        fout << setw(8) << yCount*LatticeSpace + .5*LatticeSpace;
        fout << setw(8) << zCount*LatticeSpace + .5*LatticeSpace << endl;
        Total_Si_Atoms++;
        if( TetherSurfaces && xCount==0 )
                {
                TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                if( TetheredCount < 9999 ) TetheredCount++;
                }
        }
else
        AtomsInHole++;
//--------------------------------------------------------------------
// Atom #6 of unit cell...
if( sqrt(pow((xCount*LatticeSpace + .75*LatticeSpace)-xHole,2)
                + pow((yCount*LatticeSpace+.75*LatticeSpace)-yHole,2)) > radius )
        {
        fout << "Si " << setw(8) << xCount*LatticeSpace + .75*LatticeSpace;
        fout << setw(8) << yCount*LatticeSpace + .75*LatticeSpace;
        fout << setw(8) << zCount*LatticeSpace + .25*LatticeSpace << endl;
        Total_Si_Atoms++;
        if( TetherSurfaces && ( (xCount==(xCells-1)) || (yCount==(yCells-1)) ) )
                {
                TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                if( TetheredCount < 9999 ) TetheredCount++;
                }
        }
else
        AtomsInHole++;

//--------------------------------------------------------------------
```

```
                                // Atom #7 of unit cell...
                                if( sqrt(pow((xCount*LatticeSpace + .75*LatticeSpace)-xHole,2)
                                                + pow((yCount*LatticeSpace+.25*LatticeSpace)-yHole,2)) > radius )
                                        {
                                        fout << "Si " << setw(8) << xCount*LatticeSpace + .75*LatticeSpace;
                                        fout << setw(8) << yCount*LatticeSpace + .25*LatticeSpace;
                                        fout << setw(8) << zCount*LatticeSpace + .75*LatticeSpace << endl;
                                        Total_Si_Atoms++;
                                        if( TetherSurfaces && (xCount==(xCells-1)) )
                                                {
                                                TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                                if( TetheredCount < 9999 ) TetheredCount++;
                                                }
                                        }
                                else
                                        AtomsInHole++;
                                //----------------------------------------------------------------
                                // Atom #8 of unit cell...
                                if( sqrt(pow((xCount*LatticeSpace + .25*LatticeSpace)-xHole,2)
                                                + pow((yCount*LatticeSpace+.75*LatticeSpace)-yHole,2)) > radius )
                                        {
                                        fout << "Si " << setw(8) << xCount*LatticeSpace + .25*LatticeSpace;
                                        fout << setw(8) << yCount*LatticeSpace + .75*LatticeSpace;
                                        fout << setw(8) << zCount*LatticeSpace + .75*LatticeSpace << endl;
                                        Total_Si_Atoms++;
                                        if( TetherSurfaces && (yCount==(yCells-1)) )
                                                {
                                                TetheredAtoms[TetheredCount] = Total_Si_Atoms;
                                                if( TetheredCount < 9999 ) TetheredCount++;
                                                }
                                        }
                                else
                                        AtomsInHole++;
                                }
                        }
                }
        } // End building Si if a hole was drilled...
//----------------------------------------------------------------------------------
// Here we insert the nanotube and translate its position to center it in the tube
// Nanotube xyz file is generated by web based program
// Go to: http://turin.nss.udel.edu/research/tubegenonline.html
if( DrilHole && AddCNT ) // Only add the file if...
        {
        // Open up the nanotube file...
        fin.open( tubename.c_str() );
        fin >> lines;
        //---------------------------------------------------------------------------------
        while( counter <= lines )
                {
                fin >> AtomName;
                fin >> xPos1;
                fin >> yPos1;
                fin >> zPos1;

                xPos1 = xPos1 + xCoord;
                yPos1 = yPos1 + yCoord;
                zPos1 = zPos1 + ( zCells * ( LatticeSpace/2 ));
                fout << AtomName << " ";
                fout << setw(8) << xPos1 << " ";
                fout << setw(8) << yPos1 << " ";
                fout << setw(8) << zPos1 << endl;
                Total_C_Atoms++;
                counter++;
                }
        //-----------------------------------------------------------------------------
        // Close up the nanotube file...
        fin.close();
        } // Only add the file if there was a hole drilled...

//---------------------------------------------------------------------------------
// Find out if we are going to apply external forces to the atoms of the CNT...
if( Total_C_Atoms ) // We don't do this unless there is a CNT in the first place...
        {
        cout << endl; // Skip a line...
        cout << "Would you like to apply external forces to the atoms of the CNT?" << endl;
        cout << " (\"yes\" or \"no\"): ";
        cin >> Response;
        if( (Response == "yes") || (Response == "y") || (Response == "YES")
                        || (Response == "Y") || (Response == "Yes") || (Response == "ye")
                        || (Response == "Ye") || (Response == "YE") )
                {
                cout << endl; // Skip a line...
              cout << "There are a total of: " << setw(6) << Total_C_Atoms << "   Carbon atoms in the CNT." << endl;
                cout << "Enter the z-axis external force to be applied to each carbon atom" << endl;
                cout << "in eV/Angstrom: ";
                cin >> ZExternalForce;
                // Write a blank line to the GULP file...
                fout << endl;
                // Write external_force to the GULP file...
                fout << "external_force" << endl;
                // Loop through to add forces to all of the CNT atoms
                for( index = 1; index < Total_C_Atoms; index++ )
                        {
                        // Note that the Carbon atom numbers are listed after the
                        // Silicon atoms...therefore...the Carbon atom number is
                        // equal to: (Total_Si_Atoms + index)
                        fout << setbase(10);
                        fout << (Total_Si_Atoms + index) << " ";
                        fout << setw(8) << XExternalForce << " "; // XExternalForce set to zero...
                        fout << setw(8) << YExternalForce << " "; // YExternalForce set to zero...
                        fout << setw(16) << ZExternalForce << endl; // Finally apply the ZExternalForce...
                        } // End - for loop to add all of the applied forces...
                } // End - we did want to apply external_force ...
        } // End - there was a CNT added...
//---------------------------------------------------------------------------
// Add these lines if we have added a CNT for control of the modeling...
if( AddCNT )
        {
        // Add a line space on the screen...
        cout << endl;
        // Get the endforce...
        cout << "Enter the \"endforce\" (e.g. 25): " << endl;
        cin >> EndForce;
        // Add a line space on the screen...
        cout << endl;
        // Get the TimeStep...
        cout << "Enter the \"timestep\" (e.g. 0.001): " << endl;
        cin >> TimeStep;
        // Add a line space on the screen...
        cout << endl;
        // Get the equilibration...
        cout << "Enter the \"equilibration\" (e.g. 0.01): " << endl;
        cin >> Equilibration;
        // Add a line space on the screen...
        cout << endl;
        // Get the equilibration...
        cout << "Enter the production time \"production\" (e.g. 15.0): " << endl;
        cin >> TotalProductionTime;
        // Add a line space on the screen...
        cout << endl;
        // Get the sample...
        cout << "Enter the Movie Sample Rate\"sample\" (e.g. 0.01): " << endl;
        cin >> MovieSampleRate;
        } // End - control modeling...
else // If there was no AddCNT...
        { // then use these default values...
        EndForce = 25;
        TimeStep = 0.001;
        Equilibration = 0.01;
        MovieSampleRate = 0.01;
        TotalProductionTime = 15.0;
        }
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
// Write a blank line to the GULP file...
fout << endl;
// Add the endforce line to the GULP file...
fout << "endforce " << setw(8) << EndForce << endl;
//---------------------------------------------------------------------------
// Here we add the tethered atoms to the file to be sent to GULP...
if( TetherSurfaces ) // Only add these lines to the file if the user requested...
        {
        fout << endl; // Add a blank line...
        for( index = 0; index < 10000; index++ )
                {
                // If there are no additional atoms to tether...
                if( TetheredAtoms[index] == -1 ) index = 10000; // Jump out of for loop...
                else // Else continue to add tethered atoms...
```

```
                              {
                              fout << "tether " << setw(8) << TetheredAtoms[index] << endl;
                              }
                      }
              }
        //--------------------------------------------------------------------------------
        // Write a blank line to the GULP file...
        fout << endl;
        // Add these remaining lines to the GULP file...
        fout << "brenner 1" << endl;
        fout << "timestep " << setw(8) << TimeStep << endl;
        fout << "equilibration " << setw(8) << Equilibration << endl;
        fout << "production " << setw(8) << TotalProductionTime << endl;
        fout << "sample " << setw(8) << MovieSampleRate << endl;
        fout << "ensemble nvt .166" << endl;
        fout << "temperature 300" << endl;
        //--------------------------------------------------------------------------------
        // Finally add the MovieFileName we are going have GULP output to...
        fout << "output xyz movie " << "Movie" + filename + ".xyz" << endl;
        //--------------------------------------------------------------------------------
        //--------------------------------------------------------------------------------
        TotalAtoms = Total_Si_Atoms + Total_C_Atoms;
        //--------------------------------------------------------------------------------
        // Close up the file we are going to send to GULP...
        fout.close();
        //--------------------------------------------------------------------------------
        //--------------------------------------------------------------------------------
        //--------------------------------------------------------------------------------
        //--------------------------------------------------------------------------------
        // We are now going to create the Batch job for this test...
        //--------------------------------------------------------------------------------
        // To create a Batch Job File...
        if( AddCNT ) // only if a CNT was added...
                {
                cout << endl; // Add another line space to the screen...
                //--------------------------------------------------------------------------------
                // Get user defined number of lattice cells...
                cout << "Enter the hours of Wall time for MaryLou (e.g. 10): ";
                cin >> WallTime;
                filename1=filename + ".job";
                fout.open(filename1.c_str());
                fout << "#!/bin/sh" << endl;
                fout << "#PBS -l nodes=1:ppn=1,walltime=" << WallTime << ":00:00" << endl;
                fout << "#PBS -N " + filename << endl;
                fout << "#PBS -m abe" << endl;
                fout << "#PBS -M adan@email.byu.edu" << endl;
                fout << "NP=1" << endl;
                fout << "# cd into the directory where I typed qsub" << endl;
                fout << "cd $PBS_O_WORKDIR" << endl;
                fout << endl;
                fout << "# Run" << endl;
                fout << "mpiexec gulp <" + filename + "> " + filename + ".log" << endl;
                fout << "exit 0" << endl;
                fout.close();
                } // End - Batch Job File...
        // End of the program...
        return 0;
}       //--------------------------------------------------------------------------------
```