The Pennsylvania State University
The Graduate School

MULTISCALE MODELING OF CANCER CELL ADHESION

A Thesis in
Bioengineering
by
Julie Marie Behr

Submitted in Partial Fulfillment
of the Requirements
for the Degree of
Master of Science

May 2013

The thesis of Julie Marie Behr was reviewed and approved* by the following:

Cheng Dong
Distinguished Professor of Bioengineering
Department Head of Bioengineering
Thesis Co-Advisor

Robert Kunz
Professor of Aerospace Engineering
Senior Scientist, Applied Research Lab
Thesis Co-Advisor

William Hancock
Professor of Bioengineering

*Signatures are on file in the Graduate School.

Abstract

The work described in this thesis is one component of a larger-scaled group effort to create a simulation tool that can represent populations of cell types and substrates moving through a flow field and find the probability of aggregates of cells forming. Specifically, this project seeks to define the biochemistry between a circulating melanoma cell and an adherent neutrophil (PMN, in particular), and how the forces acting on the melanoma cell from the surrounding fluid, ability of the cell to deform, and adhesion to the PMN affect the mechanisms leading to melanoma cell metastasis. Metastasis is the process by which a cancerous cell leaves a primary tumor site somewhere in the body, travels through the vasculature, and eventually leaves the vasculature to start a secondary tumor at a distant site. To attempt to define the factors enabling a melanoma cell to metastasize, this biochemistry simulation tool will define the adhesion molecules present on the melanoma and PMN cell surfaces, and calculate their interactions as the melanoma cell moves past the PMN in a flow field. Based on the proximity of the two cells, it may be possible for molecular bonds to form, which apply an adhesive force to the tumor cell. Within the structure of a computational fluid dynamics solver, information is available to define many locations across each cell surface, where individual molecules can be simulated. Within this work, a model will be defined for determining the biochemical rates of reactions between individual molecules, based on their local individual characteristics. All models and computational routines will be made robust enough to allow for future modifications and additional considerations to be incorporated into the model, including an unlimited number of adhesion molecule types to considered, and the ability to redefine the values of parameters to represent different cell types, rather than specifically melanoma cell and neutrophils. For the sake of this model, it is assumed that the neutrophil has already adhered to the endothelial surface, and therefore selectins (which aid in the initial interaction between white blood cells and the endothelium) were not simulated. The melanoma cell in the model expresses ICAM-1 molecules on its surface, and the PMN expresses the β-2 integrins LFA-1 and Mac-1. The PMN is fully rigid, but the melanoma cell is able to move with six degrees of freedom. Computational fluid dynamics is performed using the in-house developed software NPHASE, in which routines describing the biochemistry have been embedded. The biochemistry routines contain both adhesion, representing the ability of molecules to bond and adhere the cells to each other, and repulsion, which represents microvilli, which are not explicitly modelled, pushing the cells apart. Although this project considered melanoma cells and white blood cells, the routines developed in this work can be applied to any cell type of interest, by redefining values of the input parameters. These routines make it possible to run computational fluid dynamics software that incorporates three-dimensional interactions of biological cells.

Table of Contents

List of Figures

List of Tables

List of Symbols

ΔG	Free energy of a system, units of Joules
ΔG^0	Standard free energy of a system, units of Joules
R	Gas constant, 8.314 Joules / mol Kelvin
K	Equilibrium constant, unitless
k	Linear spring constant, Repulsion Model, units of Newtons/meter
b	Nonlinear spring constant, Repulsion Model, units of Newtons/meter
d	Distance between two faces, used throughout, units of meters
k_{on}	Affinity of a molecule to form a bond, Adhesion Model, units of 1/second
k_{on}^0	Affinity of a molecule to form a bond under equilibrium conditions, Adhesion Model, units of 1/second
k_{off}	Affinity of a molecule to break a bond, Adhesion Model, units of 1/second
k_{off}^0	Affinity of a molecule to break a bond under equilibrium conditions, Adhesion Model, units of 1/second
A_L	Contact area, Adhesion Model, units of meters
n_L	Number of molecules within a surface area, Adhesion Model, unitless
n_B	Number of bound molecules within a surface area, Adhesion Model, unitless
s_{ts}	Spring constant of unbound molecules, Adhesion Model, units of Newtons/meter
λ	Critical distance, Adhesion Model, units of meters
k_b	Boltzmann constant, Adhesion Model
T	Temperature, Adhesion Model, used throughout as 310Kelvin
n_I	Number of molecules within a surface area on opposite cell, Adhesion Model, unitless
P	Probability, Adhesion Model, unitless

Δt	Timestep, used throughout, units of seconds
F_b	Bond force, Adhesion Model, units of Newtons
s	Spring constant of bound molecules, Adhesion Model, units of Newtons/meter
ϵ	Critical distance, Repulsion Model, units of meters

Acknowledgments

I would like to first thank Dr. Cheng Dong, for the many opportunities he has given me, and for constant guidance, support and encouragement. To Dr. Rob Kunz, without whom not a single line of code would have been written, thank you for providing the means, plans, and sense of direction that made all of this work feasible. Also, to Dr. Will Hancock, a member of my thesis committee, thank you for your contributions to the finalization of this thesis and for your assistance in seeing it come to fruition.

I would also like to mention all members of the Applied Research Lab and thank them for both their technical input and friendship, in particular Byron Gaskin, my partner throughout this project, Sean McIntyre, who was solicited for his computer programming expertise on numerous occasions, and Chuck Ritter, the savior of lost files and technical difficulties; as well as all members of the Cellular Biomechanics Lab, including in particular Dr. Changliang Fu, who provided invaluable knowledge to the biological aspects of this project, and Dr. Meghan Hoskins, whose doctoral work comprises the foundation for everything that I have created here.

"Essentially, all models are wrong, but some are useful." - George E.P. Box

Chapter 1

Background

1.1 Cancer and Melanoma

A neoplasm, or new growth of cells, takes place when a group of cells have undergone a variety of genetic mutations. These mutations may, at first, include the ability to replicate in the absence of growth signals, and the ability to ignore signals prohibiting further replication. In general, an initial mutation will take place in a single cell, and will be replicated in all subsequent daughter cells. In an individual daughter cell, a secondary mutation may occur, which will likewise be passed on to all descendants of that cell, and so on until a heterogeneous population of cells, a "neoplasm" or "tumor", has formed.

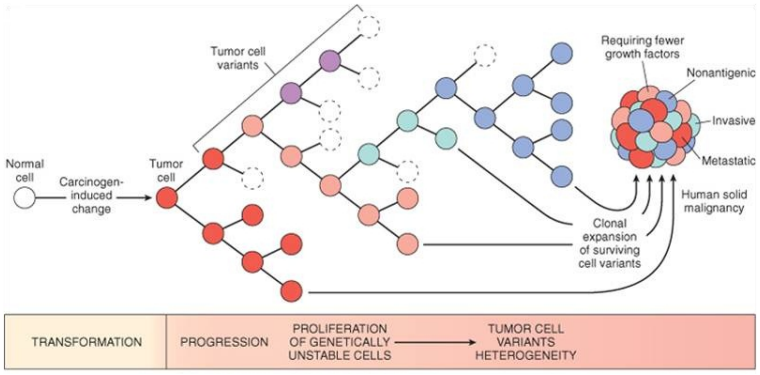


Figure 1.1: Mutations throughout a cell line can form a heterogeneous tumor mass. [1]

This neoplasm might be considered either benign or malignant, as determined by the behavior of the cells as a result of their accumulated mutations. A benign tumor, in general, is considered less potentially dangerous to its host, as it has undergone fewer genetic modifications. These tumors may have developed some combination of the seven mutations which characterize cancer.

A benign tumor has developed the ability to mimic normal growth signalling, through hypersensitivity to ambient growth factors, or some other alteration to the signalling pathway.

Coinciding is an insensitivity to growth inhibitors. Tumor cells can disrupt the pathways leading from antigrowth receptors, allowing them to ignore signals from surrounding normal tissue telling them not to reproduce. These two modifications allow the tumor cells to proliferate at a faster rate than normal, but in order to support a growing neoplasm the rate of attrition must be simultaneously decreased, resulting in a net gain of mass. Some of the mutations sustained by the growing tumor will allow the cells to avoid programmed cell death known as apoptosis, through similar mechanisms of corrupting signalling pathways. These three mutations promote tumor growth through cell-to-cell interactions, but are not sufficient to generate expanding cell populations. Tumor cells must also disrupt the internal systems which cause cells to enter a period of non-activity known as "senescence" after a certain number of replications. Finally, in order to support a growing neoplasm, the tumor must provide its cells with nutrients via sustained angiogenesis. Cells cannot survive beyond 100microns from the nearest blood vessel, and therefore tumor cells must be able to generate their own vascular supply in order to grow to a significant size. These mutations alone will result in a benign neoplasm, for which treatment is often surgical and the entire tumor mass can be removed. However, following the accumulation of these characteristic mutations, some cells within the neoplasm may also develop the ability to invade surrounding tissues. These cells can change the expression of proteins which bind them to surrounding cells and the local extracellular matrix. When cells are able to invade surrounding tissues, they will migrate to either blood or lymphatic vessels and thereby be transported to distant sites within the body, where they can settle and begin a new colony of cells. This ability of tumor cells to migrate throughout the body is known as metastasis, and is the key characteristic which classifies a neoplasm as malignant rather than benign. Malignant neoplasms are what we commonly know as cancer (Hanahan and Weinberg, 2000).

Cancer is responsible for 1 of every 4 deaths in the United States. There were approximately 1.6 million new cases diagnosed in 2012, and nearly 580,000 deaths. Melanoma accounted for 76,000 of these new cases, and 9,200 deaths. Roughly 56,000 cases of melanoma in situ (meaning in place or benign) were also diagnosed in 2012 (Siegel, 2012).

Melanoma is the cancer of melanocytes, the pigment producing cells of the skin. It is the most dangerous type of skin cancer, and the leading cause of death from skin disease (NIH, 2012). Melanoma is characterized by three primary risk markers in an individual. The most common of these markers is the presence of nevi on the skin, which give rise to approximately 30 percent of melanomas. Nevi are irregularly shaped and colored lesions of the skin, and are particularly threatening if they are markedly growing or changing. Another risk marker in a patient is having a family or personal history of melanoma. Finally, sun exposure (both acute and chronic) can provide DNA damaging radiation, with indicators including freckles, skin damage and sunburn. Many melanomas start out as benign, with a nearly 100 percent success rate of treatment, but 70 percent have already progressed beyond this stage by the time of diagnosis. Melanoma provides a model example of the stepwise progression of tumor cells through the process of metastasis which can be extrapolated to other forms of cancer (Elder, 1999).

1.2 Metastasis

As previously described, metastasis is the translocation of tumor cells from the primary cancer site to a distant site somewhere else in the body. A specific cascade of DNA mutations must take place in order to make metastasis possible for an individual tumor cell. Research has shown that only about 0.01 percent of cancer cells have the ability to metastasize after leaving the primary tumor, making it particularly interesting to determine the mechanisms by which those select few circulating tumor cells are able to translocate throughout the body. This highly selective process will result in the most severely mutated, most malignant cells being the basis of a metastatic cell population, meaning this secondary tumor has the potential to be much more damaging than the first. In general, the basic process of metastasis is the same for all cancer types, and can be broken down into six overall steps: (1) local invasion and detachment, (2) dissemination, (3) arrest, (4) extravasation, (5) angiogenesis, and (6) growth and development of metastasis (Dudjak, 1992). The first step is achieved by altering the cell-to-cell adhesion molecules which would normally adhere cells of the same tissue (or in this case, tumor cells within the primary tumor) to each other. Before an epithelial tumor has breached the basement membrane of that tissue, it is considered benign, or in situ.

However, the basement membrane in the neighborhood of a tumor is likely to be damaged or compromised, which allows for tumor cells to escape into surrounding tissue, especially when combined with their increased invasive capabilities caused by characteristic malignant mutations. Migrating tumor cells may choose to take advantage of the lymphatic system or vasculature for transportation throughout the body. Even after the tumor cells have reached the vasculature, only a small percentage (less than 1 percent) will survive to become a metastasis. Circulating cells can be killed by immune cells, insufficient oxygenation, and shear forces from the blood flow. Aggregation with blood cells may enhance the ability of tumor cells to survive, and therefore is a valuable point of analysis.

Beyond surviving their travel through the vasculature, tumor cells must successfully find a stopping point for arrest and extravasation. (Dudjak, 1992). Tumor cells use the same mechanisms of adhesion as normal cells. They may take advantage of the same mechanisms that lymphocytes use under normal conditions to attach to and pass through the endothelial wall. In particular, LFA-1 (lymphocyte function-associated antigen) is an integrin-type cell surface glycoprotein molecule which are believed to be involved in many cellular interactions between tumor cells and normal cells (Nicolson, 1989).

Polymorphonuclear neutrophils (PMNs) comprise 50-70% of all circulating white blood cells, or leukocytes. Studies have shown that inflammatory signals have enhanced the ability of circulating melanoma tumor cells to extravasate. Under some circumstances, PMNs will actually enhance metastatic capabilities, where in most cases they are cytotoxic to tumor cells. The correlation between inflammatory signals and increased melanoma cell metastasis implies that the circulating melanoma cells, rather than being killed off by the immune system like many types of circulating tumor cells would be, are able to take advantage of the immune system and use

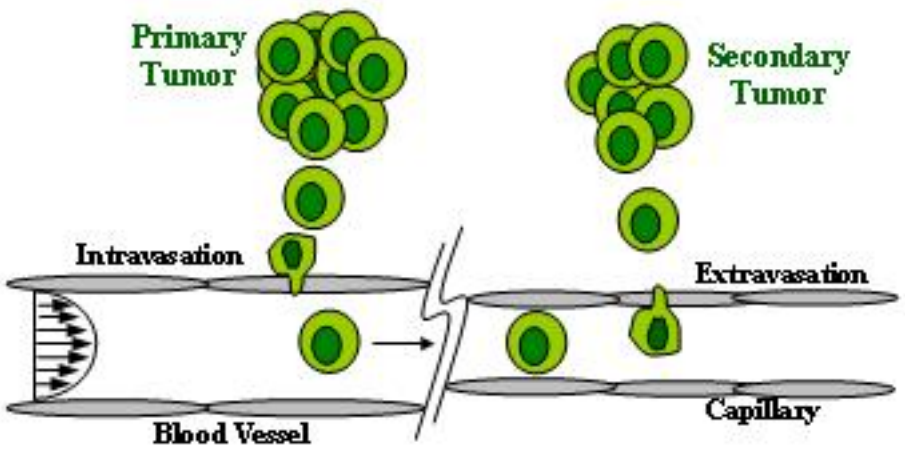


Figure 1.2: Process of cancer metastasis: specific mechanisms remain unknown. [2]

white blood cells to assist their extravasation mechanisms. In a study by Dong et al. in 2005, it was determined that, under flow conditions and in the absence of PMNs, metastatic melanoma cells were no more likely than non-metastatic melanocytes to bind to the endothelium. With the addition of PMN, melanoma extravasation increased significantly. This process was dependent on both the molecular interactions between the melanoma cells with the PMNs, and the PMNs with the endothelial cells. Both melanoma cells and endothelial cells express ICAM-1 (intercellular adhesion molecule) on their surfaces. PMNs express β-2 integrins, including Mac-1 and LFA-1. Interactions between LFA-1 and ICAM-1 aid in the initial capture of a white blood cell to the endothelium, and interactions between Mac-1 and ICAM-1 aid in stabilizing the adhesion even in shear-flow conditions. Blocking ICAM-1 expression on the melanoma cells and separately on the endothelial cells both resulted in a significant decrease in the number of melanoma cell extravasations.

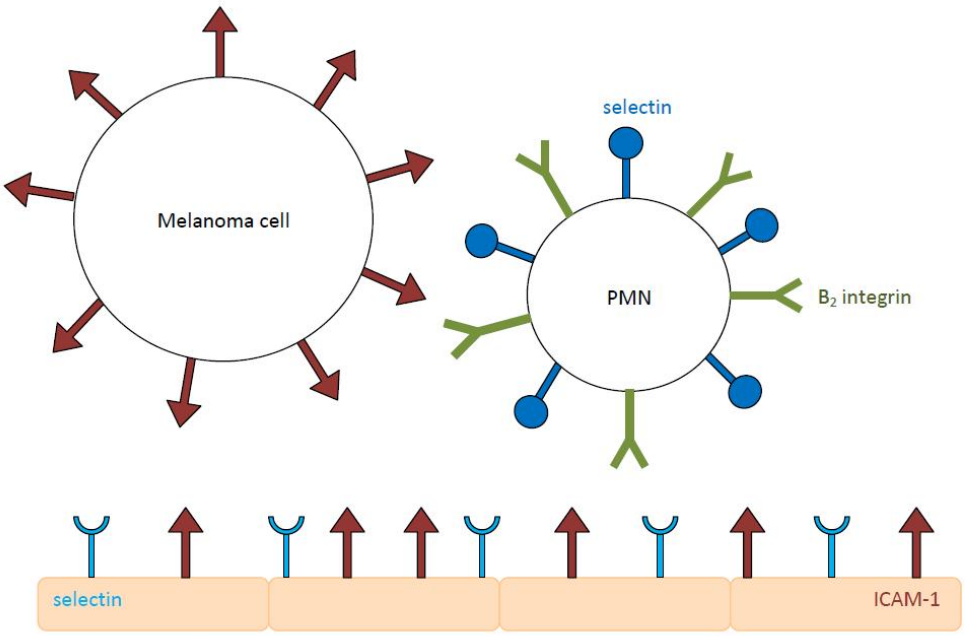


Figure 1.3: β-2 integrins expressed on the PMN can interact with ICAM-1 expressed on both the melanoma cell and endothelial cells. The PMN and endothelium can also interact through selectins.

This work will seek to recreate the effects of these adhesion molecules on the ability of a melanoma cell to extravasate. By applying calculations representative of the behaviors of these molecules, replication of the end result (i.e. a simulation which sees similar extravasation patterns to the experimental data) will imply that the interactions and forces mediated by these three adhesion molecule types are sufficient in characterizing melanoma cell extravasation. Selectin molecules on the endothelium have also been shown to be highly significant in the ability of melanoma cells to extravasate, presumably because initial interaction between a PMN and the endothelium is mediated by selectin interactions, before the stronger ICAM-1 to β-2 integrin binding takes place. The simulation represented in this thesis assumes a PMN which is already adherent to the endothelial wall, and therefore the step of the process requiring the activity of selectins has not been directly modeled. However, further work which seeks to determine extravasation as characterized by initial circulation of all cell types (for instance, assuming that no cells have yet bound to the endothelium) must include calculations representative of selectin interactions as well.

1.3 Adhesion Models

Adhesion molecules on cellular surfaces follow all the same thermodynamic laws which govern any chemical reaction. According to the second law of thermodynamics, a reaction will only

process, and likewise a bond between two adhesion molecules will only form, if it results in a net increase in the disorder of the universe, which is the sum of the internal disorder to the system and the external disorder of the surroundings. From the second law, a quantity describing the change in free energy, ΔG, of a system can be derived. Reactions which occur spontaneously must have a negative ΔG, and a more negative value of ΔG indicates increased likelihood that a reaction will take place. The value of ΔG is determined by the concentrations of all reactants and products within the system, and the energy stored within each molecule. To directly determine the likelihood of a particular type of reaction taking place, rather than a system with certain concentrations, the standard free energy change, ΔG^0, is used. This value is determined for many molecular interactions through large amounts of empirical data.

A particular system's ΔG can be calculated from ΔG^0 as:

for the reaction

$$Y \Longleftrightarrow X \tag{1.1}$$

$$\Delta G = \Delta G^0 + RTln\left(\frac{[X]}{[Y]}\right) \tag{1.2}$$

where R is the gas constant and T is absolute temperature. If ΔG is initially negative, the reaction will proceed from Y to X, resulting in a change in their relative concentrations, which will make ΔG less negative, slowing down the rate of the reaction over time. Eventually, the products and reactants will reach concentrations such that $\Delta G = 0$, and the net reaction will cease to take place. Individual molecules may continue to undergo either the forward or reverse reaction, but the concentrations of X and Y will remain constant, at a state of equilibrium. The ratio of product concentration to reactant concentration at this state is known as the equilibrium constant, K.

$$K = \frac{[X_{eq}]}{[Y_{eq}]} \tag{1.3}$$

In a more complex reaction, K requires the concentrations of all products and reactants. For instance,

$$AB + C \Longleftrightarrow A + BC \tag{1.4}$$

$$K = \frac{[A][BC]}{[AB][C]} \tag{1.5}$$

Based on equation (1.2), if [X]/[Y] = K when $\Delta G = 0$, then the equilibrium constant K is related to the standard free energy change ΔG^0 as

$$\Delta G^0 = -0.616lnK \tag{1.6}$$

or

$$\Delta G^0 = -1.43logK \tag{1.7}$$

where it can clearly be seen that, if ΔG^0 can be determined from existing empirical data, the value of K can be easily determined as well. Because at equilibrium the forward and reverse

reactions are both taking place at the same rate, K is equal to the ratio of the association rate, k on, to the dissociation rate, k off, as seen in Figure 1.4 below.

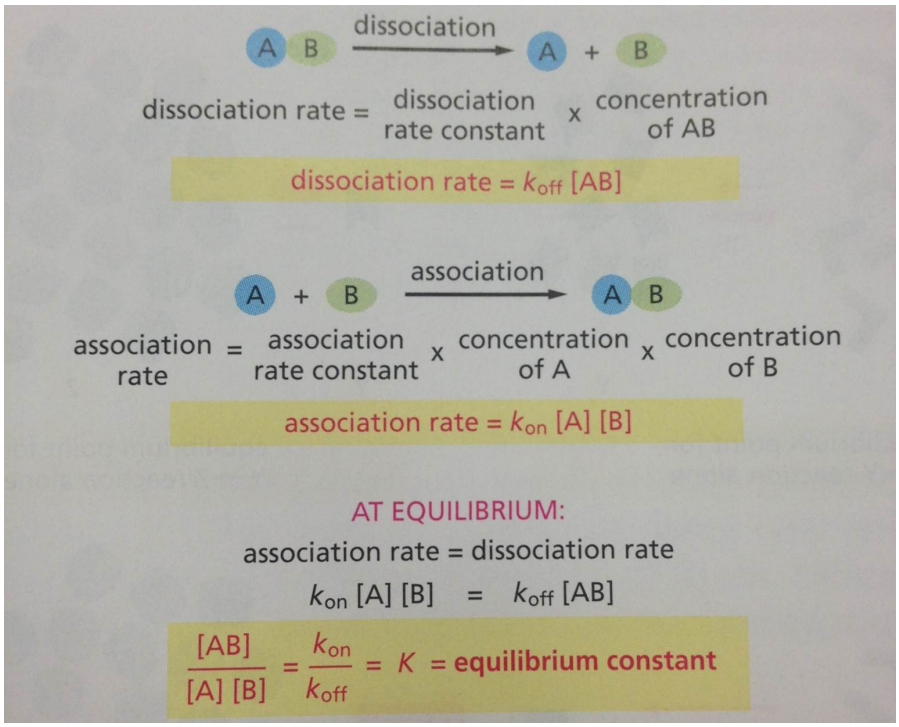


Figure 1.4: Association and Dissociation rates and the Equilibrium Constant. [3]

Adhesion between two cells will depend on the total force acting on those cells. For instance, it is much more likely that two cells will become adherent if there are no external forces present, than if there are strong external forces pushing the two cells apart. Therefore, the forces that can be applied by bonds directly affect affinity. The association rates represent the likelihood of a bond forming, starting from an unbound state when there would not be a bond force, so only the dissociation rate is affected by bond strength. The dissociation rate of a given system will differ from the intrinsic dissociation rate (k^0_{off}) by an exponential factor of the applied force. It is assumed that the interactions between adhesion molecules take the form of a linear spring, and the force on the bond will increase as the bond length deviates from its equilibrium length. Throughout this project, it is assumed that the distance between the cell surfaces at the location of the adhesion molecules is equivalent to the bond length. The dissociation rate k_{off} can therefore be defined as

$$k_{off} = k^0_{off} exp\left(\frac{s(d-\lambda)^2}{2k_bT}\right) \tag{1.8}$$

where s is a spring constant representative of the bond force, $d-\lambda$ represents the displacement of the bond length from the equilibrium bond length, k_b is the Boltzmann constant and T is absolute temperature (Orsello et al, 2001). The dissociation rate of a particular complex of two

bound adhesion molecules is not dependent on the concentrations of molecules or complexes on either cellular surface, because each dissociation event is independent of others. The two molecules involved have already been specified, by nature of being included in the complex. However, when considering the association rate for one particular molecule, or its likelihood to enter into a bond, the concentration of molecules on the opposite cell that are available to bond with becomes very important. Where the dissociation rate alone is a function of the bond force, the association rate is a function of molecule concentrations. Defining n_A as a number density of molecules per surface area, and A_c as the area of the portion of the surface being considered (in most cases, the contact area between two cells), k_{on} is modified for particular circumstances as

$$k_{on} = k_{on}^0 * n_A * A_c \tag{1.9}$$

The likelihood of an individual molecule to form a bond must be dependent on the proximity of that molecule to an available molecule on the opposite cell. In defining the bond force, after a bond has been formed, the idea of an equilibrium distance for bond formation has already been discussed. This concept suggests that there is an ideal separation distance for two adhesion molecules, at which the two molecules could bond without applying a force to either cell. This distance must be the most likely for bond formation, but it is intuitively unrealistic to assume that a bond will only form if the molecules are separated by exactly this distance. Alternatively, another spring constant is introduced, representing the ability of the molecules themselves to deform and deviate from their equilibrium length, to form bonds at separation distances varying slightly from the equilibrium distance. Defining the molecule deformation spring constant as s_{ts}, the modified calculations for the association and dissociation rates become, respectively

$$k_{on} = k_{on}^0 * n_A * A_c * exp\left(\frac{-s_{ts}(d-\lambda)^2}{2k_bT}\right) \tag{1.10}$$

$$k_{off} = k_{off}^0 exp\left(\frac{(s-s_{ts})(d-\lambda)^2}{2k_bT}\right) \tag{1.11}$$

For the purposes of this project, it is important to note that the distance, d, defined in these equations is intended to represent the distance from a microvillus tip to an opposing surface, which in this case is also a microvillus tip (Hammer and Apte, 1992). Therefore, when taken into consideration with the formation of the Repulsion model, which simulates microvilli by applying a repulsive force to areas of the cell surface that come too close together, the calculated distance between two cell surfaces should subtract the length of both microvilli before being compared to the critical adhesion distance λ.

1.4 Computational Fluid Dynamics (CFD) Modeling

Computational Fluid Dynamics is used to describe the behavior of an entire flow field. For this project, commercial software (including Pointwise, ANSYS ICEM CFD, and OpenFOAM) has

been used to create a geometric model of the experimental system, and break it down into a mesh. Boundary conditions are applied to each surface of the geometric model, to define the behavior of the fluid interacting with a wall or surface, and the effects of the inlet and outlet.

Throughout this project, the in-house developed software NPHASE has been used as the CFD solver. NPHASE, like all CFD software, finds solutions to Navier-Stokes fluid mechanics equations at every location within the entire simulated domain that is explicitly identified by the mesh. The equations being evaluated by the CFD solver can be broken down into groups of fundamental equations representing fluid properties. The first set of equations, known as conservation equations, are as follows:

$$\frac{\delta\rho}{\delta t} + \frac{\delta\rho u}{\delta x} + \frac{\delta\rho v}{\delta y} + \frac{\delta\rho w}{\delta z} = 0 \tag{1.12}$$

$$\frac{\delta\rho v_i}{\delta t} + \frac{\delta\rho v_i v_j}{x_j} + \frac{\delta\rho}{\delta x_i} = 0 \tag{1.13}$$

$$\frac{\delta(\rho h^0 p)}{\delta t} + \frac{\delta\rho h^0 v_i}{\delta x_i} = 0 \tag{1.14}$$

These equations are known as the continuity, momentum, and energy equations, respectively. The indexes i and j represent the x, y and z components of vector quantities. These equations represent all fluid types, and can be modified into the complete viscous form that is known as the Navier-Stokes equation. To fully define the system, the CFD solver also needs to compute constitutive relations that define the specific fluids properties. These relations are as follows:

Stagnation enthalpy:

$$h^0 = h + \frac{v_i}{2} \tag{1.15}$$

Gibbs function:

$$h(p,T) = gTg_T \tag{1.16}$$

$$g = g(p,T) \tag{1.17}$$

Density:

$$\rho = \frac{1}{g_p} = \rho(p,T) \tag{1.18}$$

In the general case, with these two sets of relations, there are 9 equations and 9 unknowns, meaning a unique solution can be found at every mesh face location so long as all other conditions have been met.

In this project, the flows under consideration are incompressible. Accordingly, density is constant, and the energy equation does not need to be solved. To run the simulation, creating a mesh and running NPHASE are controlled with a script written in Python (entire script can be seen in Appendix D). A mesh is created primarily in Pointwise, and is controlled by a few user-controlled parameters in the script. The mesh will contain a rectangular domain defined by the overall x-, y- and z-dimensions, a spherical tumor cell and an adherent deformed white blood cell. The radii of both cells, as well as their initial centroid locations, can be dictated by the

user. The script will build a mesh, run NPHASE, and run post-processing programs that allow the output of NPHASE to be visualized via video files. For the purposes of adding biochemistry, CFD provides a geometric location for every mesh face that falls on the surface of both cells in the simulation. It is this geometry that is taken advantage of for the purposes of adding simulated adhesion molecules, and allowing them to have individually defined local behaviors.

1.5 Multi-Scale Population Model

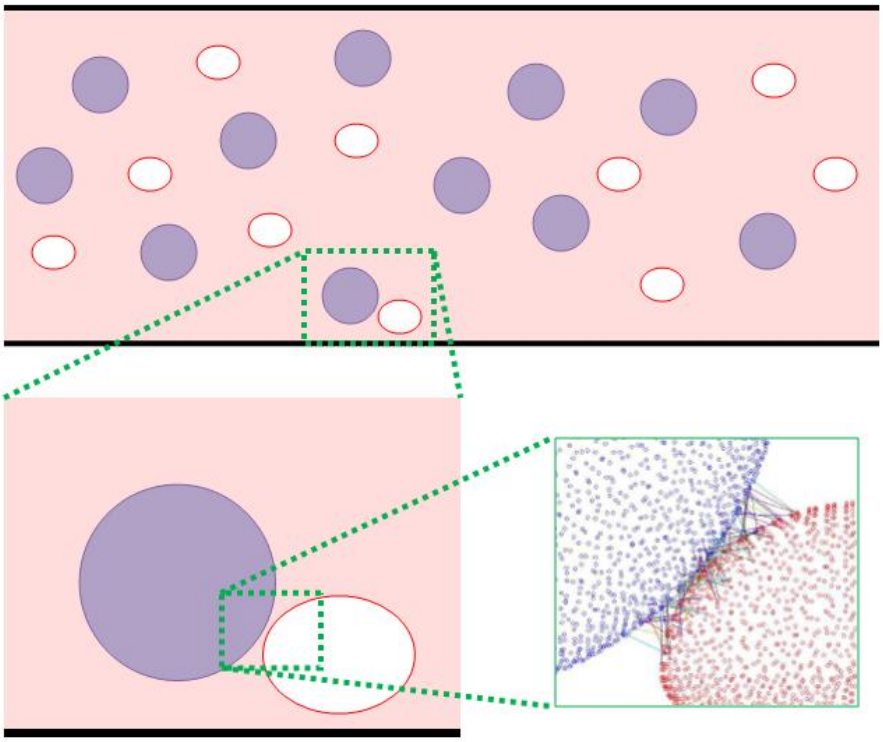


Figure 1.5: The simulation tool represents interactions on a macroscopic level, cellular level, and molecular level.

The work described in this thesis is one component of a larger-scaled group effort to create a simulation tool that can represent populations of cell types and substrates moving through a flow field and find the probability of aggregates of cells forming. The model is currently built to represent an experimental flow chamber, featuring the geometry of this chamber and the fluid flow profile that corresponds to it rather than that of a blood vessel, which allows for direct experimental confirmation of observed outcomes and can later be modified to represent a blood vessels geometry, making the extrapolation that if the applied conditions were able to match a test chamber with corresponding geometric constraints, then the same set of conditions will similarly accurately define the properties of cells and fluid flow in a blood vessel if the geometric constraints are appropriately changed. The computational model is meant to have the ability

to simulate cells, proteins, and any other substances present in the blood, to determine via a stochastic model how these cells would be expected to react and behave. The model should be robust enough to accept as an input any combination and concentration of cell types and other solvents, defined by their relevant properties. Each cell type will be characterized by its surface molecule properties, deformability, geometry and concentration. On the largest scale of the model, populations of cell types will be cycled through the simulated blood flow, and interactions between different cells and macromolecules will be captured. For each determined interaction, the subroutine described in this thesis will be called to calculate the local interactions between those two particular cells, temporarily ignoring the rest of the population.

Once the overall model has moved to the cellular scale, it calculates the interactions between two cells in terms of the fluid dynamics of the blood flow moving past and between them, the biochemical kinetics of adhesion molecules between the two cells if approach sufficiently close to each other to form contact, the membrane mechanics of the deformability of the cells, and full six degree of freedom motion. All of these forces that act on the two cells are interdependently coupled. Fluid dynamics will determine the most governing cell motion, and forces from the fluid will likely attempt to separate the two cells should adhesion occur. Adhesion between the cells will cause bond forces to act on the cell surfaces at the point of the bonded molecules, which will then also have an effect on the overall motion of the cells. As the cells bond and are pushed into each other from the flow, they will likely deform and the contact area between the two cells will change, which enables more adhesion molecules to potentially form bonds, and also changes the geometry of the cells that affects the resultant flow profiles and hydrodynamic forces. Therefore, in a computer simulation of the entire process, all three sources of forces are considered at every time step interval, and compiled into an overall sum of forces that dictates the motion and deformation of the cell, which will then be iteratively introduced as the initial condition for the next time step.

The biochemistry subroutine transforms the two solid objects that currently exist in the mesh into living, interacting cells. Without biochemistry of surface molecules, the model could be simulating inorganic substances, which can be pushed by the flow and deform on contact but otherwise have no interaction with each other. To define the cells, it is necessary to introduce simulated adhesion molecules on their surfaces, enable those molecules to form bonds with each other, and incorporate forces on the cell surfaces resulting from those bonds, as well as to consider the surface topography of the rough membranes.

The biochemistry sub-routine features one primary routine in which the cell types are defined and characterized, and two sub-functions that then calculate adhesion and repulsion forces based on the input parameters. Cell repulsion is characterized by the simplification in simulating cell membranes as smooth that would actually have many folds and microvilli. Conceptually, once the smooth cell models are within a certain significant distance from each other, the real cells would have already had some sort of contact between microvilli that would impede their continued movement into each other. Rather than creating a grid that models the geometry of each microvillus, which would be inefficient and unnecessary, they are modeled as a non-linear

spring that repels the cells when their surfaces are within the critical distance. Cell adhesion is modeled as a probability model of bonds forming between adhesion molecules on the portions of the cell membranes within a critical bonding distance from each other, and the attractive forces pulling along those adhesion molecules should a bond form. Specifically, this project seeks to define the biochemistry between a circulating melanoma cell and an adherent neutrophil (PMN, in particular), and how the forces acting on the melanoma cell from the surrounding fluid, ability of the cell to deform, and adhesion to the PMN affect the mechanisms leading to melanoma cell metastasis. To attempt to define the factors enabling a melanoma cell to metastasize, this biochemistry simulation tool will define the adhesion molecules present on the melanoma and PMN cell surfaces, and calculate their interactions as the melanoma cell moves past the PMN in a flow field. Based on the proximity of the two cells to each other, it may be possible for molecular bonds to form, which apply an adhesive force to the tumor cell. For the sake of this model, it is assumed that the neutrophil has already adhered to the endothelial surface, and therefore selectins (which aid in the initial interaction between white blood cells and the endothelium) were not simulated. The melanoma cell in the model expresses ICAM-1 molecules on its surface, and the PMN expresses the β-2 integrins LFA-1 and Mac-1. The PMN is fully rigid, but the melanoma cell is able to move with six degrees of freedom. Computational fluid dynamics is performed using the in-house developed software NPHASE, in which routines describing the biochemistry have been imbedded.

The biochemistry component of the multi-scale model allows for the possibility of two cells interacting with each other and becoming bound. The relative velocity between these two cells approaches zero, and they will move through the blood flow, or remain adhered to the wall, as a single unit. Studies have shown that melanoma cells are more likely to extravasate out of a blood vessel in the presence of white blood cells, which suggests that this ability of the tumor cells to aggregate to PMNs is an essential step of the metastasis process. Therefore, we are interested in seeing the probability of aggregates forming between two cells, based on input population parameters regarding the concentrations and characteristics of different cell types moving through the vessel or test chamber. This probability of aggregation is the ultimate output of the multi-scale model.

1.6 Overview

The organization of this thesis is as follows. The repulsion and adhesion biochemistry routines will be described separately. Chapter 2 focuses on the Repulsion model, including the physical basis of the model, the equations used to represent repulsive forces, implementation of those equations into the CFD software, and results and discussion. Chapter 3 focuses on the Adhesion model, and will similarly cover the physical basis of the model, equations used to represent binding and unbinding probabilities and bond forces, how those equations were incorporated into the software, and results and discussion. Chapter 4 details how both models were numerically tested and analysed, to verify that the routines were calculating repulsive and adhesive forces as

intended. Finally, Chapter 5 discusses how the biochemistry routines can be verified or modified with empirical data, and outlines future directions for this project.

Chapter 2

Repulsion

2.1 Introduction

An artificial repulsion force is applied to the surface of the melanoma cell as it approaches the white blood cell to represent the normal forces applied by contacting microvilli on the surfaces of real cells.

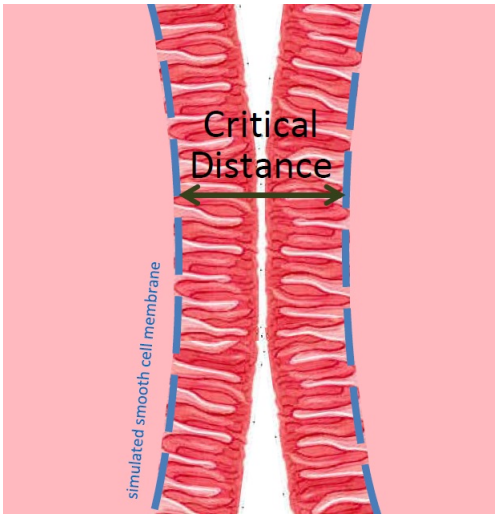


Figure 2.1: Microvilli on real cells will be coming into contact with each other when the simulated smooth cell surfaces are within a critical distance of each other. Adapted from [4]

Microvilli and their behavior therefore must be quantified by characteristic parameters. The critical distance seen in Figure 2.1 is based on the average length of microvilli, which is 0.5 μm on a neutrophil (Hammer and Apte, 1992).

2.2 Model

Cellular repulsion as modeled as a non-linear spring force. As the cells approach each other within a defined critical distance, they will experience a repulsive force, defined with respect to the separation distance as

$$Force = k * distance + b * distance^3 \tag{2.1}$$

where k and b are spring constants representing the microvilli (Bongrand and Bell, 1984). The values of these two parameters have been initiated to k = $-110 * 10^{-6}$ Newtons/meter and b = $600 * 10^6$ Newtons/meter^3 to maintain a cell surface separation of approximately 0.3 μm, but these values must be calibrated to match empirical findings through future experimentation. Because cells are deformable, this force is calculated separately between each individual face of the grid lying on the cell membrane. The faces that are actually approaching contact will, in this simulation and realistically, experience a much greater repulsion force than faces on the trailing edges that do not interact with the opposite cell to a significant extent. Allowing each individual face to experience a force allows the cell to deform, which is defined separately in a different routine. Within the critical distance, the cell will experience a repulsion force with a magnitude dependent on the separation distance as seen below in Figure 2.2.

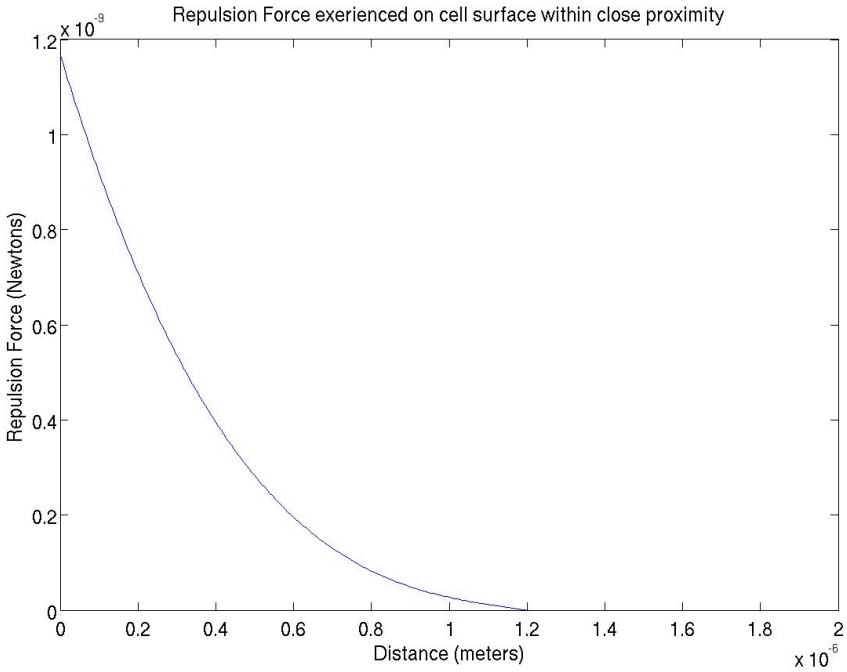


Figure 2.2: Magnitude of the repulsion force that is applied to the cell surface with respect to the separation distance between two cell faces

2.3 Implementation

To calculate repulsion, the distance between every face on cell 1 and every face on cell 2 is first calculated, creating a 2-dimensional array that has “number of faces on cell 1 rows and “number of faces on cell 2 columns.

Tumor Cell Face Identification Number	PMN Face Identification Number: 1	2	3	4	...	n
1	$distance_{1-1}$	$distance_{1-2}$	$distance_{1-3}$	$distance_{1-4}$	...	$distance_{1-n}$
2	$distance_{2-1}$	$distance_{2-2}$	$distance_{2-3}$	$distance_{2-4}$	...	$distance_{2-n}$
3	$distance_{3-1}$	$distance_{3-2}$	$distance_{3-3}$	$distance_{3-4}$	...	$distance_{3-n}$
4	$distance_{4-1}$	$distance_{4-2}$	$distance_{4-3}$	$distance_{4-4}$	...	$distance_{4-n}$
5	$distance_{5-1}$	$distance_{5-2}$	$distance_{5-3}$	$distance_{5-4}$	...	$distance_{5-n}$
6	$distance_{6-1}$	$distance_{6-2}$	$distance_{6-3}$	$distance_{6-4}$	...	$distance_{6-n}$
...	...	...	...	...	...	...
m	$distance_{m-1}$	$distance_{m-2}$	$distance_{m-3}$	$distance_{m-4}$	...	$distance_{m-n}$

Figure 2.3: Two-dimensional array created by the Repulsion routine representing the distances between all possible combinations of cell 1 and cell 2 faces.

2.3.1 Step 1: Sweep All Faces

A simple function first sweeps every face throughout the grid, and then saves the centroid location to a separate array if that face lies on either cell surface. The mesh faces that correspond to a cell surface have a flag that allows them to be identified. While sweeping through all of the faces, any face that is found to have the cell surface flag causes the implementation of another loop, which will record the x-, y- and z-coordinates of the centroid corresponding to that flagged face, into an array that is specific for which cell type has been identified. There are now six newly defined arrays containing the centroid location in x, y and z of every face on cell 1 and every face on cell 2 respectively.

2.3.2 Step 2: Distance Calculation

Next, a distance calculation is performed between the centroids of every possibly combination of faces from cell 1 and cell 2. To perform this calculation, two more loops are used, one that will go through every face on cell 1 and another within that loop that goes through every face on cell 2. The concept of repulsion is a model to simulate the effects of actual contact between extending microvilli mechanically pushing on each other, and therefore does not need to be considered unless two faces are within the predefined critical distance from each other there is not some negligent amount of force, but rather no force at all outside of this maximum distance. For a given pair of faces, the distance is calculated using the standard distance equation:

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.2}$$

2.3.3 Step 3: Force Calculation

The critical distance represents the sum of the lengths of microvilli from the two cell types. If the distance has been calculated between two particular faces and those faces are separated by a distance greater than the critical distance, the routine moves on to another pair of faces, taking no further action. If two faces have approached each other within the critical distance, a non-linear spring force acting in the direction opposite to a vector connecting their centroids will be imposed. The spring force is modeled as follows:

$$F = kd + bd^3 \tag{2.3}$$

where k and b are previously defined spring constants. For this simulation, the critical distance ϵ = 1.2 microns. Similar to k and b, as stated in section 2.2, this ϵ value was found from literature and must be experimentally confirmed or modified.

2.3.4 Step 4: Force Summation, and Future Use

This force is then split into x, y and z components as follows:

$$F_x = F * \frac{\Delta x}{d} \tag{2.4}$$

$$F_y = F * \frac{\Delta y}{d} \tag{2.5}$$

$$F_z = F * \frac{\Delta z}{d} \tag{2.6}$$

and the components are summed across each face for the total repulsive force acting on it from all faces from the other cell within the critical distance. A total sum is also calculated, so that the effects of the motion of the centroid of cell 1 can be considered. These forces will later be sent to the six-degree-of-freedom and membrane-deformation routines.

Table 2.1: Simulation Defined Initial Parameters

Parameter	Value
Domain Size:	
X	60
Y	32
Z	42
Tumor Cell Centroid:	
X	20
Y	10
Z	21
PMN Centroid:	
X	30
Y	2.5
Z	21
Timesteps	100

2.4 Results

The addition of the repulsion force allows the melanoma cell to flow smoothly over the PMN, where it was previously getting stuck. Without the repulsion routine, the melanoma cells trajectory would begin to rise away from the test section floor, but not quickly enough to clear the PMN. It would run into the side of the PMN, and require infinitesimal time steps to resolve the hydrodynamic forces building up between the two cells. Running the simulation for higher and higher total time steps resulted in the same visualization of the melanoma cell stalling in front of the PMN. With the addition of the repulsion routine, the time steps were able to stay much larger, allowing sufficient time throughout the entire simulation for the melanoma cell to pass over the PMN.

For the sake of direct comparison, the following simulations were run using identical input parameters, and an alteration only between the nphase.dat files. In the first simulation, the python script was edited such that the nphase.dat file would not include the line "employ julie biochemistry", which prevents NPHASE from calling the repulsion routine. For the second simulation, this line was included, meaning this routine is activated and a repulsion force is calculated across the surface of the cell, and added into the net forces acting on the cell. The initial conditions for both simulations are stated in Table 2.1. The tumor cell was intentionally initiated in the immediate vicinity of the white blood cell, so that the effects of the repulsion model would have immediate impact.

Although the two simulations were intended to run for 100 timesteps, without repulsion forces the two cells crash into each other after 82 timesteps, which causes NPHASE to crash in the first simulation. For a direct comparison of the motion of the cell throughout the two simulations, the following figures show the position of the tumor cell initially, after 50 timesteps, and after 80 timesteps. The final figure shows the second simulation, in which the repulsion forces were

activated, at the final timestep, demonstrating clearly that the repulsion force prevented the simulation from crashing.

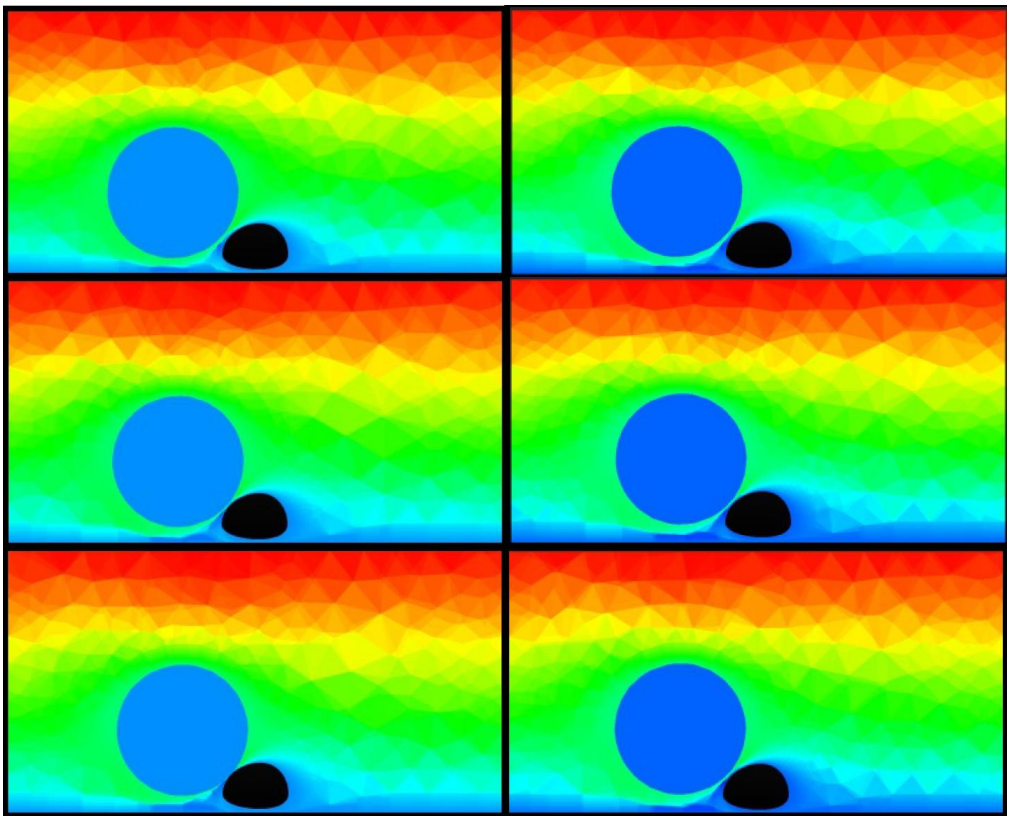


Figure 2.4: Simulation Comparisons with the same initial conditions, with and without repulsion forces. A:Simulation 1, No Repulsion Force, Timestep 0, initial formation. B: Simulation 2, Repulsion Force Activated, Timestep 0, initial formation. Both simulations were initiated in the same positions. C: Simulation 1, No Repulsion Force, Timestep 50. D: Simulation 2, Repulsion Force Activated, Timestep 50. The melanoma cell is being kept further away from the PMN by the repulsion force that pushes them apart. E: Simulation 1, No Repulsion Force, Timestep 80. The simulation did not progress to Timestep 100, because the melanoma cell crashed into the PMN after 82 timesteps, causing NPHASE to fail. F: Simulation 2, Repulsion Force Activated, Timestep 80. With the repulsion force turned on, the melanoma cell is still kept at a distance from the PMN, allowing the simulation to continue.

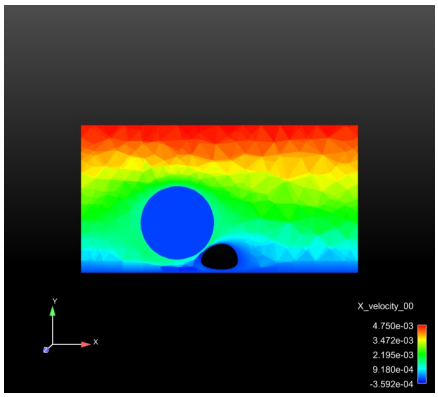
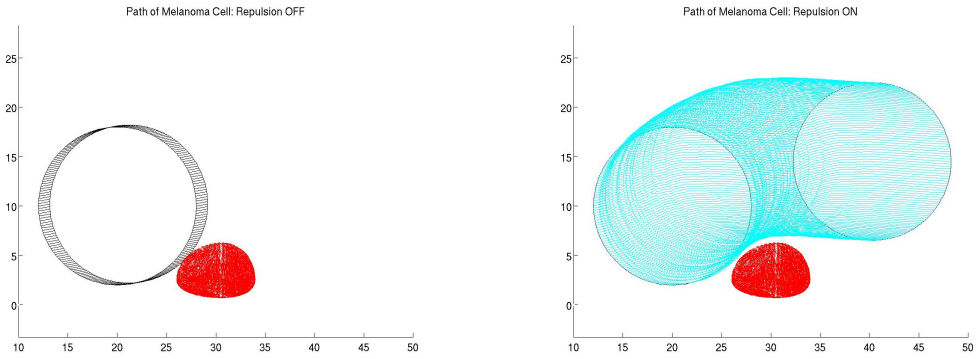


Figure 2.5: Simulation 2: Repulsion, Timestep 100

The repulsion routine allows the melanoma cell to pass over the PMN without crashing into it. Because the parameters defining the spring constants in the non-linear spring model were arbitrarily chosen to maintain a desired separation of the two cells that matches the average separation seen experimentally, imaging of non-adherent cells in a flow chamber should be used to confirm whether real cells follow the same path described by the simulation.



(a) (b)

Figure 2.6: Path of melanoma cell, with same initial position, with and without repulsion. (a) Repulsion OFF: the melanoma cell follows a linear path, and does not avoid the PMN. When the two cells come into contact at Timestep 82, the simulation fails. (b) Repulsion ON: the melanoma cell is able to avoid the PMN, and the simulation continues until the melanoma cell is completely past the PMN.

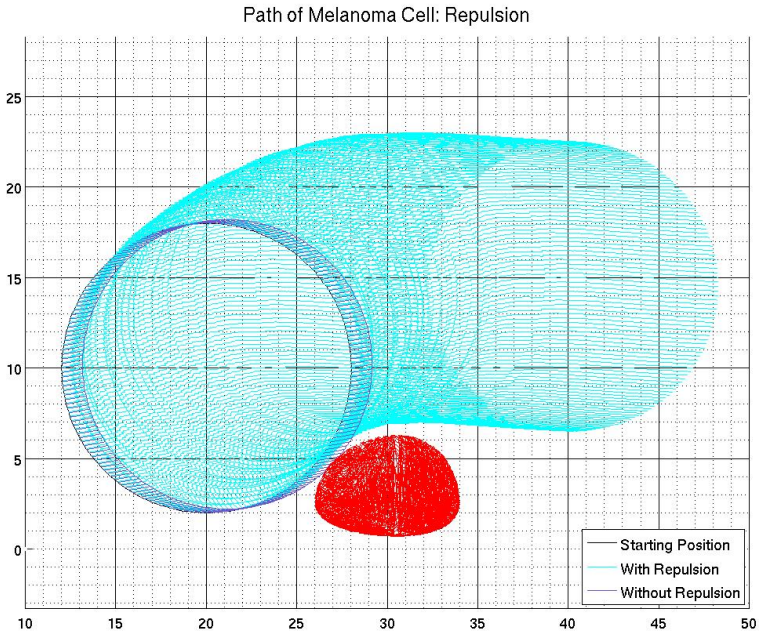


Figure 2.7: Overlay of the path of the melanoma cell with and without repulsion.

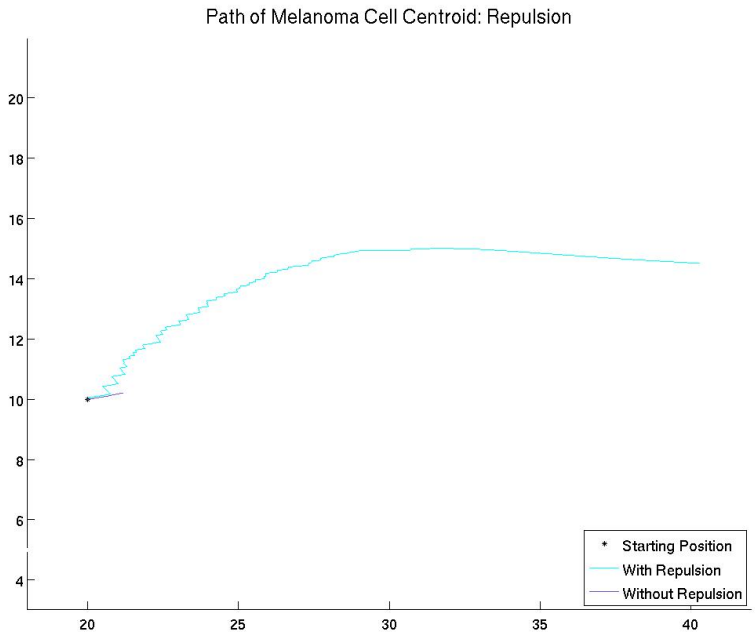


Figure 2.8: Path of the centroid of the melanoma cell, with and without repulsion. Without repulsion, the melanoma cell follows a fairly linear path, unaffected by the PMN. With repulsion, the path of the melanoma cell arcs around the PMN, and is also clearly not smooth. This is the effect of using too large a timestep to fully resolve the smooth continuous motion of the melanoma cell. However, the continuation of the simulation proves that the simulation will not crash even when using larger timesteps.

An update to NPHASE allowed the melanoma cell to remain stationary after contacting the PMN, such that the cell will stop moving but will not crash the system, and the simulation continues. With this modification, the melanoma cell remains stationary for many timesteps, until eventually enough pressure builds up from the fluid behind it to push it over the PMN. Without repulsion, the melanoma cell will roll along the surface of the PMN until it gets to the other side.

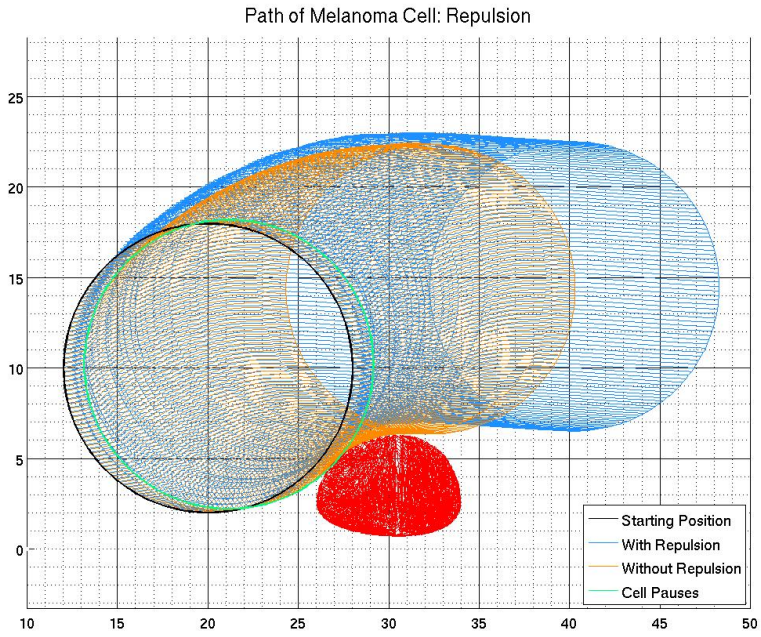


Figure 2.9: Without repulsion, the melanoma cell crashes into the PMN. Here, a modification to NPHASE allows the melanoma cell to remain stationary for many timesteps after it crashes into the PMN, and eventually it begins moving again and rolls over the PMN. it is clear that the melanoma cell without repulsion comes directly into contact with the PMN, while the melanoma cell with repulsion is able to maintain a consistent gap. (Data for simulation without repulsion interpreted from Ensight visualization.)

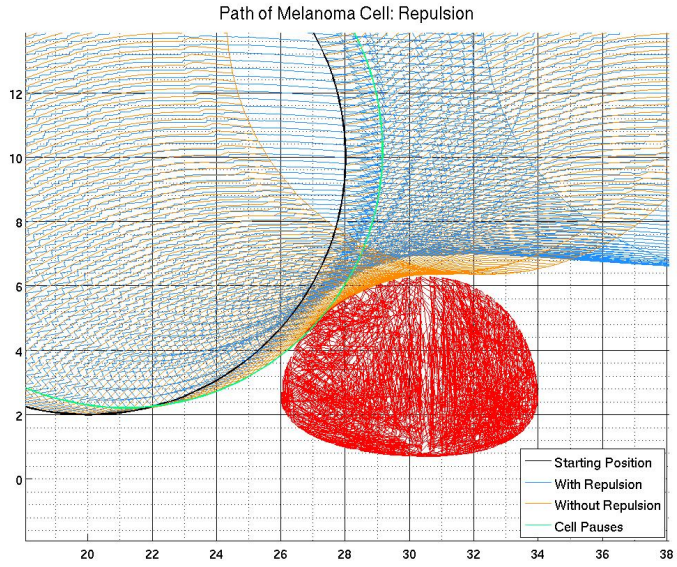


Figure 2.10: Close-up of paths of the melanoma cell, with and without repulsion.

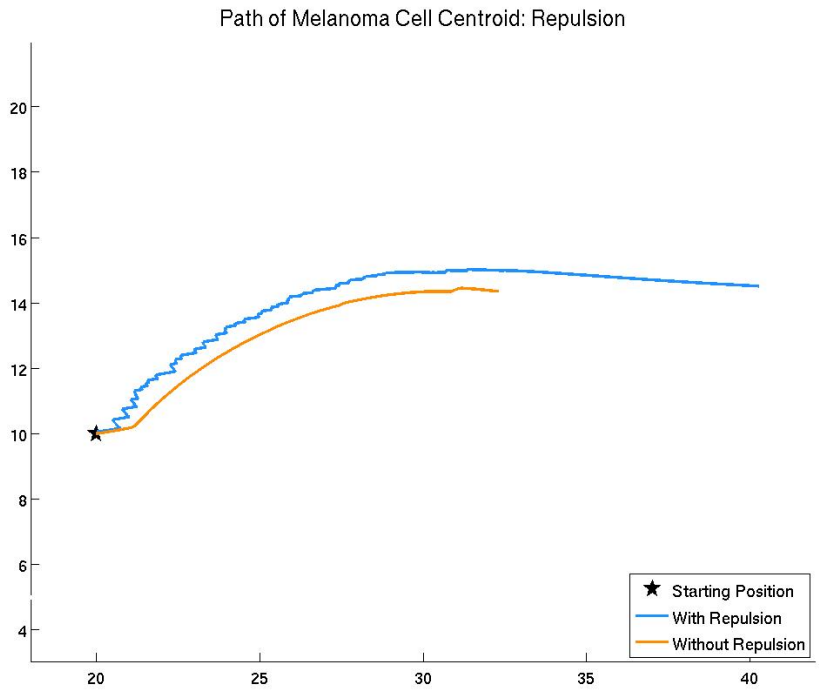


Figure 2.11: The centroid paths of the melanoma cell, with and without repulsion. (Data for simulation without repulsion interpreted from Ensight visualization.)

2.5 Discussion

As previously stated, there are three parameters within this model that have been assigned values that need verification. Experimental data should be provided to confirm the values of k, b and ϵ.

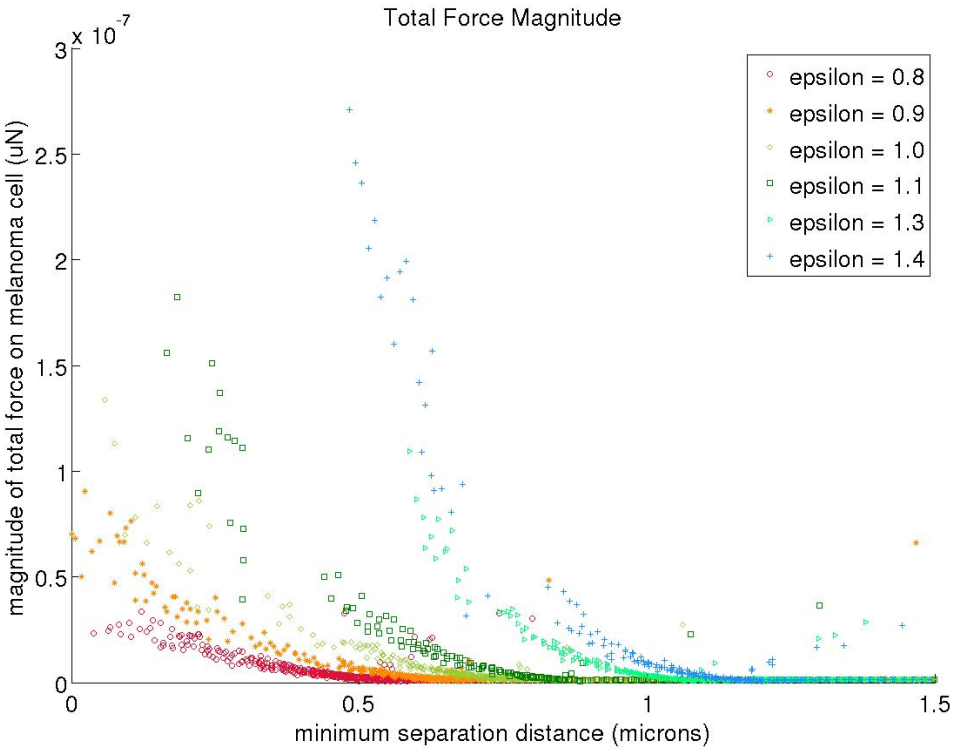
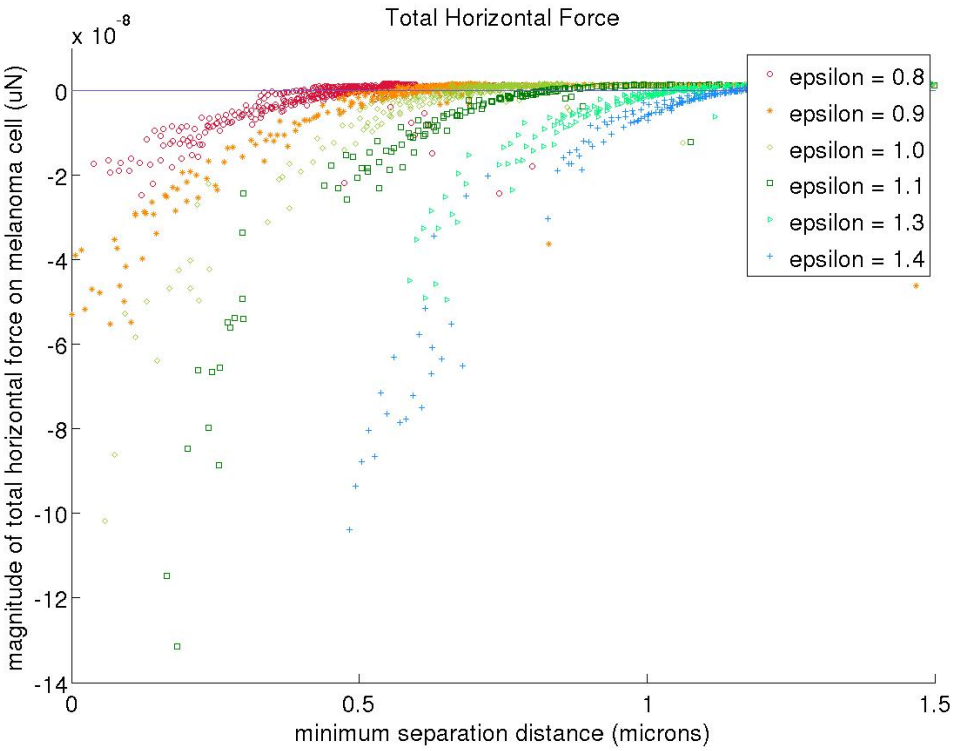
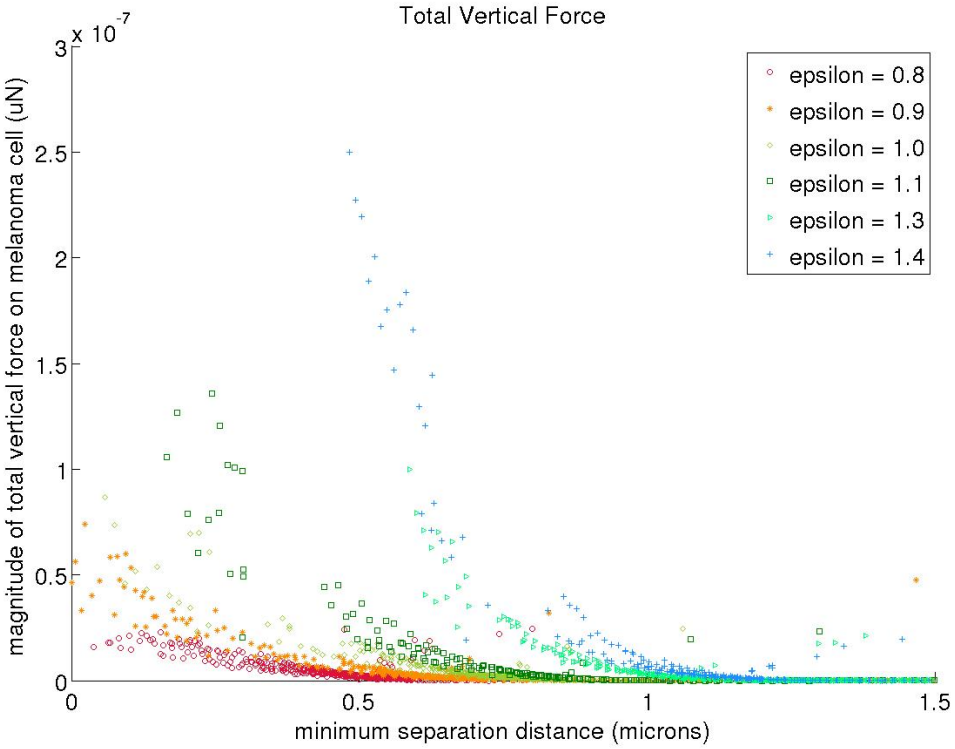


Figure 2.12: Increasing the value of ϵ causes the repulsion forces to activate while the cells are further apart. With a larger ϵ, cells will experience the same pattern of repulsion at a greater minimum distance from each other. Therefore, varying the value of ϵ will affect the minimum approach distance between the two cells. At a greater ϵ value, the cells will maintain a larger separation distance. Some scatter can be seen in this plot because the cells are not both spherical, so the minimum separation distance between them does not dictate the total surface area within the critical repulsion distance. Empirical data will determine what the separation distance should be, and ϵ can therefore be modified to match the desired separation.

The formulation of this model assumes that, because the repulsion model represents existing contact between microvilli surfaces, the mesh surfaces of the cells should never approach each other within a slight deviation from the critical distance ϵ, assuming sufficient timestep resolution to allow the force to prevent their proximity. However, this means that when the mesh surfaces are 1.2 microns apart, the cells in reality have parts of their surfaces that come in contact with each other. This would mean that any adhesion molecules on the microvilli surfaces should be able to interact with each other. If the repulsion force succeeds in preventing the cells from approaching within 1.2 microns of each other, the mesh surfaces will never be close enough to be within the critical separation distance for the adhesion model, which is much less. Consideration must be given to whether the distance between the mesh faces, as calculated by the adhesion model, needs to somehow be modified to account for the repulsive simulation.



(a)



(b)

Figure 2.13: Total force acting on the melanoma cell, broken into (a) the horizontal component of the forces (in general, repulsion is pushing the melanoma cell in the upstream direction of the flow, which results in a negative force), and (b) the vertical component of the forces (pushing the melanoma cell up over the PMN).

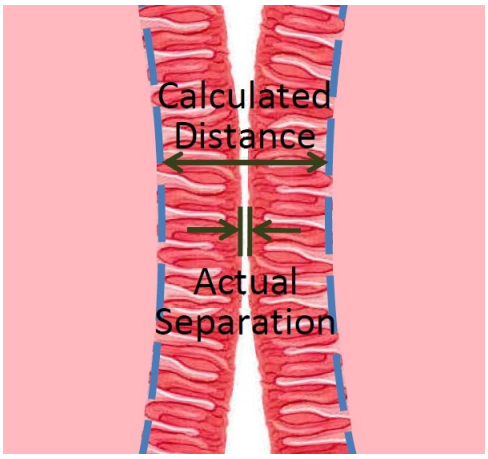


Figure 2.14: The repulsion force will prevent the cell surfaces from ever reaching the critical adhesion proximity. Adapted from [4]

Output files from Ensight that include the effects of the repulsion model show clear jumps in the position of the tumor cell. When the tumor cell suddenly approaches the white blood cell too closely, a large repulsive force is calculated, which causes the cell to be pushed backward quickly in the next timestep. This is an effect of using too large of a timestep to get adequate resolution on the movement of the cell, which would be continuous and smooth in reality. Also, because the cell cannot currently deform, forces that actually would be acting on a local area of the cell surface are resulting in a net movement of the cell. These repulsive forces are more likely to force the contacting faces to flatten, rather than the entire cell to alter its path. It is likely that if the cell surfaces could deform, the localized retraction of an area of the cell's surface would correct for the artificial jumping of the entire cell that is currently seen.

Chapter 3

Adhesion

3.1 Introduction

It would, theoretically, be possible to model every adhesion molecule as the sum of its compositional amino acids, and the attractive or repulsive force fields emitted by each component governing the relations between molecules on each cell. New research efforts that focus on individual proteins may attempt to define the molecule this explicitly. For the scale of our simulation, this would be both unnecessary and prohibitively expensive in terms of computational power and time. However, existing adhesion models reflect only the overall behavior of all molecules within the cellular contact area, and therefore to identify the behaviors of specific molecules at certain locations across the membrane surface an extrapolation of existing models must be designed and implemented.

What we currently know is that, assuming all coefficients have been appropriately calibrated, at the cellular level we expect adhesion to follow the pattern

$$k_{on} = A_L(n_L n_B)k^0_{on} exp\left(\frac{-s_{ts}(d-\lambda)^2}{2k_bT}\right) \tag{3.1}$$

where each molecule has this affinity to bind to an opposite molecule. We would expect empirical data to reflect this affinity as the average affinity for each molecule within the contact area. However, we can assume observationally that some molecules, for instance those towards the center of the contact area, which are surrounded by a greater number of molecules on the opposite cell within their reach, will have a higher affinity to bond than others. We therefore want to suggest some calculation that will give a personalized affinity value to each molecule, while the overall average of all affinities throughout the contact area still approaches the affinity value defined by the currently accepted adhesion model, which we will refer to as the global model or affinity value. We do not currently possess experimental techniques capable of determining the likelihood of each individual molecule to form a bond, or to correlate the bonding of each

individual molecules distance to the opposite cell to its binding ability. We can, however, assume that an averaged value as represented by the global model would be representative of each individual molecule if each individual molecule had identical properties with respect to their locations on the cell. Therefore, we want to use the same calculation as the global model to determine an affinity value for each individual molecule, but use localized versions of each constant a local distance to the opposite cell, and a local number of adhesion molecules on the opposite cell within reach of the molecule, for instance. In this way, we maintain the existing adhesion model that has been widely accepted and base our modifications only on the idea that the existing model is accurate. The localized affinity values will be much smaller than the global value, because they include the greatly reduced contact area within the reach of that individual molecule rather than the total area of cell membrane within a certain critical distance from the other, so they will each be corrected by a factor such that the average local affinity will be equal to the global affinity. This calculation is therefore very simple in terms of the necessary coefficients, as the numbers can be determined from empirical global data (which is very easy and well established to gather) and the localized values will, by definition of the nature of the calculation, average out to match the experimentally observed results. Assuming that each molecules local affinity would behave following this model assumes that the global affinity calculation is an accurate representation of a uniform contact area. We do not have any further evidence to support the idea that the distribution of local affinities would be proportionate to the local topographical variations.

3.2 Model

The previous model for bond formation (in Meghan's thesis) used global values for the system to define affinity and separation distance, and therefore assumed that each molecule within the defined contact area had an equal probability of forming a bond. Contact area was defined as a circular region with a radius equal to the surface protrusion length, or the length of roughness elements on the cell surface, of the cancer cell. Distance between the two cells was found based on the separation of their centroids. The value of affinity, k_{on}, for each molecule on the cancer cell was then calculated once for the entire system, using:

$$k_{on} = A_L(n_L n_B)k_{on}^0 exp\left(\frac{-s_{ts}(d-\lambda)^2}{2k_bT}\right) \tag{3.2}$$

Where (n_L - n_B) represents the total number of available (unbound) molecules (either Mac-1 or LFA-1) within the contact area on the PMN surface, k_{on}^0 is the rate of bond formation under equilibrium conditions, s_{ts} is a spring constant of the adhesion molecule being represented as a linear spring, and λ is the equilibrium length of the bound molecules. The number of potential bonds is taken to be all the possible combinations between each molecule on the cancer cell and every molecule on the PMN within the contact area, and therefore a total number of potential bonds was found using:

$$N = (n_L A_L)(n_I A_L) \tag{3.3}$$

The model then found N random numbers, and compared them against the probability calculated from kon to determine whether or not a bond formed. Each molecule on the cancer cell had an equal probability of forming a bond with some molecule on the PMN, regardless of the local distance of that molecule to the PMN surface.

The new model will take advantage of our knowledge of each face location, and will calculate a localized bond formation probability for each adhesion molecule on the cancer cell. For the first iteration of the simulation, the biochemistry routine will first sweep through all of the faces in the grid, and save arrays containing specifically the area and x, y and z coordinates of the centroid of the faces on the cancer cell and PMN, separately. It will then calculate the distance from each face on the cancer cell to every face on the PMN. If this distance between two particular faces is less than λ, a local k_{on} will be calculated that represents the affinity of the adhesion molecules on that cancer cell face to bind to the molecules on that particular PMN face. For this local k_{on}, A_L is the area of the PMN face, (n_L - n_B) is the number density of available molecules on that face only, and d is the local distance that has just been calculated. In this model, we assume that the adhesion molecules are uniformly distributed across the cell surface. We also use the assumption from Dembo et al, 1988, that the molecules are fixed within the plane of the cell membrane, and not able to diffuse laterally throughout the membrane. Per Simon and Green, 2005, we can assume that our time scales are small enough to ignore any cell-mediated changes in molecule expression. Because we are looking at a much smaller area of potential bond formation for each molecule, this local k_{on} must be corrected by some factor such that the average local k_{on} and the k_{on} computed using the global method (as in the previous version of the model) are the same value. To allow for this, after calculating the local value of k_{on} for all of our cancer cell faces, we will multiply each local k_{on} by (global k_{on})/(average local k_{on}).

To find the probability of bond formation, we will again sweep through all of the cancer cell and PMN faces, and for every non-zero value of k_{on}, probability is calculated using (Migliorini et al, 2002):

$$P = 1 - exp(-k_{on} * \Delta t) \tag{3.4}$$

A random number between 0 and 1 will then be generated, and if its value is less than P, a bond has been formed. The random number generator makes this model inherently probabilistic, rather than deterministic, and should yield similar but not identical results if run multiple times. This formulation was chosen to account for the inherent randomness of an actual biologic system and our inability to deterministically define the behaviors of the molecules with absolute certainty. Pertinent information about this bond (the types of molecules, face ids for both the cancer cell face and PMN face that have been bonded, and the current distance between those two faces) will be saved to an array.

Once all the potential bonds have been tested for bond formation, forces from each existing bond will be calculated. This force calculation will use the same model as the previous biochemistry routine (Hoskins, 2008), which models the bonded molecules as linear springs (Hammer and

Apte, 1992), using:

$$F_b = s(d - \lambda) \tag{3.5}$$

Where the force acts along the vector connecting the centroids of the two bonded faces. The x, y and z components of each bond force will be calculated, and the total x y and z forces and total x y and z torques will be output to the 6DOF routine. Note that, while the previous routine considered only one type of adhesion molecule per time step, this routine will consider all molecules every time step by repeating for each type of adhesion molecule on the cancer cell. The array containing information on existing bonds will be written out to a file at the end of the routine.

For every subsequent time step, the routine will first read in the existing bond file, and calculate the probability of each of those bonds breaking, based on the new positions of the bonded faces. Because k_{off} has always related specifically to one individual bond and the length of that bond only, the same calculations can be used as the old model (Hoskins, 2008):

$$k_{off} = k_{off}^0 exp\left(\frac{(s - s_{ts})(d - \lambda)^2}{2k_b T}\right) \tag{3.6}$$

Where s is the linear spring constant of the bond, used to calculate bond force, and k_{off}^0 is the rate of bond breakage under equilibrium conditions. The same probability model is used as for bond formation, and a random number generator determines whether a bond breaks or remains. Bonds that do not break (ie. The random number generated was greater than P of the bond breaking) are re-saved to the array of existing bonds. From here, the routine repeats as previously, with k_{on} calculated for all remaining unbound cancer cell adhesion molecules.

Turning the biochemistry into an iterative routine compatible with the CFD code requires consideration to be given to all of the following areas of concern. We assume that we have sufficient knowledge to track each individual bond on the surface of each cell, and must therefore be able to locate each molecule. Given that our ability to assign a location in space within the simulation is dictated by the faces on the cell surface mesh, we must figure out how to approach the potential issue of having a fractional number of molecules on a given face. Non-integer molecules must either be impossible, which would require some method of rounding, or accommodated, which would require modification of the structure of the random-number-per-molecule methodology. We do not have to track the lifespan of the bonds, because we can assume with the probability model that giving the bond an opportunity to break at each time step invalidates the need to include a longer Δt in the bond breakage probability calculation. We are also assuming that we will be able to keep track of the location of a formed bond through multiple simulation timesteps, which means we will have to be able to locate the face in the mesh corresponding to the bound face through multiple grid generations. Finally, while we know that affinity of each individual molecule will be different across the surface of the cancer cell depending primarily on that molecule's distance to the PMN, we do not know exactly how affinity varies as a function of its position on the cell surface, as we only have an existing model for calculating k_{on}

Table 3.1: Adhesion Molecules

	Molecule	Surface Density	Reference
Melanoma Cell:	ICAM-1	13 x 10^{12} molecules / m^2	Simon and Green, 2005
PMN:	LFA-1	45 x 10^{12} molecules / m^2	Simon and Green, 2005
	Mac-1	5 x 10^{12} molecules / m^2	Simon and Green, 2005

Table 3.2: Adhesion Parameters

	LFA-1 to ICAM-1	Mac-1 to ICAM-1	Reference
k_{on}	3000 1/Ms	3000 1/Ms	Hoskins, 2008
k_{off}	0.3 1/s	0.29 1/s	Hoskins, 2008
s	2×10^{-3} N/m	2×10^{-3} N/m	Hammer and Apte, 1992
s_{ts}	1×10^{-3} N/m	1×10^{-3} N/m	Hammer and Apte, 1992
λ	0.05 μm	0.05 μm	Springer, 1990

based on the average across the entire contact area. We have chosen to use this same calculation but implement a localized contact area and displacement with some correction factor to keep the average k_{on} consistent with what we know it to be, but we do not know if this results in a distribution that is locally accurate.

3.3 Implementation - Five Step Routine

The ultimate output of the biochemistry modeling, within the context of a CFD program, will be the forces that affect the dynamics of every surface within the model. To calculate the forces resulting from adhesion, a five step approach was implemented.

3.3.1 Step 1: Read In Existing Bonds

The first step must be to consider whether there are already bonds existing from before this timestep. This step will be skipped for the first timestep of the simulation. The output file created at the end of the previous execution of the routine will be read in, providing information as to the number of bonds that existed at the previous timestep, and for each bond the type of molecules involved, a face identification number for the bound face on the tumor cell and a face identification number for the bound face on the PMN. The new separation distance between these two faces will be calculated, and the probability for bond breakage calculated using

$$k_{off} = k_{off}^{0} exp\left(\frac{(s - s_{ts})(d - \lambda)^2}{2kbT}\right) \tag{3.7}$$

$$P = 1 - exp(k_{off} * \Delta t) \tag{3.8}$$

A random number is generated for each bond, and if that bonds probability for breaking has a greater value than the random number, that bond is broken. If the probability is less than the random number, then the bond does not break, and the type of molecules and face identification numbers are resaved into an array that will contain information for every bond existing at this timestep. The distance between the two faces is saved into another array, which will be used for calculating the forces caused by each bond.

3.3.2 Step 2: Calculate Preliminary k_{on}

Similar to the Repulsion Model, in this step two "for" loops will sweep through every face on both cells. At every possible combination of a face from the tumor cell and a face from the PMN, the distance between the two faces is calculated. If the calculated distance is less than a critical adhesion value, as defined in the overall biochemistry routine, then a local affinity will be calculated between these two faces. Additionally, in anticipation of the next step, a counter keeps track of the number of times a value for k_{on} is calculated, or the number of face pairs that are within the critical distance from each other. Each k_{on} is added into an overall sum of all k_{on} values. The distance between these two faces is added to a total distance between all critical face pairs. The face area of the tumor cell face is added to a total contact area value.

3.3.3 Step 3: Adjust k_{on}

This step will only run if, in the previous step, the counter representing the number of times k_{on} was calculated is greater than zero, or in essence, if there is at least one pair of faces that are within the critical adhesion distance from each other. The total k_{on} value is divided by the number of k_{on}s, in order to find an average value. Similarly, the total distance between all critical face pairs is divided by the number of critical face pairs to find an average distance. The global value of k_{on} is then calculated using the existing adhesion model (Eq 4.1). We know that this value should, in reality, represent the average k_{on} for all molecules within the contact area. Therefore, we want to multiply each local value of k_{on} by a correction factor that will cause the average and the global values to be equivalent, namely

$$correction = \frac{k_{on,global}}{k_{on,average}} \tag{3.9}$$

The faces will now be swept through again. Again, if a pair of faces is within the critical adhesion distance from each other, they will have a corresponding value for k_{on}. This value is now modified by the correction factor, as well as the ratio of unbound molecules on the PMN face to the total number of molecules on the PMN face, to adjust the affinity to reflect the actual number of available opposing molecules.

3.3.4 Step 4: Calculate Probabilities

Once again using the probability equation

$$P = 1 - exp(-k_{on} * \Delta t) \tag{3.10}$$

a probability is calculated for every relevant face pair. A random number is generated for each molecule on the tumor cell face, and compared against the value of the probability. Because the number of molecules on the face may not be an integer, the last value of a probability (representing the probability that a fraction of a molecule will bond) is modified by the fraction of the molecule. If the probability of bond formation is less than the random number, then a bond has not formed, and the routine moves on to the next molecule. If the probability of bond formation is greater than the random number, a bond has formed. The types of molecules involved, and face identification numbers for each face involved, are saved to the same array used for continually existing bonds from Step 1. The counter tracking the total number of bonds existing in this timestep is increased by one, and the number of available molecules for both the tumor cell face and the PMN face is decreased by one. The distance between the two faces is saved to the second array from Step 1.

3.3.5 Step 5: Calculate Bond Forces

This step of the routine will sweep through every entry of the array containing distances between two bound faces. For each face pair, a spring force will be calculated based on their separation distance, by the equation

$$F_b = s(d - \lambda) \tag{3.11}$$

where s is the spring constant representing the elasticity of the bond and λ is the equilibrium bond length. This bond force is then broken into x-, y-, and z-components using the following equations

$$F_{b,x} = F_b * \frac{\Delta x}{d} \tag{3.12}$$

$$F_{b,y} = F_b * \frac{\Delta y}{d} \tag{3.13}$$

$$F_{b,z} = F_b * \frac{\Delta z}{d} \tag{3.14}$$

that will be sent out to the six degree-of-freedom solver.

3.4 Results

By initializing the tumor cell in close proximity to the neutrophil, it is possible to verify bond formation without running a full simulation, and only considering one timestep. By saving the locations of all pairs of bound faces, Figure 3.1 was generated. In this simulation, the centroid

of the tumor cell was initialized to (20.8, 10, 21) (units in μm), and the centroid of the PMN was initialized to (30, 2.5, 21).

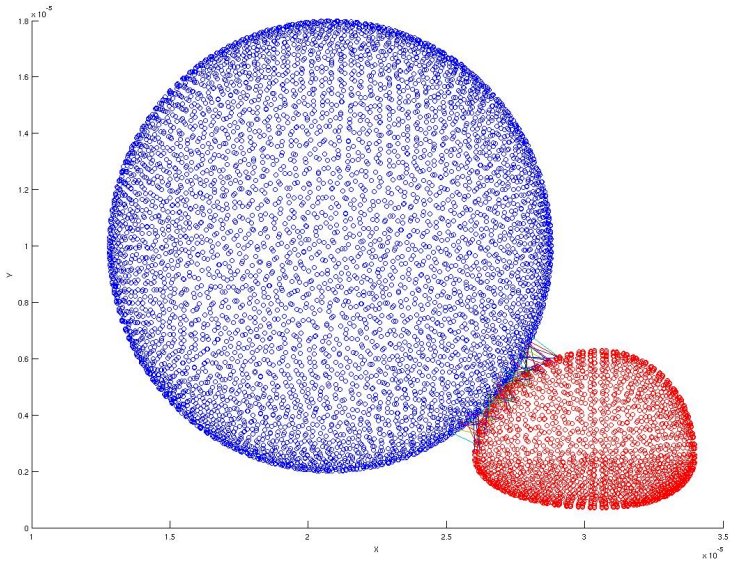


Figure 3.1: The Adhesion routine was modified to write out the locations of both cell surfaces and the locations of each face involved in a bond to files, which could be used to generate a visual representation of the bonds that formed. The blue figure is the melanoma cell, and the red figure is the PMN. The lines connecting them each represent a bond (or multiple bonds between the same faces) that has formed, and is drawn between the two faces on which the involved adhesion molecules reside.

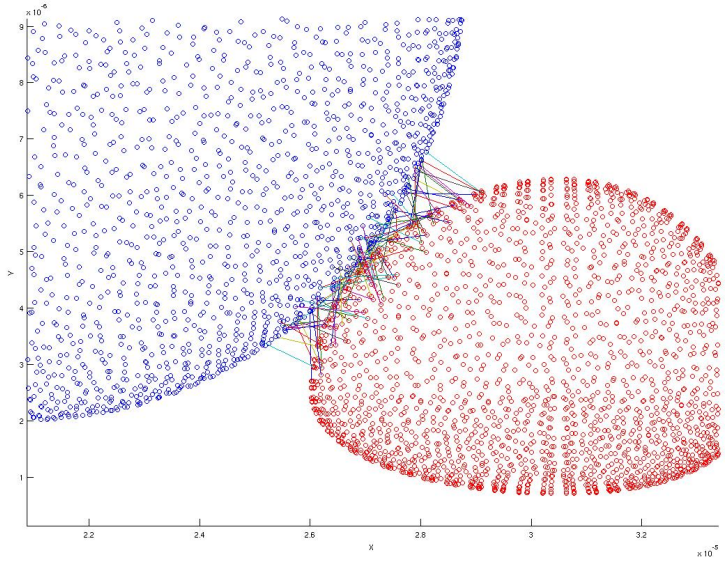


Figure 3.2: Close-up of bonds.

The simulation that generated the bond distribution seen in Figure 3.2 used a tumor cell grid with a maximum element size of 0.5 μm. Every blue circle in the figure represents the centroid of a tumor cell mesh face. A total of 96 bonds were generated between the two cells. To verify whether bonding is dependent on the mesh, the simulation was rerun allowing a maximum element size of 1.5 μm, shown below in Figure 3.3. Although their are fewer pairs of faces involved in bonds, a total of 97 bonds were formed. It can be assumed that the slight variation is due to the stochastic nature of the adhesion routine.

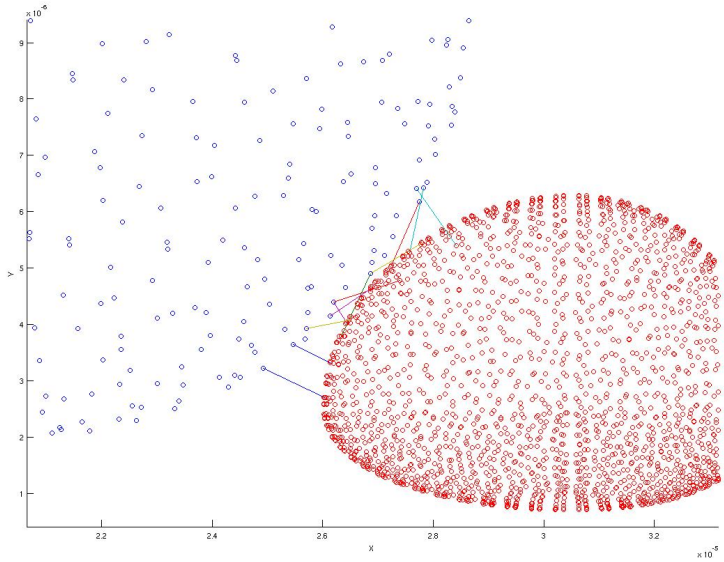


Figure 3.3: Visual representation of bonds formed between the melanoma cell and PMN, using a less refined mesh on the melanoma cell. Fewer faces are bound, but each bound pair of faces are bound by more molecules, resulting in the same total number of bonds between the two cells.

3.5 Discussion

The design of this model was required to address a few challenges outlined at the end of section 3.2 The first of these challenges was to assign a location to every simulated molecule. The assumption first had to be made that adhesion molecules are evenly distributed across the cell surface. Although this may be unlikely, as the molecules are free to move throughout the membrane, we have no empirical data that suggests a non-uniform distribution that we could replicate. Furthermore, for the purposes of our model, adhesion molecules outside of the contact area between the two cells are irrelevant, so we only need verification that adhesion molecules are uniformly distributed within the contact area for our model to be spatially accurate. As previously stated, our ability to assign locations in space within the model is dictated by the mesh faces within the grid. Adhesion molecules are assigned a location based on the number density of molecules across the cell surface area and the area of an individual face on the cell surface mesh. Every molecule on a given face is assumed to be located at the centroid of that face, and so the spatial uniformity of the molecules is dictated by the refinement of the mesh, which corresponds to the accuracy of the CFD as well. There may be a non-integer number of molecules on a given face. If a rounding method had been used, to protect the integer requirement inherent in the random number generator probability assessment, there would be no way to maintain the uniformity of molecules across the surface, and some inherent bias would be given to areas that had been rounded up rather than rounded down. It would also be very difficult to maintain the proper overall number density, and ensure that a proportional number of molecules had been either

rounded up or down such that the total number of molecules on the surface was representative of a real cell. To accommodate for the non-integer number of molecules, the loop controlling the random number generator was modified such that it would calculate a random number for every molecule or fractional molecule on the face. For each full molecule, the random number is compared against the probability value that was found for that face. For the fractional molecule, the random number is compared against a modified probability, which is the original probability value multiplied by the fractional amount of the molecule. For instance, if a given face is found to have 8.3 adhesion molecules on its surface, 9 random numbers will be generated. The first 8 will be compared against the probability value P, and the ninth will be compared against P * 0.3, thus making it less likely but still possible for a bond to form.

The next design consideration was the assessment of the probability model controlling bond breakage kinetics. In general, it is assumed that a given bond has a certain expected lifetime, which makes it more likely for the bond to break after it has existed for a given period of time. This would require our ability to flag each formed bond with an indicator of how many timesteps the bond has existed. However, in general these calculations are only done across the population, and represent the total number of bonds that exist at a certain time. Because we are testing the probability of bond breaking at every timestep, we can assume that each timestep represents an independent event dictated by the same probability of breaking and considering only the amount of time that has elapsed since bond breakage was last calculated, and the increased likeliness of bond breakage is accommodated for by the fact that the bond has had multiple opportunities to break.

Having bonds exist through multiple timesteps means we must be able to record their location within the simulation space. In a single timestep, the location of a bond is defined by the location of the two interacting mesh faces, as has been stated previously, and is easily stored within NPHASE. However, the grid is regenerated every timestep. We must have a way to save the location in space of two bound molecules, without being able to rely on the face identification number. Fortunately, within this simulation the grids representing the surfaces of each cell are only generated once at the very beginning, and we can take advantage of their specific identification system to remain constant throughout all timesteps of a simulation. This requires an extra conversion within the adhesion routine from the overall grid identification system to the non-changing cell surface specific system and back again.

The final consideration was how to calculate k_{on} in a local setting in a way that both takes advantage of the CFD capabilities and remains aligned with empirical data and models from real cells. No model currently exists representing a distribution of affinities across a cellular contact area. As has been previously discussed, the model used here assumes that the existing model for calculating affinity,

$$k_{on} = A_L(n_L n_B)k_{on}^0 exp\left(\frac{-s_{ts}(d-\lambda)^2}{2k_bT}\right) \tag{3.15}$$

can be scaled down to represent individual grid face areas and distances while maintaining the accuracy of the model. The values of k_{on} given by this equation will be proportional to their

actual distribution, and must be adjusted by a global k_{on} value such that the average local value and the global value are equivalent. I developed this theory for finding a non-uniform local affinity distribution, based exclusively on conjecture and the assumption that the existing model is accurate at the molecular (rather than just cellular) scale.

According to the adhesion model equations used here, each molecule has the ability to behave as a spring. The value of s_{ts} represents the ability of a molecule to deform from its equilibrium length, to form bonds with other molecules within a certain range of distances centered around the equilibrium distance (Hammer and Apte, 1992). The computational routine forces the value of k_{on} to zero when the distance is greater than the equilibrium distance, based on the assumption that the molecules cannot possibly reach each other (whereas if the distance is less than the equilibrium distance, molecules could bend toward each other). Because the equations are only meant to mimic molecular behavior and are only approximations, it is possible that a value of k_{on} ought to be calculated at every distance. The value would become prohibitively small for bond formation if the distance is too much greater than the equilibrium distance, but at distances only slightly deviating from equilibrium, k_{on} would be non-zero and bonding would be possible. Experimental data collection will be required to determine whether the current design of the code, which includes the distance cut-off, should be modified.

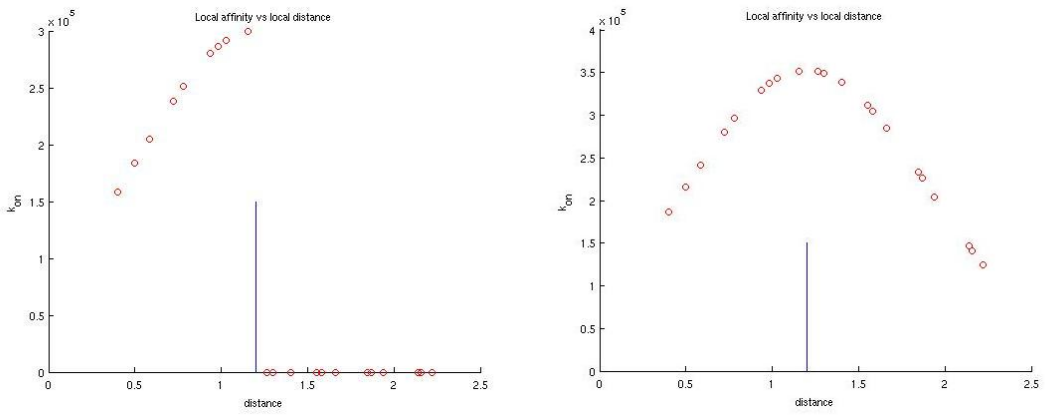
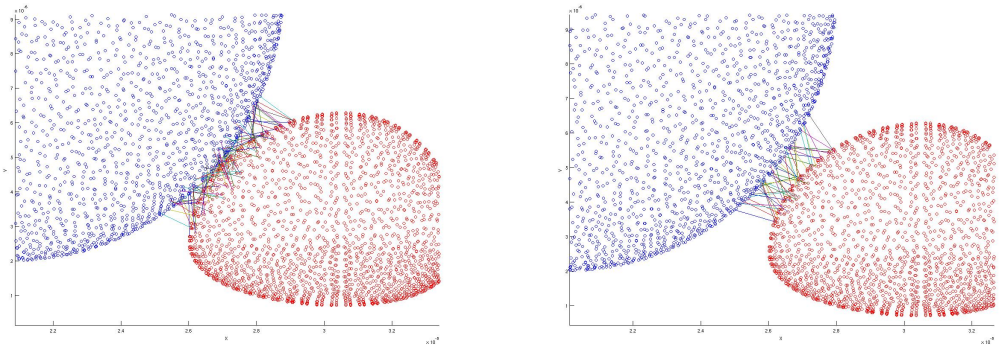


Figure 3.4: The distribution of k_{on} values, at left seen how they are currently calculated with a cut-off at the critical adhesion distance forcing all values beyond that distance to zero, and at right how they would be calculated if this requirement were removed.

Based on the visualizations of bond formation, when the minimum distance between the cells is near the equilibrium distance (i.e, the value of k_{on} between faces in the closest proximity to each other approaches the maximum values of k_{on}), fewer total bonds will form between the two cells but more of those bonds will be approximately normal to the cell surfaces, as compared to when the minimum distance between the cells is near zero. This is because, as the distance between the cells approaches zero, the faces that are closest to each other have a low affinity to bind to each other (per Figure 3.5), but have a high affinity for faces further away. Because the cells are closer, there are more faces on each cell with the ability to bond to a face on the



(a) (b)

Figure 3.5: Bond formation between the tumor cell and white blood cell as the separation approaches 0 or equilibrium adhesion distance. (a) Bond formation between the cells when the separation distance is near 0. More bonds have formed, because there are a greater number of faces within the critical distance from each other, but it is clear that most of the bonds have a bond length close to the equilibrium bond length, rather than forming between faces in the closest proximity with each other. (b) Bond formation between the cells when the separation distance is near the equilibrium bond distance. Fewer bonds have formed, but bonds are more likely to form approximately normal to both cell surfaces.

opposite cell, whether that face is nearest to it or at some radius away from the nearest face.

Chapter 4

Computational Verification

Computational Verification refers to the troubleshooting and fine-tuning of the model to ensure it is actually calculating what it was designed with the intent to calculate. This most frequently entails isolating one small piece of the routine, with an easily calculated and/or predicted outcome, and running the entire routine to verify that the output matches what is expected. The equations themselves must be computationally verified before any significance can be given to the numerical output of the model. In this way, the model is ensured to behave as desired, and is ready to be calibrated versus empirical data to verify the accuracy of input parameters.

4.1 Biochemistry Routine Control of Subroutines

The purpose of the overall routine is that all parameters, and no calculations, can be defined in one specified place. For different cell types, the values assigned to variables within this routine must merely be updated, and assuming multiple adhesion molecule types on each cell surface, the values must be assigned to variable arrays using consistent indexing. By testing the subroutines, and in particular the Adhesion subroutine (assuming the Repulsion Model will see little modification regardless of different cell types), to ensure they are sufficiently robust as to handle changes in sizes of input arrays, this routine must pass a computational verification checkpoint, as it has no actually computations within itself. That the input remains usable even with different cell-type modifications is, at this point, up to the discretion of the user to define arrays as intended; and assuming proper use, this routine will not by itself calculate incorrectly. Extensive commenting exists within the code to assist any future user in modifying parameters. To test that the values of parameters are successfully sent to the subroutines, print statements are placed in the subroutine at the first line below the "entered(*subroutine*)" command, as follows:

```
entered("julie_biochem_repulsion");
printf("entered repulsion routine");
printf("value of k = %20.13e \n",k);
```

```
printf("value of b = %20.13e \n",b);
```

When NPHASE is run, and the biochemistry routines are implemented, these print statements will send the values of k and b that the repulsion model is receiving from the overall biochemistry routine to the output file. They can therefore be checked and verified that their values are correct.

4.2 Verifying the Adhesion Model

4.2.1 MATLAB

MATLAB codes of the adhesion calculations were built at two separate points during the development of the Adhesion Model. MATLAB was used before the initial implementation of the Adhesion Model, to simulate the intended calculations without the presence of a mesh or fluid. This was used simply to ensure the continuity of variable definitions, initializations and agreements. MATLAB was used again while NPHASE was not running to isolate the activity of the Adhesion Model and verify proper behavior. For instance, Step 3 (from section 3.3.3) originally did not have the requirement that there must be at least one face pair within the critical adhesion distance from each other in order to run, and would instead run following every timestep. This would cause NPHASE to crash immediately in all instances except those where the cells were initialized artificially within adhesion proximity, because it resulted in a dividing by zero situation. Running the routine through MATLAB quickly revealed a number of parameters to be calculated as "NaN" (meaning Not a Number) that is often a sign of dividing by zero, and made the error easy to identify and correct.

An artificial "mesh" was created in MATLAB, to test the ability of the Adhesion Model to calculate distances between faces and simulate bonds. Two arrays, one representing each cell, were created, containing eight centroid locations for parallel rectangular planar surfaces.

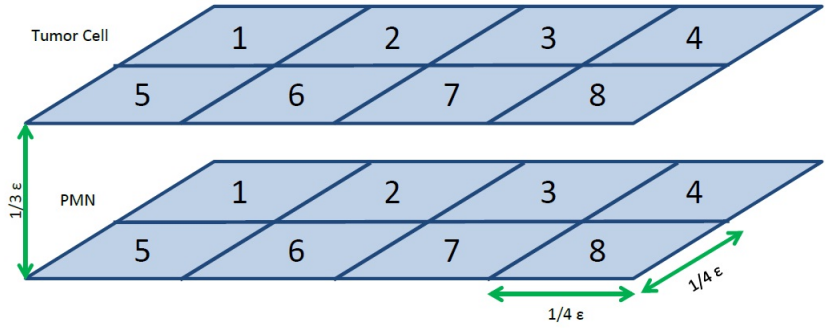


Figure 4.1: This simplified version of a mesh was used to run the Adhesion model through MATLAB without requiring the input of the detailed geometric mesh used by NPHASE.

Face Identification Number	Centroid Location: x-coordinate	y-coordinate	z-coordinate
1	X_1	Y_1	Z_1
2	X_2	Y_2	Z_2
3	X_3	Y_3	Z_3
4	X_4	Y_4	Z_4
5	X_5	Y_5	Z_5
6	X_6	Y_6	Z_6
7	X_7	Y_7	Z_7
8	X_8	Y_8	Z_8

Figure 4.2: The rectangular mesh model that is geometrically described in the previous figure was numerically defined in terms of the centroid location of each face using this structure.

Based on the molecule number density and the face area, the code calculated the number of molecules that would be on each face. It then calculated the distance between every possible pair of faces, generated an 8x8 matrix of distance values. For any distance values that were less than the critical adhesion distance, ϵ, a value of k_{on} was calculated as well. The values of k_{on} are also saved in an 8x8 matrix, and if the two faces corresponding to a position in this matrix are at a greater distance from each other than the critical adhesion distance, that k_{on} value is saved as zero. Every time a non-zero value for k_{on} is calculated, it is added to a sum of all k_{on} values so far, so that an average can be found and compared to a "global contact area wide value for k_{on}. The ratio of the average local k_{on} and the contact area global k_{on} is used to correct each local value, so that the global value becomes the average local value. The first results gathered from this MATLAB model were outputting a plot of the local k_{on} values as a function of the separation distance between the two faces. Also, the local k_{on} values were plotted three-dimensionally against the face identification numbers of both surfaces.

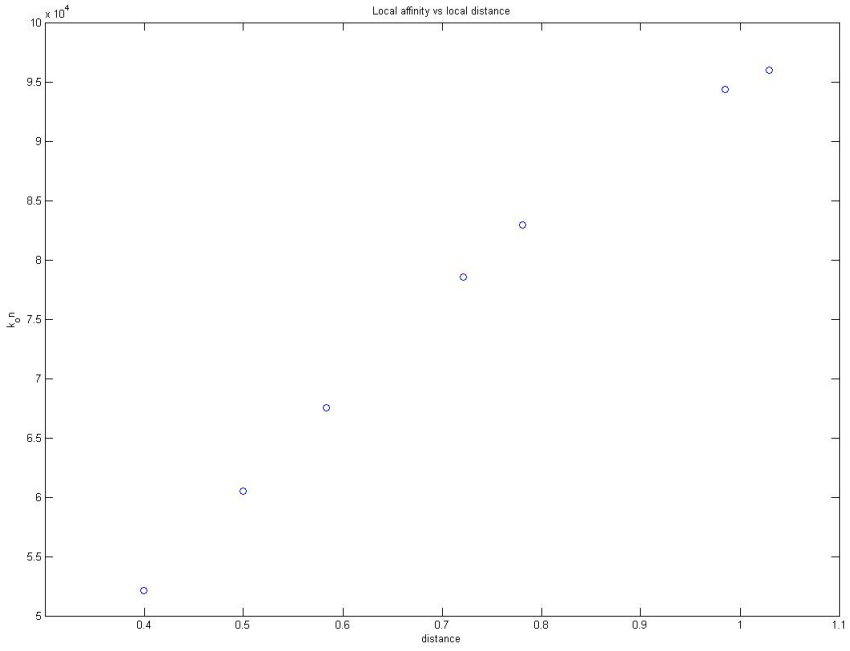


Figure 4.3: MATLAB model of Adhesion routine, affinity values plotted against separation distance between two mesh faces.

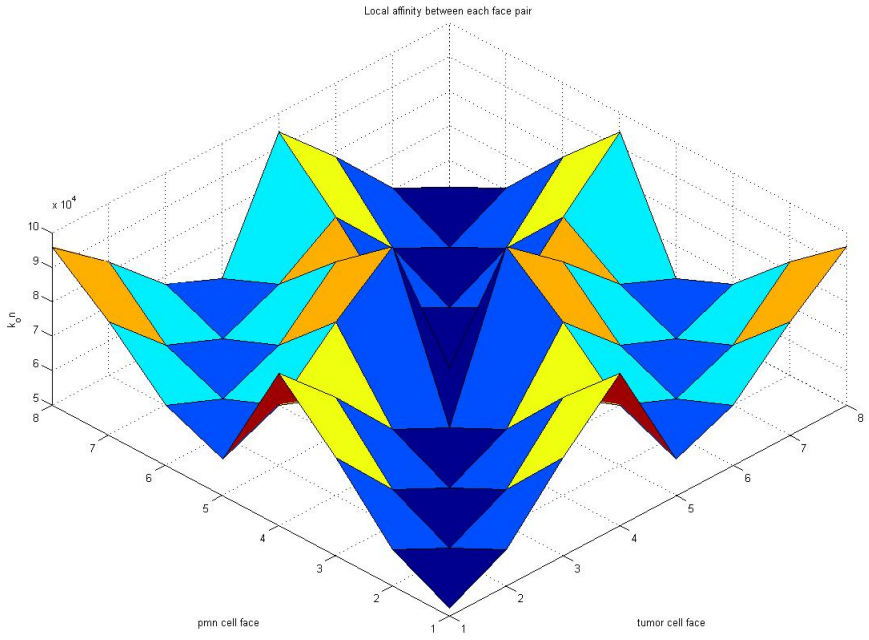


Figure 4.4: MATLAB model of Adhesion routine, affinity values plotted as a surface with respect to the face identification numbers of the faces on either surface.

4.2.2 Reusing Molecules

The next step of building the MATLAB model was to generate bonds. Rather than using the probability function with a random number generator, this phase of the model was meant to verify what molecules were being allowed to bond. Therefore, if a value of k_{on} was calculated (that is, if the two faces were within the critical adhesion distance from each other), the probability of bond formation was set as P = 1, or in other words, there is a 100% chance that a bond will form for every available molecule on that face. This was done in order to determine whether the calculated number of molecules per face was actually limiting the number of bonds that could form. The initial formulation of the adhesion model allowed for the molecules of the tumor cell, which were looped through first (one face on the tumor cell surface will be compared against all the faces on the PMN surface, before moving on to the next tumor cell face), to limit the number of bonds that could form with the first molecule type, but then be reused for the second molecule type.

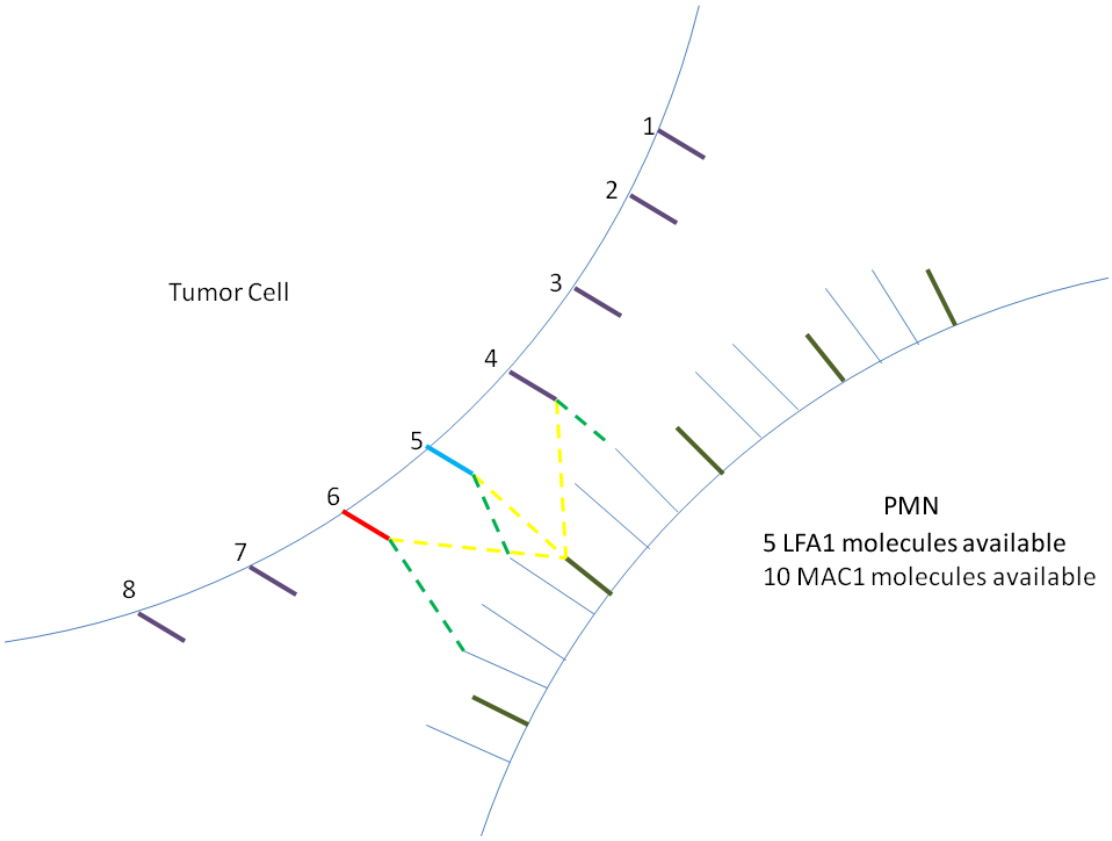


Figure 4.5: Each molecule throughout the simulation becomes available for bonding, whether or not it has previously bound, when a new pair of molecule types is being considered.

The molecules on the PMN were unlimited, and would impact the value of affinity but would not prevent more bonds forming than there were molecules available. The first modification to

be made addressed the issue of unlimited PMN molecules.

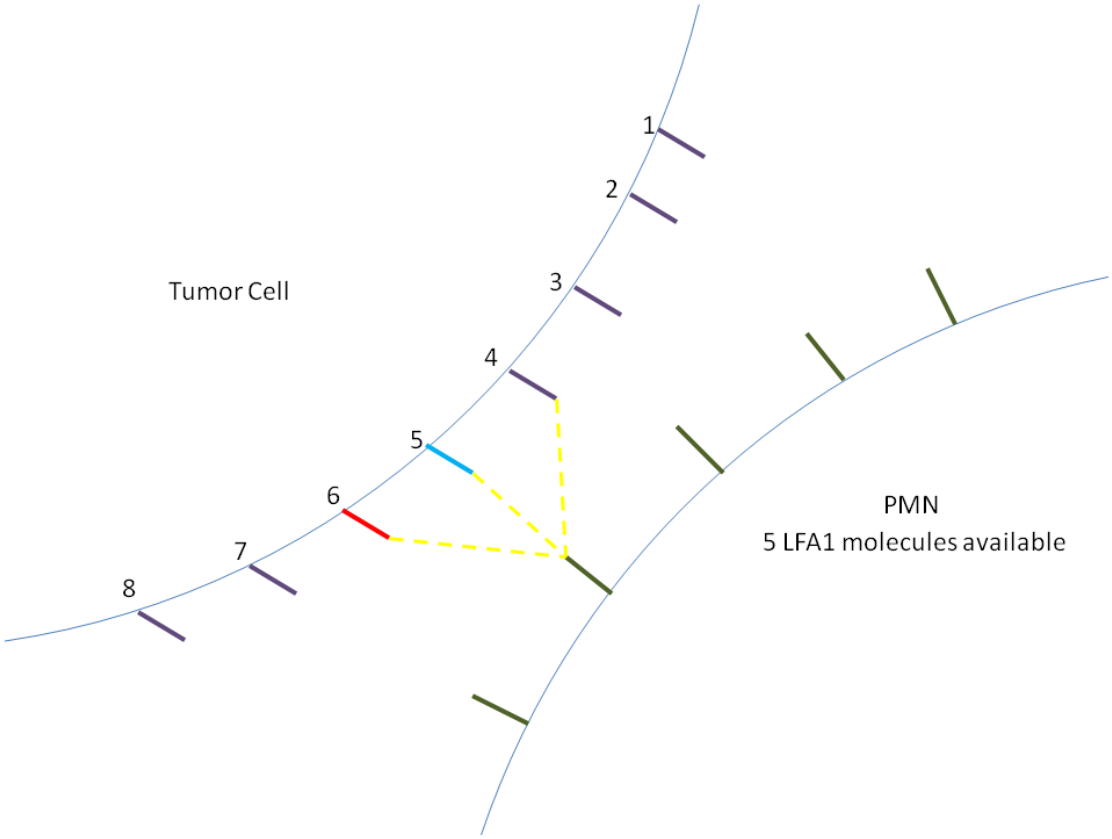


Figure 4.6: For one pair of adhesion molecule types, each tumor cell molecule will only have one opportunity to form a bond, but multiple tumor cell molecules may bind to the same PMN molecule.

Every ICAM-1 molecule on the tumor cell has an individually calculated affinity, which is based on the total number of molecules on the opposing face, per the adhesion equation. However, the opposing molecules can bind unlimited times. To correct this, a new array was created to keep track of the number of unbound molecules on the PMN faces, and this value was used to modify the affinity of ICAM-1 as more of the opposing molecules bond.

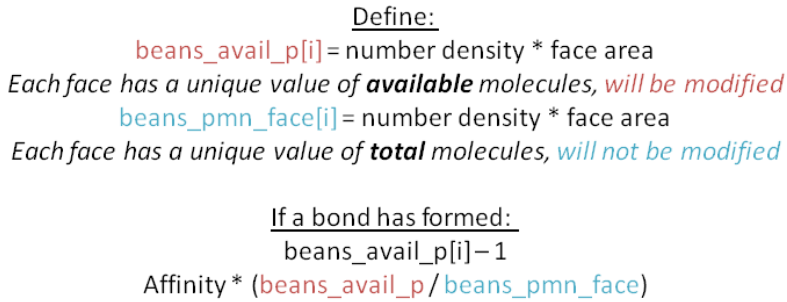


Figure 4.7: In the Adhesion routine, a new array has been built that will keep track of the total number of molecules on every face, as well as the number of available molecules. This way, an available molecule can be subtracted every time a bond forms, and this value can be saved as the routine moves through multiple tumor cell faces and compares them to the same PMN face.

The next issue to address was that the tumor cell adhesion molecules would have the same issue of being reused for bonds once a different pair of adhesion molecules (e.g. ICAM-1 to LFA-1 or ICAM-1 to Mac-1) was being considered. Based on the current formulation of the routine, including the modification to the PMN adhesion molecules made in the previous step, every molecule on either cell will be used no more than once *per molecule-type pair*, rather than no more than once total. Because ICAM-1 molecules are considered twice, and given the ability to interact with either LFA-1 or Mac-1, each one of them had the ability to form up to two bonds.

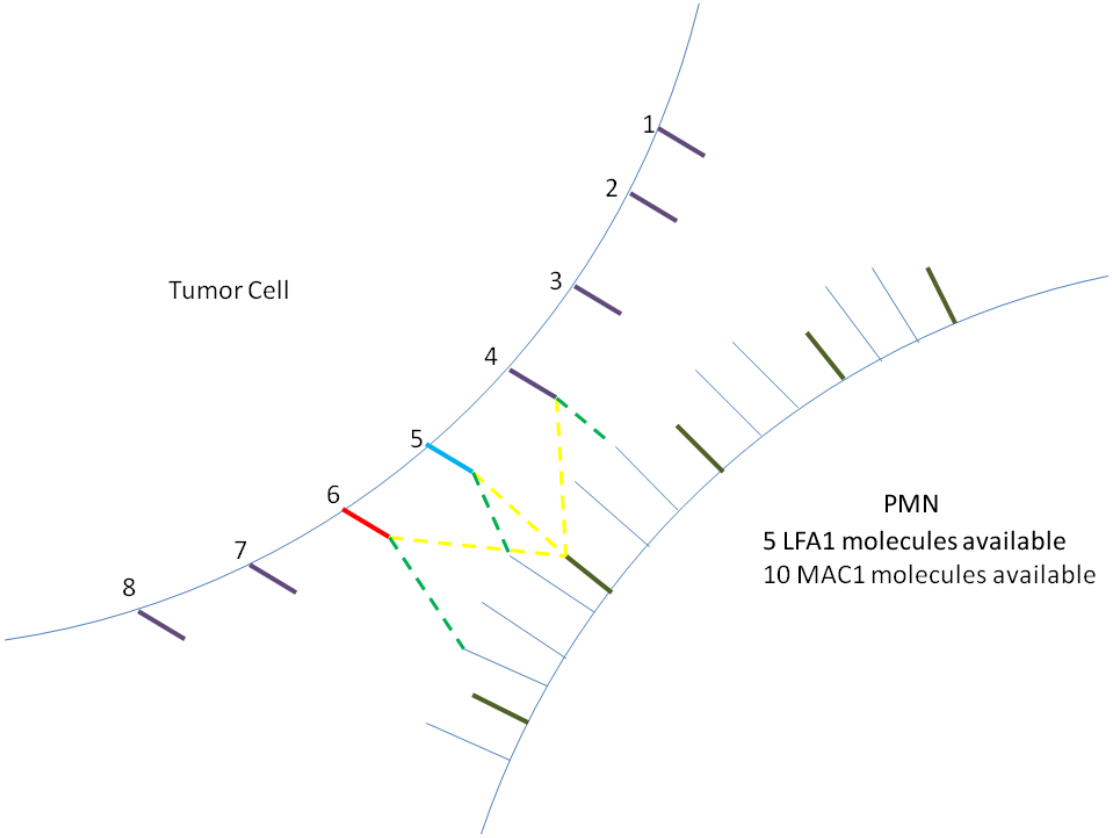


Figure 4.8: A molecule can only form one bond to any other particular molecule type, but if there are multiple types of molecules it may bind with, it may form multiple bonds.

To address this issue, the entire routine was encompassed within two overall loops, where previously there had been one. When the routine was first built, a loop with one iteration per molecule-pair-type controlled the entire adhesion model, and the calculations of affinity and bonding would be repeated for the possible molecule pairings. In this simulation, there were two iterations of the loop, one for ICAM-1 to LFA-1 bonding and one for ICAM-1 to Mac-1 bonding. There will now be two loops, one that has an iteration for every molecule on the tumor cell (in this case, there is only one, but the model needs to be robust enough to accommodate for a different cell type that is defined as having multiple relevant adhesion molecules), and one that has an iteration for every molecule on the PMN (in this case, there are two).

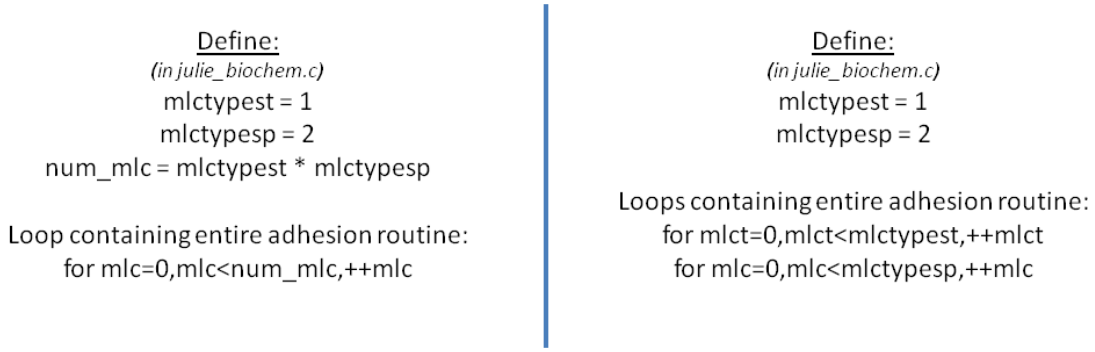


Figure 4.9: Two loops were created to contain the entire Adhesion routine, so that each molecule type, regardless of on which cell it is located, can be individually indexed and referenced throughout the routine. This allows for saving the information of a bond having already formed on a particular molecule, even if a different pair of molecules is being considered for bonding probability.

The previous step laid the groundwork for a robust enough model to accommodate for having a tumor cell with any number of molecule types on the surface, rather than the current model, which has only one. However, if a second molecule type were present on the tumor cell, it would allow for the same issue of double-counting molecules that was just corrected for the PMN adhesion molecules.

Figure 4.10: Each molecule on the tumor cell surface can bind to any molecule on the PMN surface, even if one tumor cell molecule of a different molecule type has already bound to that same PMN molecule.

To correct this problem, and complete the robustness of the routine to be able to accurately calculate the interactions between any number of molecule types, the arrays that were already defined to track how many molecules on each face are still available were modified to contain two dimensions. Two separate indexes, one representing the face on the cell surface (as before) and one representing the molecule type, control the position within the 2-dimensional array where information regarding available molecules is stored. The second index is controlled by the iteration of the overall loops that were defined in the previous step. Every time a bond forms, one available molecule is subtracted from each of the involved faces. With these 2-dimensional arrays, controlled by the overall loops of the routine, the information regarding unbound molecules will remain saved (rather than re-writing to the total number of molecules, bound or unbound, on a given face) throughout all iterations of molecule types and face references.

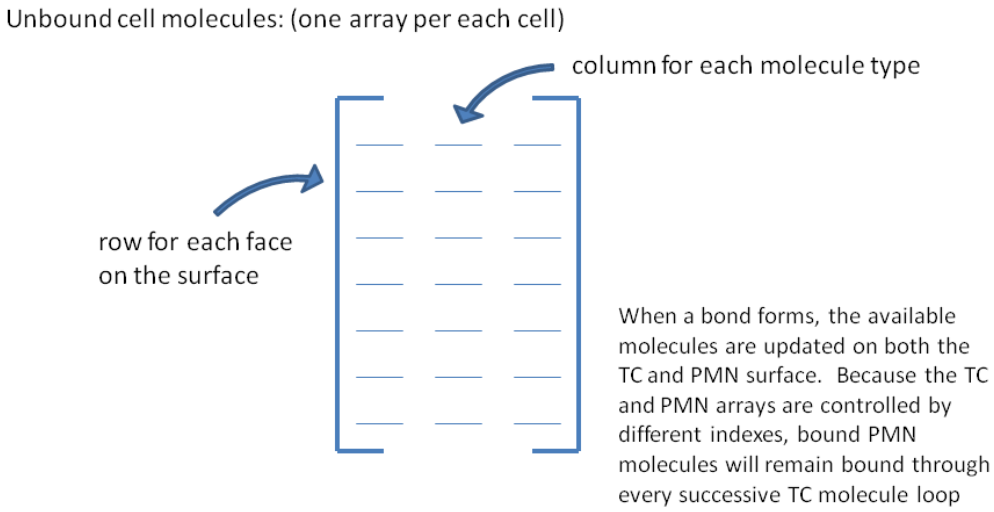


Figure 4.11: Two-dimensional arrays containing available molecule information are indexed with respect to the current iteration of the overall routine loops, and therefore once a bond is formed, those two molecules remain bound throughout the rest of the routine, regardless of which faces or molecule types are being considered.

4.2.3 Bonding Feasibility

Based on the existing values for parameters, cell surfaces will never get close enough for bonds to form. The length of microvilli, which dictates how close cell surfaces can approach each other before being inhibited by repulsion forces, is clearly orders of magnitude greater than the length of adhesion molecules, which dictates how close cell surfaces must approach each other before bonding will be considered. To correct for this influence, and allow for the propagation of bond formation between the two cells, the critical adhesion distance is modified to be the actual critical adhesion distance plus the critical repulsion distance. This formulation essentially assumes that all adhesion molecules, at all locations across the cell surface, are located at the fully-extended tips of microvilli, which is of course not accurate. Experimental verification will be required to determine whether this formulation, although obviously physically inaccurate, can still represent the actual behavior of adhesion molecules and how they approach each other, or if some other modification is required to allow for binding between the two cells to be possible.

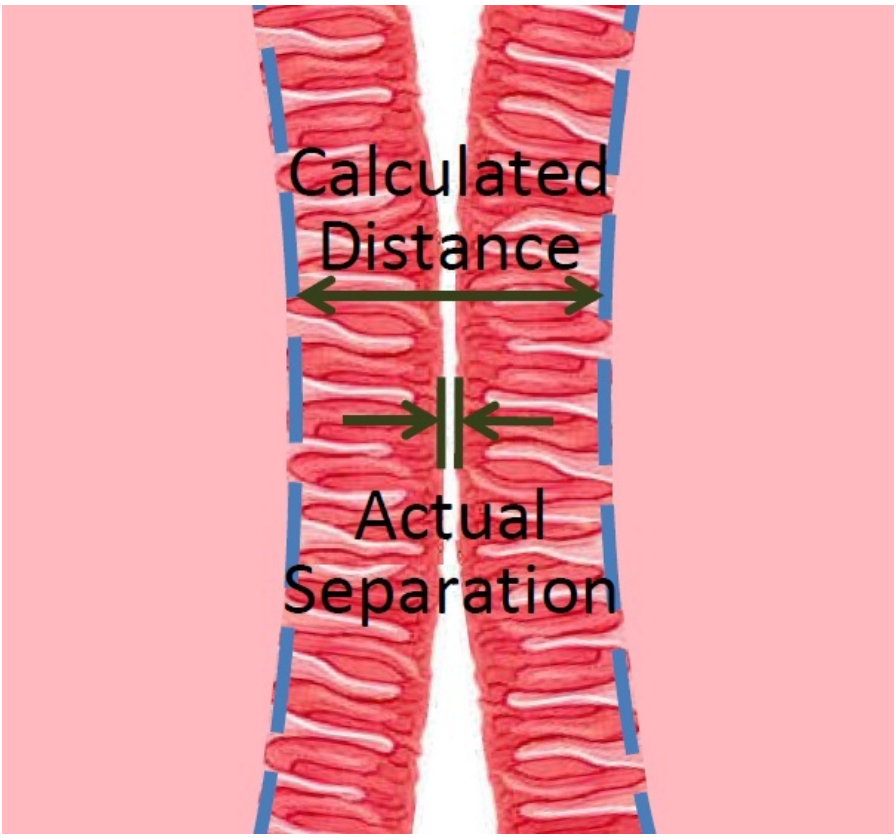


Figure 4.12: Adhesion molecules will never be able to form bonds if it is assumed that they are still separated by too great a distance when the repulsion force begins to push the surfaces even further apart. Adapted from [4]

All values for parameters within this simulation were previously defined in a past generation of the CFD code. Further literature search and experimental data is necessary to verify or modify the values, which currently provide values of k_{on} prohibitively close to zero to allow for bond formation.

4.3 Verifying the Repulsion Model

In the course of verifying the Adhesion Model, it became clear that the output of bond formation indicated that all bonds were forming between the same two cellular faces. The routine writes an output file containing, in the first line, the number of bonds that have formed, and in the following lines information related to the molecule type and face identification numbers for the faces and molecules involved.

Adding print statements into the Adhesion routine reveals that bonds are formed between many different face pairs, but each new bond overwrites all the previous bonds, such that the output file contains the information for the final bond that was formed in the routine repeated

Table 4.1: Output from the Adhesion routine clearly shows that all bonds are supposedly being generated between the same two faces

14		
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203
1	281	203

once for every bond that had previously formed. The issue was incorrectly defined 2-dimensional arrays, which had been used to store bond information, as well as to calculate distances between cell surface faces, the number of available molecules on every face (as described in the previous section), and the values of affinity for each face pair. A new method for allocating memory to 2-dimensional arrays was added into the NPHASE routines, and all 2-dimensional arrays within the Adhesion routine were allocated using this new syntax. Because the Repulsion routine also makes use of a 2-dimensional array, for storing the distances between every pair of faces, the new memory allocation method had to be used in this routine as well. Previously, it was assumed that by giving an array two input indexes, a location within that array related to the row of index 1 and the column of index 2 was being described. Because of the necessity of the Adhesion routine to reference a position within the arrays repeatedly and in any order, it was essential that they perform as intended. In the Repulsion routine, this consistency is less important. The distance will be calculated between a pair of faces, and if they are within the critical distance, forces will be output to the six-degree-of-freedom motion solver. Each pair of faces is only considered once, and it is irrelevant to the output force to know where each force was generated. So long as the distance calculation itself is accurate, it does not matter where that distance is stored within the array, or whether it is stored at all. However, for the sake of accuracy and consistency, the realization made through the Adhesion routine of how to properly build a 2-dimensional array was carried over to the Repulsion routine. All 2-dimensional arrays have now been verified as storing data as they were intended, as described in multiple figures in previous sections.

Chapter 5

Conclusions and Recommendations for Future Work

5.1 Proposed Experimental Verification

The models represented thus far within this thesis can hold no validity without experimental verification. Every parameter value, as well as the mathematical models themselves, must be consistent with data from the corresponding experiments that the simulation seeks to match. Our model must be confirmed at every scale that it seeks to represent.

5.1.1 Verification of Adhesion

At the molecular level, we must confirm the parameters for k_{on}^0 and k_{off}^0 that were found from literature. We must also examine the functions representing the patterns of distribution for k_{on} and k_{off} with various densities of molecules and proximities of surfaces, in order to confirm the mathematical modification of k_{on}^0 and k_{off}^0 based on simulation-specific conditions. The strength of the bonds that form between actual adhesion molecules should be tested, in order to verify s_{ts} and s, the spring constants used to represent bound molecules as linear springs.

It is significant to note that, at present, altering the parameters of affinity within the adhesion model such that the overall total bonding observed and modelled are equivalent would likely lead to making each simulated molecule more likely to form a bond than any real molecule. This is because the cells in the simulation are currently undeformable. We expect to see less bonding in the simulated cells than in real cells, because the contact areas will be much smaller. Calibration of this part of the model must wait until cell deformation is being calculated, and has been compared to observed cell deformation and proven representative.

Many researchers (including Zaccai, 2000; Tegenfeldt et al, 2004; and Chtcheglova, 2004) have determined methods for empirically calculating the spring constant, s, of a molecule. Bell et al

(1984) determined numerically that the spring constant of α-helical proteins must fall between 10^{-2} and 10^{3} dyn/cm, and their approximation of s = 0.1 dyn/cm has been largely accepted since. Recent researchers have used atomic force microscopy to apply a force to opposite ends of a bound molecular complex and determine the magnitude of the applied force as well as measure the change in length of the complex, in order to determine an empirical value s representing the linear spring constant.

Experimental data so far has suggested that the interactions between ICAM-1 and LFA-1 are primarily responsible for the initial tethering of a melanoma cell to a PMN (Dong et al, 2005). The interactions between ICAM-1 and Mac-1 played a greater role in stabalizing the adhesion between the two cells, even in the presence of shear flow, after the initial tethering had taken place. It is likely that this difference of roles between the two molecule types should be reflected in the simulation by different parameters describing their behaviors. Possibly the initial reliance on LFA-1 can be explained simply by its increased membrane concentration compared to Mac-1. However, the distinction of Mac-1 playing a greater role in combating shear forces possibly reflects a difference in the strength of bonds formed between ICAM-1 and Mac-1, or a difference in the typical lifespan of those bonds as reflected in k_{off}^{0}. The bond strength variation, if proven to be the case, should be reflected in a different value for the spring constants that represent molecular bond forces as the force on a spring for the ICAM-1 to Mac-1 interaction than the value assigned to the ICAM-1 to LFA-1 interaction. Currently both types of bonds are governed by the same parameters, but it is likely that experimental verification of parameter values will reveal differences between the two molecule types that can explain their differences in behavior.

5.1.2 Verification of Repulsion

At the cellular level, we must confirm the strength and extent of the repulsion model. The pattern of the non-linear spring force, as well as the critical distance at which it is activated, should be representative of the motion of an actual rough-surfaced cell. Imaging of the motion of real cells will help to determine whether our pattern of repulsion matches with the observed altered motion of the cells caused by the presence of unmodelled microvilli.

5.1.3 Verification of Overall Model

At the system level, data must be gathered representing the aggregation probability for different input populations. The model should be able to predict the likelihood of aggregation between two cells in a flow, with variable inputs of population concentrations of every defined cell type. Therefore, many experiments must be run with various ratios of cell type concentrations, to build a recreatable database of output aggregation values that the model must match.

5.2 Future Code

The next generation of the simulation to be implemented will be allowing deformability of the two cells in the model. All calculations herein defined have assumed the eventual deformation of the cells to be a necessary possibility, but at present the cells are rigid in the form of their initially generated meshes. Most experimental validation will either be irrelevant until after the implementation of cell membrane deformability, or will represent determination of parameters that must be recalibrated after cells become deformable.

Future generations of the code will be able to simulate the interactions of many different cells types, not only a melanoma cell and a PMN. Definitions of the relevant characteristics of different cell types (such as the types of adhesion molecules, the $k_o n^0$ rates of interactions between different adhesion molecule pairs, and the characteristic spring constants that represent the bond forces of bonds formed between different adhesion molecule pairs) should be included in the overall biochemistry routine. Determination must be made as to how to identify a cell type within NPHASE. Possibly this will require the creation of keywords that can be identified in the nphase.dat file (which is created within the Python script) to call on different definitions within the biochemistry routine, or some other method by which the cell surfaces within the grid can be flagged as a certain type of cell, whose parameters are internally defined. An arbitrary number of cell types should be possible within a single simulation, rather than limiting the model to handling two cell types at a time.

The routines should also be modified such that they can handle the interactions of an arbitrary number of cells, and identify the interactions between all molecules on a surface with any surrounding molecules within the adhesion proximity, regardless of how many separate cells are involved.

In addition to circulating cells, the software should eventually be able to simulate circulating molecules and proteins, which can interact with adhesion molecules on the cell surfaces as well.

To specifically address the application of this software to studying the metastasis processes of circulating melanoma cells, further detail must be given to the cells. In addition to the β-2 integrins currently defined on the PMN surface, selectins, for instance, should be modeled as well. The simulation currently assumes an adherent white blood cell, which remains rigidly stationary throughout the interactions with the melanoma cell. A later version of the simulation should allow for interaction with the endothelial wall, which is coated with various adhesion molecules similar to the circulating cells. Selectin interaction between the PMN and endothelium initiates PMN rolling along the endothelial wall, which later becomes strong attachment when the interactions between β-2 integrins and ICAM-1 of the endothelium become significant. The ability of the endothelium in the simulation to interact with the cells will also allow for the study of the eventual interactions between the melanoma cells and the endothelium, which will be mediated by the PMN and is the behavior of ultimate interest to this study.

The geometry of the mesh that is input to NPHASE, the CFD solver, is currently representative of the experimental flow chamber that will be used to confirm the results of the simulation.

When the parameters of the computational model have been modified, calibrated and confirmed, the geometry of the vessel will be redesigned to represent the geometry of in vivo blood vessels. If the model has been rigorously verified, designing the domain to represent biological flow fields rather than experimentally defined flows can be trusted to yield accurate results.

5.3 Conclusions

The major development of this work in particular was the creation of a numerical model for local adhesion affinity. The assumption upon which this model was contrived is that the existing calculation for affinity at the cellular-level is inherently physically accurate and therefore can be extrapolated to a smaller scale. The equation itself implies that bonding is actually most likely at exactly the equilibrium distance, and gets less likely if the molecules are too close to each other, but impossible if they are further away. This counterintuitive relationship between distance and affinity was clearly seen in the MATLAB simplification of the biochemistry routines.

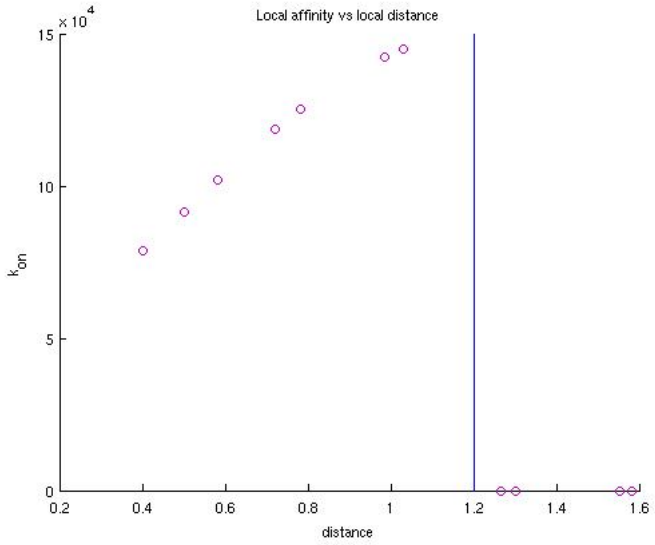


Figure 5.1: MATLAB model output of local affinity values versus the separation distance between two faces; cutoff line indicating the critical adhesion separation distance.

Affinity becomes possible only once the cell surfaces are within the critical distance; at distances greater than the critical adhesion distance, the affinity immediately becomes 0. As soon as the surfaces are in close enough proximity to generate a non-zero value of affinity, affinity is at its maximum. As surfaces move even closer to each other, and adhesion molecules may be overlapping each other but less likely for their active sites to come into contact, affinity remains non-zero but decreases. The adhesion molecules are able to reach each other, but would now have to bend in a particular way that allows their active sites to interact. It will be interesting to see experimental results that either confirm the model presented here or suggest a modification.

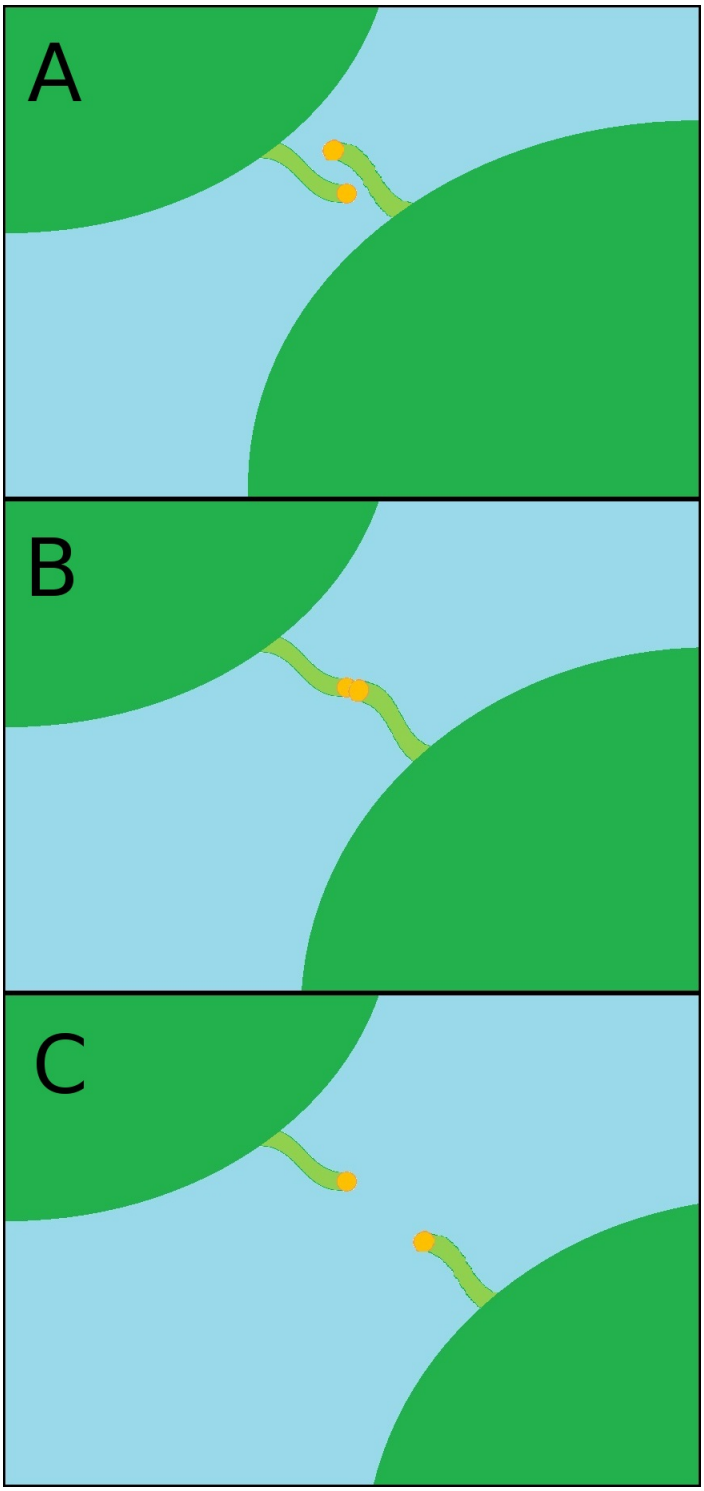


Figure 5.2: Physical representation of affinity values at various levels of cell separation. A: Molecules on surfaces that are closer together than the critical adhesion distance have the possibility of bond formation, yet must bend for bonding to be possible, and therefore the probability is non-zero but decreased. B: Molecules on surfaces at the critical separation distance are the most likely to form bonds, as their active sites are already in very close proximity. C: Molecules on surfaces that are farther apart than the critical separation distance have no opportunity to form bonds.

The work presented here has already been implemented in real-world application simulations, as demonstrated in Figure 2.9. It is ready to be applied to future generations of the overall, multiscale simulation effort, which models macroscale populations of cell types and uses the cellular-level and molecular-level models developed and described herein to predict aggregation behaviors.

Appendix A

Overall Biochemistry Routine

This file is saved in NPHASE as *julie_biochem.c*, and includes the declaration of all parameters and assignment of values to variables. When the final, fully robust version of the biochemistry routines are complete, it is intended that this is the only file that would need to be modified to add in new cell types. This file is written in C.

```
#include "nphase_struct.h"
#include "mpi.h"
#include <math.h>
#include <stdio.h>
extern struct boundary_patch wall;
extern struct boundary_patch porwall;
extern struct boundary_patch partition;
extern struct data var;
extern int myid;
extern int numprocs ;
int  julie_biochem()
{
/*-------------------------------------------------


called from:
nphase

V2.0 baseline code
initials comment
rfk
```

```
-------------------------------------------------*/

        int *ijulie_biochem=var.ijulie_biochem ;
        real k, b, epsilon;
        int *mlc_id, num_mlc;
        real *k_off_not, spring, springts, lambda, kbT, *k_on_not;
        real *pmn_dens, *tc_dens;
        int mlctypesp, mlctypest;
        int i;

        real *tempralloc(),*temprdealloc() ;
        int *itempalloc(),*tempdealloc() ;


entered("julie_biochem") ;

if(*ijulie_biochem==1){ //moving cell

        k=-110.e-6;          // value from Meghan's thesis
        b=600.e6;            // value from Meghan's thesis
        epsilon=1.2e-6;      // value from Meghan's thesis, verified by Hammer
        and Apte, units meters -- 1.2e-6 unless testing

        // For a PMN, there are 2 types of surface adhesion molecules being
        modeled
        mlctypesp = 2;  // THIS WAS CHANGED FROM 2 TO 1 FOR THE SAKE OF TESTING
        ADHESION LOOPS
        // "molecule 1" will be LFA-1
        // "molecule 2" will be MAC-1

        // For a Melanoma cell, there is 1 type of surface adhesion molecule
        being modeled
        mlctypest = 1;
        // "molecule 1" will be ICAM-1

        // total number of types of interactions is num_mlc
        num_mlc = mlctypesp*mlctypest;

        // allocate memory
        mlc_id=itempalloc(num_mlc-1);
```

```
        k_off_not=tempralloc(num_mlc-1);
        k_on_not=tempralloc(num_mlc-1);
        pmn_dens=tempralloc(num_mlc-1);
        tc_dens=tempralloc(num_mlc-1);


        // initialize arrays as zeros before specifying cell types
        for(i=0;i<num_mlc;++i){
            mlc_id[i]=0;
            k_off_not[i]=0;
            k_on_not[i]=0;
            pmn_dens[i]=0;
            tc_dens[i]=0;
        }

        // for a PMN and Melanoma cell, where there are two types of
        interactions considered

        // Meghan's thesis: Page 231 of pdf (numbered 217)

        mlc_id[0] = 1;              // calling molecule id 1 = LFA-1, molecule
        id 2 = MAC-1
        mlc_id[1] = 2;              // calling molecule id 1 = LFA-1, molecule
        id 2 = MAC-1

        k_off_not[0] = 0.3;         // value from Meghan's thesis, units 1/s
        k_off_not[1] = 0.29;        // value from Meghan's thesis, units 1/s

        spring = 2.0e-3;            // value from Hammer and Apte 1992,
        units N/m
        springts = 1.0e-3;          // value from Hammer and Apte 1992,
        units N/m
////////////////////                // VALUE EDITED TO TEST, SHOULD BE 0.05e-6
        lambda = 0.01e-6;           // value from Hammer and Apte 1992,
        units m (was 0.02e-6 in Meghan's thesis)
        lambda+= epsilon;           // critical distance for repulsion --
        should be 1.0e-6, length of 2 microvilli
        kbT = 4.28e-21;             // value from Meghan's thesis,
        meter^2 * kg / s^2, T = 310K
```

```
        k_on_not[0] = 3000;          // value from Meghan's thesis, units 1/Ms
        k_on_not[1] = 3000;          // value from Meghan's thesis, units 1/Ms

        pmn_dens[0] = 45e12;         // value from Meghan's thesis, molecules
        per meter^2
        pmn_dens[1] = 5e12;          // value from Meghan's thesis, molecules
        per meter^2

        tc_dens[0] = 13e12;          // value from Meghan's thesis, molecules
        per meter^2
        tc_dens[1] = 13e12;          // value from Meghan's thesis, molecules
        per meter^2


        julie_biochem_adhesion(mlctypest,mlctypesp,k_off_not,spring,springts,
        lambda,kbT,k_on_not,pmn_dens,tc_dens);
        julie_biochem_repulsion(k,b,epsilon);

        tempdealloc(mlc_id);
        temprdealloc(k_off_not);
        temprdealloc(k_on_not);
        temprdealloc(pmn_dens);
        temprdealloc(tc_dens);

}

exiting("julie_biochem") ;


return 0 ;
}
```

Appendix B

Repulsion Biochemistry Routine

This file is saved in NPHASE as *julie_biochem_repulsion.c*, and takes input parameters from the overall routine to calculate the simulated microvilli repulsion force. This file is written in C.

```
#include "nphase_struct.h"
#include "mpi.h"
#include <math.h>
#include <stdio.h>
extern struct boundary_patch wall;
extern struct boundary_patch porwall;
extern struct boundary_patch partition;
extern struct data var;
extern int myid;
extern int numprocs ;
int  julie_biochem_repulsion(real k, real b, real epsilon)
{
/*-------------------------------------------------


called from:
nphase

V2.0 baseline code
initials comment
rfk

-------------------------------------------------*/
```

```
int faceid, faceid2;
int *iblank=var.iblank;
int *nnode=var.nnode;
        int *iter=var.iter;
        real *tphys=var.tphys;
        real *deltatphys=var.deltatphys;
        real *transx=var.transx;
        real *transy=var.transy;
        int *ijulie_biochem=var.ijulie_biochem ;
        int *grid_motion=var.grid_motion ;
// int *lcyl=var.lcyl ;
int *num_pid=var.num_pid ;
// int *number_dynamic_groups=var.number_dynamic_groups ;
int ibf,inode,ifield,n1,n2;
int ibf2,inode2,fstride,iface;
        int *pid=var.pid ;
        int *gid=var.gid ;
        int ipid;
real cd,cdexact,red;
        real rhoref,uref,axs;
        real xtemp,veltemp,ytemp;
        real vxforce, vyforce, vzforce;
        real uf,vf,wf;
        real ycur,yprev,xprev,yvel;
       real sumforcex0;
        real sumforcey0;
        real sumforcez0;
        real sumforcevx,sumforcevx0;
        real sumforcevy,sumforcevy0;
        real sumforcevz,sumforcevz0;
    real sumforcepx,sumforcepx0;
        real sumforcepy,sumforcepy0;
        real sumforcepz,sumforcepz0;
        real torquexy,sumtorquexy0;
        real torqueyz,sumtorqueyz0;
        real torquezx,sumtorquezx0;
        real sumtorquexy;
        real sumtorqueyz;
        real sumtorquezx;
```

```
        real forceix,forceiy,forceiz;
        real sumsurfa,sumsurfa0;
    real deltay, deltax, deltaz;
        real distx,disty,distz;
        real normx,normy,normz;
        real *vism=var.vism;
        real *sp=var.sp;
        real *u=var.u;
        real *v=var.v;
        real *w=var.w;
        real *p=var.p;
        real *pi=var.pi;
        real *ucur=var.ucur;
        real *vcur=var.vcur;
        real *wcur=var.wcur;
        real *oxycur=var.oxycur;
        real *oyzcur=var.oyzcur;
        real *ozxcur=var.ozxcur;
        real *xcentcur=var.xcentcur;
        real *ycentcur=var.ycentcur;
        real *zcentcur=var.zcentcur;

real *areax=var.areax;
real *areay=var.areay;
real *areaz=var.areaz;
real *amag=var.amag;

real *face_imm_xvel=var.face_imm_xvel;
real *face_imm_yvel=var.face_imm_yvel;
real *face_imm_zvel=var.face_imm_zvel;

real *facetmlt1=var.facetmlt1;
real *facetmlt2=var.facetmlt2;

        real *xface=var.xface;
        real *yface=var.yface;
        real *zface=var.zface;

int *num_imm=var.num_imm;
int *cell_imm=var.cell_imm;
```

```
int *face_imm=var.face_imm;

int *nface0=var.nface0;

int *nface=var.nface;

        int numface_t;
        int numface_p;
        int i;
        int j;
int ic1=0;
int ic2=0;
int ic1g=0;
int ic2g=0;

  real *biochem_rep_xforce=var.biochem_rep_xforce ;
   real *biochem_rep_yforce=var.biochem_rep_yforce ;
real *biochem_rep_zforce=var.biochem_rep_zforce ;
real *biochem_rep_yztorque=var.biochem_rep_yztorque ;
real *biochem_rep_zxtorque=var.biochem_rep_zxtorque ;
real *biochem_rep_xytorque=var.biochem_rep_xytorque ;

        FILE *fi;
        int maxline=1000;
        char line[1000];
        float centroidx,centroidy,centroidz,radius;
        real centroidx8,centroidy8,centroidz8,radius8;


        real **face_sep_dist;
        real **force_rep;
        real *force_rep_x, *force_rep_y, *force_rep_z;

        real force_rep_sumx;
        real force_rep_sumy;
        real force_rep_sumz;
        real force_rep_sumx0;
        real force_rep_sumy0;
        real force_rep_sumz0;
```

```
real *globXIB,*globYIB,*globZIB;
real *globXW,*globYW,*globZW;
real *locXIB,*locYIB,*locZIB;
real *locXW,*locYW,*locZW;
int *globFlagIB,*globFlagW,*locFlagIB,*locFlagW;

        real *facecen_tx,*facecen_ty,*facecen_tz;
        real *facecen_px,*facecen_py,*facecen_pz;

int *countF,*offsetF,*countF0,*offsetF0;
int iOffset,icpf,icpf0,icpf1;
        int icpfTC, icpfTC0, icpfPMN, icpfPMN0;
int tmp=0;

int *f_n1=var.f_n1;
int *f_n2=var.f_n2;

real *tempralloc(),*temprdealloc() ;
int *itempalloc(),*tempdealloc() ;
        real **temp2dealloc(), **tempr2alloc();
FILE *fo, *fo2;

entered("julie_biochem_repulsion") ;
 \n",myid);}

if(*ijulie_biochem==1){ //moving cell


        // everything following is Byron, not Julie, until myid==0
        countF=itempalloc(2*numprocs);
        offsetF=itempalloc(2*numprocs);

        countF0=itempalloc(2*numprocs);
        offsetF0=itempalloc(2*numprocs);

for(i=0;i<=2*numprocs-1;i++){
  countF[i]=0;
  offsetF[i]=0;
  countF0[i]=0;
  offsetF0[i]=0;
```

```
}

countF[myid]=*nface;
countF[numprocs+myid]=partition.nbcface;
icpf=0;
icpf1=0;
        icpf=partition.nbcface;


        MPI_Reduce(countF,countF0,2*numprocs,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
        MPI_Reduce(&icpf,&icpf1,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
if(myid==0){

 for(i=0;i<=numprocs-1;i++){
  for(j=0;j<=i-1;j++){
    offsetF0[i]+=countF0[j];
    offsetF0[i+numprocs]+=countF0[j+numprocs];
  }
         }

}

   MPI_Bcast(offsetF0,2*numprocs,MPI_INT,0,MPI_COMM_WORLD);
   MPI_Bcast(&icpf1,1,MPI_INT,0,MPI_COMM_WORLD);

        locXIB=tempralloc(*nface0+icpf1);
        locYIB=tempralloc(*nface0+icpf1);
        locZIB=tempralloc(*nface0+icpf1);
        locFlagIB=itempalloc(*nface0+icpf1);

        locXW=tempralloc(*nface0+icpf1);
        locYW=tempralloc(*nface0+icpf1);
        locZW=tempralloc(*nface0+icpf1);
        locFlagW=itempalloc(*nface0+icpf1);

        globXIB=tempralloc(*nface0+icpf1);
        globYIB=tempralloc(*nface0+icpf1);
        globZIB=tempralloc(*nface0+icpf1);
        globFlagIB=itempalloc(*nface0+icpf1);
```

```
        globXW=tempralloc(*nface0+icpf1);
        globYW=tempralloc(*nface0+icpf1);
        globZW=tempralloc(*nface0+icpf1);
        globFlagW=itempalloc(*nface0+icpf1);

for(i=0;i<=*nface0+icpf1-1;i++){
  locXIB[i]=0;
  locYIB[i]=0;
  locZIB[i]=0;
  locFlagIB[i]=0;

  locXW[i]=0;
  locYW[i]=0;
  locZW[i]=0;
  locFlagW[i]=0;

  globXIB[i]=0;
  globYIB[i]=0;
  globZIB[i]=0;
  globFlagIB[i]=0;

  globXW[i]=0;
  globYW[i]=0;
  globZW[i]=0;
  globFlagW[i]=0;
}


      ifield = 0 ; //single phase problem
      fstride=ifield* *nface ;
      iOffset=offsetF0[myid];
      for(iface=0;iface<=*nface-1;++iface){
      n1=f_n1[iface];
      n2=f_n2[iface];
      faceid=wall.bcfaceid[iface];
      if(face_imm[iface]==1){
        // Start: Tumor Cell
        if(cell_imm[n1]==2 && cell_imm[n2]!=2){
          //The first of two face loops needed for immersed cell faces
          locXIB[iface+iOffset]=xface[iface];
```

```
          locYIB[iface+iOffset]=yface[iface];
          locZIB[iface+iOffset]=zface[iface];
          locFlagIB[iface+iOffset]=1;
  ic1+=1;
        }

        if(cell_imm[n2]==2 && cell_imm[n1]!=2){
          //The second of two face loops needed for immersed cell faces
          locXIB[iface+iOffset]=xface[iface];
          locYIB[iface+iOffset]=yface[iface];
          locZIB[iface+iOffset]=zface[iface];
          locFlagIB[iface+iOffset]=1;
  ic1+=1;
        }
        // End: Tumor Cell

        // Start: PMN
        if(cell_imm[n1]==1 && cell_imm[n2]!=1){
          //The first of two face loops needed for immersed cell faces
          locXW[iface+iOffset]=xface[iface];
          locYW[iface+iOffset]=yface[iface];
          locZW[iface+iOffset]=zface[iface];
          locFlagW[iface+iOffset]=1;
  ic2+=1;
        }

        if(cell_imm[n2]==1 && cell_imm[n1]!=1){
          //The second of two face loops needed for immersed cell faces
          locXW[iface+iOffset]=xface[iface];
          locYW[iface+iOffset]=yface[iface];
          locZW[iface+iOffset]=zface[iface];
          locFlagW[iface+iOffset]=1;
  ic2+=1;
        }
        // End: PMN
       }
      }

icpf=0;
icpf0=0;
```

```
        icpfTC=0;
        icpfTC0=0;
        icpfPMN=0;
        icpfPMN0=0;
iOffset=0;
iOffset=*nface0+offsetF0[numprocs+myid];
      for(iface=0;iface<=partition.nbcface-1;++iface){
        inode=partition.bcface_n[iface];
        if(cell_imm[inode]!=2 && partition.cell_immipp[iface]==2){
        locXIB[iface+iOffset]=partition.bxface[iface];
        locYIB[iface+iOffset]=partition.byface[iface];
        locZIB[iface+iOffset]=partition.bzface[iface];
        locFlagIB[iface+iOffset]=1;
        icpfTC+=1;
        }
        if(cell_imm[inode]!=1 && partition.cell_immipp[iface]==1){
        locXW[iface+iOffset]=partition.bxface[iface];
        locYW[iface+iOffset]=partition.byface[iface];
        locZW[iface+iOffset]=partition.bzface[iface];
        locFlagW[iface+iOffset]=1;
        icpfPMN+=1;
        }
      }
\n",myid); }
        MPI_Reduce(locXW,globXW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locYW,globYW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locZW,globZW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locFlagW,globFlagW,*nface0+icpf1,MPI_INT,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(&ic2,&ic2g,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

        MPI_Reduce(locXIB,globXIB,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locYIB,globYIB,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locZIB,globZIB,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
```

```
        MPI_Reduce(locFlagIB,globFlagIB,*nface0+icpf1,MPI_INT,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(&ic1,&ic1g,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

        MPI_Reduce(&icpfTC,&icpfTC0,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
        MPI_Reduce(&icpfPMN,&icpfPMN0,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

if(myid==0){    // on processor 0: start of actual calculations


        // allocating memory to arrays of face locations
        facecen_tx=tempralloc(ic1g+icpfTC0);
        facecen_ty=tempralloc(ic1g+icpfTC0);
        facecen_tz=tempralloc(ic1g+icpfTC0);

        facecen_px=tempralloc(ic2g+icpfPMN0);
        facecen_py=tempralloc(ic2g+icpfPMN0);
        facecen_pz=tempralloc(ic2g+icpfPMN0);

        // initializing face location arrays to 0
for(i=0;i<=ic1g+icpf0-1;i++){
          facecen_tx[i]=0;
          facecen_ty[i]=0;
          facecen_tz[i]=0;
        }

for(i=0;i<=ic2g-1;i++){
          facecen_px[i]=0;
          facecen_py[i]=0;
          facecen_pz[i]=0;
        }

////////////////////////////////

        // initializing counters of faces to 0
  numface_t=0;
numface_p=0;
        // sweeping through faces, saving locations to arrays
for(i=0;i<=*nface0+icpf1-1;i++){
  if(globFlagIB[i]==1){
```

```
    facecen_tx[numface_t]=globXIB[i];
    facecen_ty[numface_t]=globYIB[i];
    facecen_tz[numface_t]=globZIB[i];
    numface_t+=1;
          }

          if(globFlagW[i]==1){
            facecen_px[numface_p]=globXW[i];
            facecen_py[numface_p]=globYW[i];
            facecen_pz[numface_p]=globZW[i];
            numface_p+=1;
          }
        }
printf("numface_t = %d (used in a temp alloc)\n",numface_t);

        // allocating memory to repulsion arrays
        force_rep_x=tempralloc(numface_t);
        force_rep_y=tempralloc(numface_t);
        force_rep_z=tempralloc(numface_t);

        face_sep_dist=tempr2alloc(numface_t,numface_p);
        force_rep=tempr2alloc(numface_t,numface_p);
        // initializing arrays to 0
        for(i=0;i<numface_t;++i){
          force_rep_x[i]=0;
          force_rep_y[i]=0;
          force_rep_z[i]=0;
          for(j=0;j<numface_p;++j){    // for the 2D arrays
            face_sep_dist[i][j]=0;
            force_rep[i][j]=0;
          }
        }



if(myid==0){ printf("ic1g,ic2g,icpf0 %d %d %9d\n",ic1g,ic2g,icpf0); }
if(myid==0){ printf("numface_t,numface_p %d %d\n",numface_t,numface_p); }


        // read in centroid location (I have actually no idea why)
```

```
        fi = fopen("centroid.dat","r") ;
        fgets(line,maxline,fi);
        sscanf(line,"%g %g %g %g",&centroidx,&centroidy,&centroidz,&radius);
        fclose(fi);

        centroidx8=centroidx;
        centroidy8=centroidy;
        centroidz8=centroidz;
        radius8=radius;

        // sweeping through all the combinations of faces
for(i=0;i<numface_t;++i){
    for(j=0;j<numface_p;++j){
        face_sep_dist[i][j]=sqrt(pow(facecen_tx[i]
        -facecen_px[j],2)+pow(facecen_ty[i]-facecen_py[j],2)
        +pow(facecen_tz[i]-facecen_pz[j],2));
// assuming this force is solving in micro newtons -- wait, it's
definitely not
                // only calculate a repulsion force if the distance is less
                than an allowable value
        if(face_sep_dist[i][j]<epsilon){
            force_rep[i][j]=-k*(epsilon-face_sep_dist[i][j])+b*
            pow(epsilon-face_sep_dist[i][j],3);
                    // break the force into x, y and z components to apply
                    to a face
            force_rep_x[i]+=force_rep[i][j]*
            (facecen_tx[i]-facecen_px[j])/face_sep_dist[i][j];
            force_rep_y[i]+=force_rep[i][j]*
            (facecen_ty[i]-facecen_py[j])/face_sep_dist[i][j];
            force_rep_z[i]+=force_rep[i][j]*
            (facecen_tz[i]-facecen_pz[j])/face_sep_dist[i][j];
        }
    }
            // currently the forces are calculated for each individual
            face, but only utilized as a net force on TC
    *biochem_rep_xforce+=force_rep_x[i];
    *biochem_rep_yforce+=force_rep_y[i];
    *biochem_rep_zforce+=force_rep_z[i];
            *biochem_rep_xytorque+=force_rep_y[i]*(facecen_tx[i]-centroidx8)
            -force_rep_x[i]*(facecen_ty[i]-centroidy8);
```

```
            *biochem_rep_yztorque+=force_rep_z[i]*(facecen_ty[i]-centroidy8)
            -force_rep_y[i]*(facecen_tz[i]-centroidz8);
            *biochem_rep_zxtorque+=force_rep_x[i]*(facecen_tz[i]-centroidz8)
            -force_rep_z[i]*(facecen_tx[i]-centroidx8);
}


////////////////////////////////

        // deallocate memory to all arrays previously allocated


        // 2D arrays require input of their first dimension
        temp2dealloc(face_sep_dist,numface_t);
        temp2dealloc(force_rep,numface_t);


        temprdealloc(force_rep_x);
        temprdealloc(force_rep_y);
        temprdealloc(force_rep_z);

temprdealloc(facecen_tx);
temprdealloc(facecen_ty);
temprdealloc(facecen_tz);

temprdealloc(facecen_px);
temprdealloc(facecen_py);
temprdealloc(facecen_pz);
}

        temprdealloc(locXIB);
        temprdealloc(locYIB);
        temprdealloc(locZIB);
        tempdealloc(locFlagIB);

        temprdealloc(globXIB);
        temprdealloc(globYIB);
        temprdealloc(globZIB);
        tempdealloc(globFlagIB);
```

```
        temprdealloc(locXW);
        temprdealloc(locYW);
        temprdealloc(locZW);
        tempdealloc(locFlagW);

        temprdealloc(globXW);
        temprdealloc(globYW);
        temprdealloc(globZW);
        tempdealloc(globFlagW);

        tempdealloc(countF);
        tempdealloc(offsetF);
        tempdealloc(countF0);
        tempdealloc(offsetF0);

}

exiting("julie_biochem_repulsion") ;


return 0 ;
}
```

Appendix C

Adhesion Biochemistry Routine

This file is saved in NPHASE as *julie_biochem_adhesion.c*, and takes input parameters from the overall routine to calculate bond formation and the resultant attractive forces. This file is written in C.

```
#include "nphase_struct.h"
#include "mpi.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
extern struct boundary_patch wall;
extern struct boundary_patch porwall;
extern struct boundary_patch partition;
extern struct data var;
extern int myid;
extern int numprocs ;
int  julie_biochem_adhesion(int mlctypest, int mlctypesp, real
*k_off_not, real spring,real springts,real lambda,real kbT,real
*k_on_not,real *pmn_dens,real *tc_dens)
{
/*-----------------------------------------------------


called from:
nphase

V2.0 baseline code
```

```
initials comment
rfk

-------------------------------------------------*/

int faceid, faceid2;
int *iblank=var.iblank;
int *nnode=var.nnode;
        int *iter=var.iter;
        real *tphys=var.tphys;
        real *deltatphys=var.deltatphys;
        real *transx=var.transx;
        real *transy=var.transy;
        real *deltatqs=var.deltatqs;
        int *iqstime=var.iqstime;
        int *nqsOffset=var.nqsOffset;

        int *ijulie_biochem=var.ijulie_biochem ;
        int *grid_motion=var.grid_motion ;
// int *lcyl=var.lcyl ;
int *num_pid=var.num_pid ;
// int *number_dynamic_groups=var.number_dynamic_groups ;
int ibf,inode,ifield,n1,n2;
int ibf2,inode2,fstride,iface;
        int *pid=var.pid ;
        int *gid=var.gid ;
        int ipid;
        real k,e1;
real cd,cdexact,red;
        real rhoref,uref,axs;
        real xtemp,veltemp,ytemp;
        real vxforce, vyforce, vzforce;
        real uf,vf,wf;
        real ycur,yprev,xprev,yvel;
       real sumforcex0;
        real sumforcey0;
        real sumforcez0;
        real sumforcevx,sumforcevx0;
        real sumforcevy,sumforcevy0;
        real sumforcevz,sumforcevz0;
```

```
    real sumforcepx,sumforcepx0;
        real sumforcepy,sumforcepy0;
        real sumforcepz,sumforcepz0;
        real torquexy,sumtorquexy0;
        real torqueyz,sumtorqueyz0;
        real torquezx,sumtorquezx0;
        real sumtorquexy;
        real sumtorqueyz;
        real sumtorquezx;
        real forceix,forceiy,forceiz;
        real sumsurfa,sumsurfa0;
    real deltay, deltax, deltaz;
        real distx,disty,distz;
        real normx,normy,normz;
        real *vism=var.vism;
        real *sp=var.sp;
        real *u=var.u;
        real *v=var.v;
        real *w=var.w;
        real *p=var.p;
        real *pi=var.pi;
        real *ucur=var.ucur;
        real *vcur=var.vcur;
        real *wcur=var.wcur;
        real *oxycur=var.oxycur;
        real *oyzcur=var.oyzcur;
        real *ozxcur=var.ozxcur;
        real *xcentcur=var.xcentcur;
        real *ycentcur=var.ycentcur;
        real *zcentcur=var.zcentcur;

real *areax=var.areax;
real *areay=var.areay;
real *areaz=var.areaz;
real *amag=var.amag;

real *face_imm_xvel=var.face_imm_xvel;
real *face_imm_yvel=var.face_imm_yvel;
real *face_imm_zvel=var.face_imm_zvel;
```

```
real *facetmlt1=var.facetmlt1;
real *facetmlt2=var.facetmlt2;

        real *xface=var.xface;
        real *yface=var.yface;
        real *zface=var.zface;

int *num_imm=var.num_imm;
int *cell_imm=var.cell_imm;
int *face_imm=var.face_imm;

int *nface0=var.nface0;

int *nface=var.nface;

        int numface_t;
        int numface_p;
        int i;
        int j;
int ic1=0;
int ic2=0;
int ic1g=0;
int ic2g=0;

real *biochem_adh_xforce=var.biochem_adh_xforce ;
real *biochem_adh_yforce=var.biochem_adh_yforce ;
real *biochem_adh_zforce=var.biochem_adh_zforce ;
real *biochem_adh_yztorque=var.biochem_adh_yztorque ;
real *biochem_adh_zxtorque=var.biochem_adh_zxtorque ;
        real *biochem_adh_xytorque=var.biochem_adh_xytorque ;

        FILE *fi, *fi2;
        int maxline=1000;
        char line[1000];
        float centroidx,centroidy,centroidz,radius;
        real centroidx8,centroidy8,centroidz8,radius8;


real *globXIB,*globYIB,*globZIB,*globAIB;
real *globXW,*globYW,*globZW,*globAW;
```

```
real *locXIB,*locYIB,*locZIB,*locAIB;
real *locXW,*locYW,*locZW,*locAW;


//      variables I just added, in case they don't work

        int num_bonds, new_bonds, kons, b, tb, pb, mlc, mlcb, mlctb, mlct,
        nb, exb_size;
        real exbt, exbp, exb, tcsurfa, pmnsurfa;

        int **existing_bonds, **existing_bonds_new;
        real **existing_bonds_df;
        real **kon_local_old, **kon_local;
        real **beans_avail_t, **beans_avail_p;
        real **face_sep_dist;
        real **beans_pmn_face, **beans_tc_face;


        real k_off, Prob, rand_num;
        real k_on_ave, contact_area, ave_dist, delta_contact_area;
        real *facearea_p, *facearea_t, *bond_dist;
        real kon_global, correction;
        real facebeans;
        real *force_ad_x, *force_ad_y, *force_ad_z;

        char file_name_faces_tc_tran[17];

////////////////////////////////////////////////////////////////


int *globFlagIB,*globFlagW,*locFlagIB,*locFlagW;

        real *facecen_tx,*facecen_ty,*facecen_tz;
        real *facecen_px,*facecen_py,*facecen_pz;

int *countF,*offsetF,*countF0,*offsetF0;
int iOffset,icpf,icpf0,icpf1;
int icpfTC,icpfPMN,icpfTC0,icpfPMN0;
int tmp=0;
```

```
int *f_n1=var.f_n1;
int *f_n2=var.f_n2;

//        real *pid_cgx=var.pid_cgx;
//        real *pid_cgy=var.pid_cgy;
real *tempralloc(),*temprdealloc() ;
int *itempalloc(),*tempdealloc() ;
        real **temp2dealloc(), **tempr2alloc();
        int **i2tempalloc();
FILE *fo, *fo2, *fo3, *fo4, *fo5;

entered("julie_biochem_adhesion") ;


if(*ijulie_biochem==1){ //moving cell

        countF=itempalloc(2*numprocs);
        offsetF=itempalloc(2*numprocs);

        countF0=itempalloc(2*numprocs);
        offsetF0=itempalloc(2*numprocs);


for(i=0;i<=2*numprocs-1;i++){
  countF[i]=0;
  offsetF[i]=0;
  countF0[i]=0;
  offsetF0[i]=0;
}


countF[myid]=*nface;
countF[numprocs+myid]=partition.nbcface;
icpf=0;
icpf1=0;
        icpf=partition.nbcface;


        MPI_Reduce(countF,countF0,2*numprocs,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
        MPI_Reduce(&icpf,&icpf1,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
```

```
if(myid==0){


 for(i=0;i<=numprocs-1;i++){
  for(j=0;j<=i-1;j++){
    offsetF0[i]+=countF0[j];
    offsetF0[i+numprocs]+=countF0[j+numprocs];
  }
        }
}

   MPI_Bcast(offsetF0,2*numprocs,MPI_INT,0,MPI_COMM_WORLD);
   MPI_Bcast(&icpf1,1,MPI_INT,0,MPI_COMM_WORLD);


        locXIB=tempralloc(*nface0+icpf1);
        locYIB=tempralloc(*nface0+icpf1);
        locZIB=tempralloc(*nface0+icpf1);
        locAIB=tempralloc(*nface0+icpf1);
        locFlagIB=itempalloc(*nface0+icpf1);


        locXW=tempralloc(*nface0+icpf1);
        locYW=tempralloc(*nface0+icpf1);
        locZW=tempralloc(*nface0+icpf1);
        locAW=tempralloc(*nface0+icpf1);
        locFlagW=itempalloc(*nface0+icpf1);


        globXIB=tempralloc(*nface0+icpf1);
        globYIB=tempralloc(*nface0+icpf1);
        globZIB=tempralloc(*nface0+icpf1);
        globAIB=tempralloc(*nface0+icpf1);
        globFlagIB=itempalloc(*nface0+icpf1);


        globXW=tempralloc(*nface0+icpf1);
        globYW=tempralloc(*nface0+icpf1);
        globZW=tempralloc(*nface0+icpf1);
        globAW=tempralloc(*nface0+icpf1);
```

```
        globFlagW=itempalloc(*nface0+icpf1);

////////////////////////////////////////////////

        force_ad_x=tempralloc(*nface0+icpf1);
        force_ad_y=tempralloc(*nface0+icpf1);
        force_ad_z=tempralloc(*nface0+icpf1);

////////////////////////////////////////////////


for(i=0;i<=*nface0+icpf1-1;i++){
  locXIB[i]=0;
  locYIB[i]=0;
  locZIB[i]=0;
          locAIB[i]=0;
  locFlagIB[i]=0;

  locXW[i]=0;
  locYW[i]=0;
  locZW[i]=0;
          locAW[i]=0;
  locFlagW[i]=0;

  globXIB[i]=0;
  globYIB[i]=0;
  globZIB[i]=0;
          globAIB[i]=0;
  globFlagIB[i]=0;

  globXW[i]=0;
  globYW[i]=0;
  globZW[i]=0;
          globAW[i]=0;
  globFlagW[i]=0;

          force_ad_x[i]=0;
          force_ad_y[i]=0;
          force_ad_z[i]=0;
```

```
}


      ifield = 0 ; //single phase problem
      fstride=ifield* *nface ;
      iOffset=offsetF0[myid];
      for(iface=0;iface<=*nface-1;++iface){
      n1=f_n1[iface];
      n2=f_n2[iface];
      faceid=wall.bcfaceid[iface];
      if(face_imm[iface]==1){
        // Start: Tumor Cell
        if(cell_imm[n1]==2 && cell_imm[n2]!=2){
          //The first of two face loops needed for immersed cell faces--TC
          locXIB[iface+iOffset]=xface[iface];
          locYIB[iface+iOffset]=yface[iface];
          locZIB[iface+iOffset]=zface[iface];
          locAIB[iface+iOffset]=amag[iface];
          locFlagIB[iface+iOffset]=1;
  ic1+=1;
        }

        if(cell_imm[n2]==2 && cell_imm[n1]!=2){
          //The second of two face loops needed for immersed cell faces--TC
          locXIB[iface+iOffset]=xface[iface];
          locYIB[iface+iOffset]=yface[iface];
          locZIB[iface+iOffset]=zface[iface];
          locAIB[iface+iOffset]=amag[iface];
          locFlagIB[iface+iOffset]=1;
  ic1+=1;
        }
        // End: Tumor Cell

        // Start: PMN
        if(cell_imm[n1]==1 && cell_imm[n2]!=1){
          //The first of two face loops needed for immersed cell faces--PMN
          locXW[iface+iOffset]=xface[iface];
          locYW[iface+iOffset]=yface[iface];
          locZW[iface+iOffset]=zface[iface];
          locAW[iface+iOffset]=amag[iface];
```

```
          locFlagW[iface+iOffset]=1;
          ic2+=1;
        }

        if(cell_imm[n2]==1 && cell_imm[n1]!=1){
          //The second of two face loops needed for immersed cell faces--PMN
          locXW[iface+iOffset]=xface[iface];
          locYW[iface+iOffset]=yface[iface];
          locZW[iface+iOffset]=zface[iface];
          locAW[iface+iOffset]=amag[iface];
          locFlagW[iface+iOffset]=1;
          ic2+=1;
        }
        // End: PMN

       }

      }


/*
      // Use when PMN is defined as a wall, not immersed boundary
      for(iface=0;iface<=wall.nbcface-1;++iface){
      faceid=wall.bcfaceid[iface];
        if(faceid==1){
          //Face sweep for rigid boundary cells
          locXW[iface+iOffset]=wall.bxface[iface];
          locYW[iface+iOffset]=wall.byface[iface];
          locZW[iface+iOffset]=wall.bzface[iface];
          locAW[iface+iOffset]=wall.bamag[iface];
          locFlagW[iface+iOffset]=1;
          ic2+=1;
       }
      }
*/

icpf=0;
icpf0=0;
        icpfTC=0;
        icpfTC0=0;
```

```
        icpfPMN=0;
        icpfPMN0=0;
iOffset=0;
iOffset=*nface0+offsetF0[numprocs+myid];
      for(iface=0;iface<=partition.nbcface-1;++iface){
        inode=partition.bcface_n[iface];
        if(cell_imm[inode]!=2 && partition.cell_immipp[iface]==2){
        locXIB[iface+iOffset]=partition.bxface[iface];
        locYIB[iface+iOffset]=partition.byface[iface];
        locZIB[iface+iOffset]=partition.bzface[iface];
        locAIB[iface+iOffset]=partition.bamag[iface];
        locFlagIB[iface+iOffset]=1;
        icpfTC+=1;
        }

        if(cell_imm[inode]!=1 && partition.cell_immipp[iface]==1){
        locXW[iface+iOffset]=partition.bxface[iface];
        locYW[iface+iOffset]=partition.byface[iface];
        locZW[iface+iOffset]=partition.bzface[iface];
        locAW[iface+iOffset]=partition.bamag[iface];
        locFlagW[iface+iOffset]=1;
        icpfPMN+=1;
        }
      }

        MPI_Reduce(locXW,globXW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locYW,globYW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locZW,globZW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locAW,globAW,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locFlagW,globFlagW,*nface0+icpf1,MPI_INT,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(&ic2,&ic2g,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

        MPI_Reduce(locXIB,globXIB,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locYIB,globYIB,*nface0+icpf1,mpireal,MPI_SUM,0,
```

```
        MPI_COMM_WORLD);
        MPI_Reduce(locZIB,globZIB,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locAIB,globAIB,*nface0+icpf1,mpireal,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(locFlagIB,globFlagIB,*nface0+icpf1,MPI_INT,MPI_SUM,0,
        MPI_COMM_WORLD);
        MPI_Reduce(&ic1,&ic1g,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

        MPI_Reduce(&icpfTC,&icpfTC0,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
        MPI_Reduce(&icpfPMN,&icpfPMN0,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);

if(myid==0){   // start of calculations

        facecen_tx=tempralloc(ic1g+icpfTC0);
        facecen_ty=tempralloc(ic1g+icpfTC0);
        facecen_tz=tempralloc(ic1g+icpfTC0);
        facearea_t=tempralloc(ic1g+icpfTC0);
        bond_dist=tempralloc(ic1g+icpfTC0);

        facecen_px=tempralloc(ic2g+icpfPMN0);
        facecen_py=tempralloc(ic2g+icpfPMN0);
        facecen_pz=tempralloc(ic2g+icpfPMN0);
        facearea_p=tempralloc(ic2g+icpfPMN0);

        new_bonds=0;

for(i=0;i<=ic1g+icpfTC0-1;i++){
          facecen_tx[i]=0;
          facecen_ty[i]=0;
          facecen_tz[i]=0;
          facearea_t[i]=0;

          bond_dist[i]=0;
        }

for(i=0;i<=ic2g+icpfPMN0-1;i++){
          facecen_px[i]=0;
          facecen_py[i]=0;
          facecen_pz[i]=0;
```

```
          facearea_p[i]=0;

        }

////////////////////////////////

        numface_t=0;
        tcsurfa=0;
        for(i=0;i<=*nface0+icpf1-1;i++){
          if(globFlagIB[i]==1){
            facecen_tx[numface_t]=globXIB[i];
            facecen_ty[numface_t]=globYIB[i];
            facecen_tz[numface_t]=globZIB[i];
            facearea_t[numface_t]=globAIB[i];
            tcsurfa+=globAIB[i];

            numface_t+=1;
          }
        }

/*
        if( *iqstime <= 9)sprintf(file_name_faces_tc_tran,
        "faces_tc_00%d.txt",*iqstime);
        if( *iqstime >= 10 && *iqstime <= 99)sprintf(file_name_faces_tc_tran,
        "faces_tc_0%2d.txt",*iqstime);
        if( *iqstime >= 100 && *iqstime <= 999)sprintf(file_name_faces_tc_
        tran,"faces_tc_%3d.txt",*iqstime);
        fo3=fopen("file_name_faces_tc_tran","w");
        for(i=0;i<numface_t;++i){
            fprintf(fo3,"%20.13e %20.13e %20.13e \n",
            facecen_tx[i],facecen_ty[i],facecen_tz[i]);
        }
        fclose(fo3);
*/

numface_p=0;
        pmnsurfa=0;
        for(i=0;i<=*nface0+icpf1-1;i++){
          if(globFlagW[i]==1){
            facecen_px[numface_p]=globXW[i];
```

```
            facecen_py[numface_p]=globYW[i];
            facecen_pz[numface_p]=globZW[i];
            facearea_p[numface_p]=globAW[i];
            pmnsurfa+=globAW[i];
            numface_p+=1;
          }
        }

/*
        if( (*iqstime-*nqsOffset) == 0){
            fo4=fopen("faces_pmn.txt","w");
            for(i=0;i<numface_p;++i){
                fprintf(fo4,"%20.13e %20.13e %20.13e \n",
                facecen_px[i],facecen_py[i],facecen_pz[i]);
            }
            fclose(fo4);
        }
*/

////////////////////////////////////////////////

////////////////////////////////////////////////

        beans_pmn_face=tempr2alloc(ic2g+icpfPMN0,mlctypesp);
        beans_avail_p=tempr2alloc(ic2g+icpfPMN0,mlctypesp);

        beans_tc_face=tempr2alloc(ic1g+icpfTC0,mlctypest);
        beans_avail_t=tempr2alloc(ic1g+icpfTC0,mlctypest);

////////////////////////////////////////////////

        exbt=0;
        for(i=0;i<mlctypest;++i){
            exbt+=tc_dens[i]*tcsurfa;
        }
        exbp=0;
        for(i=0;i<mlctypesp;++i){
            exbp+=pmn_dens[i]*pmnsurfa;
        }
        if(exbt < exbp){
```

```
            exb = exbt;
        }
        else{
            exb = exbp;
        }
        exb_size = numface_t * (int)(exb);

        kon_local_old = tempr2alloc(numface_t,numface_p);
        kon_local = tempr2alloc(numface_t,numface_p);
        face_sep_dist=tempr2alloc(numface_t,numface_p);

        existing_bonds_df = tempr2alloc(exb_size, 2);
        existing_bonds = i2tempalloc(exb_size, 4);
        existing_bonds_new = i2tempalloc(exb_size, 4);


////////////////////////////////////////////////

for(i=0;i<=ic1g+icpfTC0-1;i++){
          for(j=0;j<mlctypest;++j){
            beans_tc_face[i][j]=0;
            beans_avail_t[i][j]=0;
          }
        }

for(i=0;i<=ic2g+icpfPMN0-1;i++){
          for(j=0;j<mlctypesp;++j){
            beans_pmn_face[i][j]=0;
            beans_avail_p[i][j]=0;
          }
        }

        for(i=0;i<numface_t;++i){
            for(j=0;j<mlctypest;++j){
                beans_tc_face[i][j]=facearea_t[i]*tc_dens[j];
                beans_avail_t[i][j]=beans_tc_face[i][j];
            }
        }

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

```
        for(j=0;j<mlctypesp;++j){
            beans_pmn_face[i][j]=facearea_p[i]*pmn_dens[j];
            beans_avail_p[i][j]=beans_pmn_face[i][j];
        }
    }


    for(i=0;i<exb_size;++i){
        existing_bonds_df[i][0]=0;
        existing_bonds_df[i][1]=0;
        for(j=0;j<4;++j){
          existing_bonds[i][j]=0;
          existing_bonds_new[i][j]=0;
        }
    }

    for(i=0;i<numface_t;++i){
        for(j=0;j<numface_p;++j){
          kon_local_old[i][j]=0;
          kon_local[i][j]=0;
          face_sep_dist[i][j]=0;
        }
    }

    fi = fopen("centroid.dat","r") ;
    fgets(line,maxline,fi);
    sscanf(line,"%g %g %g %g",&centroidx,&centroidy,&centroidz,&radius);
    fclose(fi);

    centroidx8=centroidx;
    centroidy8=centroidy;
    centroidz8=centroidz;
    radius8=radius;


////////////////////////////////////////////////////////

    num_bonds = 0;
    tb = 0;
    pb = 0;
```

```
        k_off = 0;

//      Step 1: test existing bonds to see if they break, resave ones
that don't

        if( (*iqstime-*nqsOffset) != 0){

            // read in num_bonds, existing_bonds from file

            fi2 = fopen("existing_bonds.dat","r");
            fgets(line,maxline,fi2);
            sscanf(line, "%d", &num_bonds);

            for(i=0;i<num_bonds;++i){
                sscanf(line,"%d %d %d %d",&existing_bonds[i][0],
                &existing_bonds[i][1],&existing_bonds[i][2],&existing_bonds
                [i][3]);
                mlctb = existing_bonds[i][0];
                mlcb = existing_bonds[i][1];
                tb = existing_bonds[i][2];
                pb = existing_bonds[i][3];
                bond_dist[i]=sqrt(pow(facecen_tx[tb]-facecen_px[pb],2)
                +pow(facecen_ty[tb]-facecen_py[pb],2)
                +pow(facecen_tz[tb]-facecen_pz[pb],2));
                k_off=k_off_not[mlcb]*exp(((spring-springts)
                *pow(bond_dist[i]-lambda,2))/(2*kbT));
                Prob=1-exp(-1*k_off* *deltatqs);
                rand_num = (double)rand()/(double)RAND_MAX;
                //  need to find a way to eliminate one available
                molecule from each face
                if(Prob < rand_num){        // bond not broken
                    existing_bonds_new[new_bonds][0]=existing_bonds[i][0];
                    existing_bonds_new[new_bonds][1]=existing_bonds[i][1];
                    existing_bonds_new[new_bonds][2]=existing_bonds[i][2];
                    existing_bonds_new[new_bonds][3]=existing_bonds[i][3];
                    existing_bonds_df[new_bonds][0]=bond_dist[i];
                    beans_avail_p[pb][mlcb]-=1;
                    beans_avail_t[tb][mlctb]-=1;
                    new_bonds+=1;
                }
```

```
            }
            fclose(fi2);
        }


        for(mlct=0;mlct<mlctypest;++mlct){
        for(mlc=0;mlc<mlctypesp;++mlc){

        Prob = 0;
        rand_num = 0;
        k_on_ave = 0;
        kons = 0;
        contact_area = 0;
        ave_dist = 0;
        delta_contact_area = 0;
        kon_global = 0;
        correction = 0;
        facebeans = 0;


//      Step 2: face sweep, calculate preliminary kon

        k_on_ave = 0;
        kons = 0;
        contact_area = 0;
        ave_dist = 0;
        kon_global = 0;

for(i=0;i<numface_t;++i){         // each face on the tc
            delta_contact_area = 0;
    for(j=0;j<numface_p;++j){     // each face on the pmn
        face_sep_dist[i][j]=sqrt(pow(facecen_tx[i]
        -facecen_px[j],2)+pow(facecen_ty[i]
        -facecen_py[j],2)+pow(facecen_tz[i]-facecen_pz[j],2));
// assuming this force is solving in micro newtons
        if(face_sep_dist[i][j]<=lambda){
                    kon_local_old[i][j]=beans_pmn_face[j][mlc]
                    *k_on_not[mlc]*exp((-1*springts*
                    pow(face_sep_dist[i][j]-lambda,2))/(2*kbT));
                    kons+=1;
```

```
                k_on_ave+=kon_local_old[i][j];
                delta_contact_area = facearea_t[i];
                ave_dist+=face_sep_dist[i][j];
            }
        }
        contact_area+=delta_contact_area;
    }

//      Step 3: adjust kon
    if(kons>=1){
        ave_dist = ave_dist / kons;
        k_on_ave = k_on_ave / kons;
        kon_global = contact_area*pmn_dens[mlc]*k_on_not[mlc]
        *exp((-1*springts*pow(ave_dist-lambda,2))/(2*kbT));
        // TESTING: overwrite actual kon calculation excluding
        the exponent, to get a non-zero value until those numbers
        are verified
//        kon_global = contact_area*pmn_dens[mlc]*k_on_not[mlc];
        correction = kon_global / k_on_ave;

        for(i=0;i<numface_t;++i){
        beans_avail_t[i][mlct] = beans_tc_face[i][mlct];
            for(j=0;j<numface_p;++j){
                facebeans = beans_avail_t[i][mlct];
                face_sep_dist[i][j]=sqrt(pow(facecen_tx[i]
                -facecen_px[j],2)+pow(facecen_ty[i]-facecen_py[j],2)
                +pow(facecen_tz[i]-facecen_pz[j],2));
                if(face_sep_dist[i][j]<=lambda){
                    kon_local[i][j] = kon_local_old[i][j]
                    * correction * beans_avail_p[j][mlc]/
                    beans_pmn_face[j][mlc];

//      Step 4: calculate probabilities

                    Prob = 1-exp(-1*kon_local[i][j]* *deltatqs);
                    //Prob = 1;
                    for(b=0;b<facebeans;++b){    // find a random
                    number for every bean on the face
                        rand_num = (double)rand()/(double)RAND_MAX;
                        if(beans_avail_t[i][mlct]-b<1){
```

```
                        // adjust probability for non-integer bean
                            Prob = Prob * (beans_avail_t[i][mlct]-b);
                        }
                        if(Prob > rand_num){
                        // bond has been formed
                            existing_bonds_new[new_bonds][0]=mlct;
                            existing_bonds_new[new_bonds][1]=mlc;
                            existing_bonds_new[new_bonds][2]=i;
                            // MUST BE CHANGED TO AN ID
                            CONSTANT ACROSS TIMESTEPS
                            existing_bonds_new[new_bonds][3]=j;
                            // MUST BE CHANGED TO AN ID
                            CONSTANT ACROSS TIMESTEPS
                            existing_bonds_df[new_bonds][0]=
                            face_sep_dist[i][j];
                            beans_avail_t[i][mlct]-=1;
                            beans_avail_p[j][mlc]-=1;
                            new_bonds+=1;
                        }
                    }
                }
            }
        }
    }

//      Step 5: calculate bond forces

//        fo5=fopen("faces_bond.txt","w");
    for(i=0;i<new_bonds;++i){
        existing_bonds_df[i][1] = spring * (existing_bonds_df[i][0]
         - lambda);
        tb = existing_bonds_new[i][2];
        pb = existing_bonds_new[i][3];
        force_ad_x[i] += existing_bonds_df[i][1] *
        (facecen_px[pb]-facecen_tx[tb])/existing_bonds_df[i][0];
        force_ad_y[i] += existing_bonds_df[i][1] *
        (facecen_py[pb]-facecen_ty[tb])/existing_bonds_df[i][0];
        force_ad_z[i] += existing_bonds_df[i][1] *
        (facecen_pz[pb]-facecen_tz[tb])/existing_bonds_df[i][0];
//          fprintf(fo5,"%20.13e %20.13e %20.13e %20.13e
```

```
%20.13e %20.13e \n", facecen_tx[tb],facecen_ty[tb],facecen_tz[tb],
facecen_px[pb],facecen_py[pb],facecen_pz[pb]);


            *biochem_adh_xforce+= force_ad_x[i];
            *biochem_adh_yforce+= force_ad_y[i];
            *biochem_adh_zforce+= force_ad_z[i];
            *biochem_adh_xytorque+=force_ad_y[i]*(facecen_tx[tb]
            -centroidx8)-force_ad_x[i]*(facecen_ty[tb]-centroidy8);
            *biochem_adh_yztorque+=force_ad_z[i]*(facecen_ty[tb]
            -centroidy8)-force_ad_y[i]*(facecen_tz[tb]-centroidz8);
            *biochem_adh_zxtorque+=force_ad_x[i]*(facecen_tz[tb]
            -centroidz8)-force_ad_z[i]*(facecen_tx[tb]-centroidx8);
        }
//        fclose(fo5);

        }
        }

        num_bonds = new_bonds;
        printf("current bonds = %5d \n", num_bonds);

        fo=fopen("existing_bonds.dat","w");
        fprintf(fo,"%d \n", num_bonds);
        for(i=0;i<num_bonds;++i){
            fprintf(fo,"%d %d %d %d \n", existing_bonds_new[i][0],
            existing_bonds_new[i][1],existing_bonds_new[i][2],
            existing_bonds_new[i][3]);
        }
        fclose(fo);

////////////////////////////////

        temp2dealloc(existing_bonds,exb_size);
        temp2dealloc(existing_bonds_new,exb_size);
        temp2dealloc(existing_bonds_df,exb_size);
        temp2dealloc(kon_local_old,numface_t);
        temp2dealloc(kon_local,numface_t);
        temp2dealloc(face_sep_dist,numface_t);
```

```
        temp2dealloc(beans_avail_t,ic1g+icpfTC0);
        temp2dealloc(beans_avail_p,ic2g+icpfPMN0);
        temp2dealloc(beans_tc_face,ic1g+icpfTC0);
        temp2dealloc(beans_pmn_face,ic2g+icpfPMN0);

temprdealloc(facecen_tx);
temprdealloc(facecen_ty);
temprdealloc(facecen_tz);
        temprdealloc(facearea_t);
        temprdealloc(bond_dist);

temprdealloc(facecen_px);
temprdealloc(facecen_py);
temprdealloc(facecen_pz);
        temprdealloc(facearea_p);

}

        tempdealloc(countF);
        tempdealloc(offsetF);
        tempdealloc(countF0);
        tempdealloc(offsetF0);

        temprdealloc(locXIB);
        temprdealloc(locYIB);
        temprdealloc(locZIB);
        temprdealloc(locAIB);
        tempdealloc(locFlagIB);

        temprdealloc(locXW);
        temprdealloc(locYW);
        temprdealloc(locZW);
        temprdealloc(locAW);
        tempdealloc(locFlagW);

        temprdealloc(globXIB);
        temprdealloc(globYIB);
        temprdealloc(globZIB);
        temprdealloc(globAIB);
        tempdealloc(globFlagIB);
```

```
        temprdealloc(globXW);
        temprdealloc(globYW);
        temprdealloc(globZW);
        temprdealloc(globAW);
        tempdealloc(globFlagW);

        temprdealloc(force_ad_x);
        temprdealloc(force_ad_y);
        temprdealloc(force_ad_z);



}

exiting("julie_biochem_adhesion") ;


return 0 ;

}
```

Appendix D

Python Control Script

This file is used to dictate the overall parameters of the simulation to be run through NPHASE. It creates a mesh, based on input parameters of the overall mesh size, and the size and initial centroid locations of the two cells. This file controls the mesh creation, running cobalt, fump, NPHASE, and creating output files which can be viewed in Ensight. A directory must be created including this file (with the directory path specified in the opening line of the file), a copy of NPHASE, the directory "*system*", and the files "*foamMeshToCobalt*" and "*immersed_convert*".

```
#!/usr/bin/python
#PBS -N TCPMN_PW
#PBS -l walltime=48:00:00
#PBS -j oe
#PBS -l nodes=1:ppn=12
##PBS -l nodes=1

# cellcfd.py
# Original Code written by John Bistline Summer 2006 - cfdkin.py
# Written by Byron Gaskin - Fall 2012
# Current capability - import PMN deformed shape from previous deformation run,
# TC 6DOF motion, both cells rigid.
# Meshing done using Pointwise

# Import classes needed for programs
import commands, math, os, time, sys, string, random, subprocess

os.chdir('/home/jmb876/nphase_julie/python_scripts/Apr9A')
workdir = os.getcwd()
```

```
print workdir

runnum = 0

################################################
################################################
################################################
################################################
################################################




# function definitions

def initMesh():
        #This section creates an ICEM batch script for initializtion
         of the desired geometry
        output = file('pwRun/cellMechanics.ICEM.inp1','w')


## Make TC

        output.write('ic_geo_new_family GEOM\n')
        output.write('ic_boco_set_part_color GEOM\n')
        output.write('ic_surface sphere GEOM srf.00 {{' + str
        (c_xcent) + ' ' + str(c_ycent) + ' ' + str(c_zcent) + '
        } {0 0 2} ' + str(c_rad) + ' 0 180}\n')
        output.write('ic_set_dormant_pickable point 0 {}\n')
        output.write('ic_set_dormant_pickable curve 0 {}\n')
        output.write('ic_geo_new_family BODY\n')
        output.write('ic_boco_set_part_color BODY\n')
        output.write('ic_geo_build_bodies BODY\n')
        output.write('ic_geo_delete_family LUMP\n')
        output.write('ic_geo_delete_family SHEET\n')
        output.write('ic_geo_delete_family SHELL\n')
        output.write('ic_geo_set_family_params GEOM no_crv_inf
        prism 0 emax 0.5 ehgt 0.0 hrat 0 nlay 0 erat 0 ewid 0
        emin 0.0 edev 0.0 split_wall 0 internal_wall 0\n')
        output.write('ic_geo_params_blank_done part 1\n')
```

```
output.write('ic_quad2 what surfaces entities {} element
2 proj 1 conver 0.025 geo_tol 0 ele_tol 0.01 dev 0.0
improvement 1 block 0.2 bunch 0 debug 0 adjust_nodes 0
adjust_nodes_max 0 try_harder 1 error_subset Failed_surfaces
 pattern 150 big 1 board 0 remove_old -1 inner 0 simple_offset
  0 enn 0 b_smooth 0 time_max 0 ele_max 0 four 0 merge_dormant
   1 max_length 0.0 max_area 0.0 min_angle 0.0 max_nodes 0
   max_elements 0 smoothdormant 0 breakpoint 0 freeb 0
   n_threads 1 snorm 1 shape 0\n')
output.write('ic_batch_mode\n')
output.write('ic_get_ignore_size\n')
output.write('ic_uns_update_family_type visible {GEOM O
RFN BODY} {!NODE LINE_2 TRI_3 QUAD_4} update 0\n')
output.write('ic_uns_create_selection_subset 0\n')
output.write('ic_uns_create_selection_edgelist 1\n')
output.write('ic_uns_subset_configure uns_sel_0 -draw_nod
es 1\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_subset_create\n')
output.write('ic_uns_subset_copy uns_sub_0 Selected 2d\n')
output.write('ic_uns_subset_copy uns_sel_0 uns_sub_0\n')
output.write('ic_uns_subset_add_from uns_sel_0 uns_sub_0\n')
output.write('ic_uns_subset_delete uns_sub_0\n')
output.write('ic_uns_uniqify uns_sel_0\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_create_selection_edgelist 0\n')
output.write('ic_uns_change_type uns_sel_0 quad tri 0\n')
output.write('ic_uns_subset_delete uns_sel_0\n')
output.write('ic_uns_update_family_type visible {GEOM ORFN
BODY} {!NODE LINE_2 TRI_3} update 0\n')
output.write('ic_uns_create_selection_subset 0\n')
output.write('ic_uns_create_selection_edgelist 1\n')
output.write('ic_uns_subset_configure uns_sel_0 -draw_nodes
 1\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_subset_create\n')
output.write('ic_uns_subset_make_all uns_sel_0\n')
output.write('ic_uns_subset_subtract_from uns_sel_0 Selected
\n')
output.write('ic_uns_uniqify uns_sel_0\n')
```

```
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_create_selection_edgelist 0\n')
output.write('ic_uns_change_type uns_sel_0 quad tri 0\n')
output.write('ic_uns_subset_delete uns_sel_0\n')
output.write('ic_uns_update_family_type visible {GEOM ORFN
BODY} {!NODE LINE_2 TRI_3} update 0\n')
output.write('ic_uns_create_selection_subset 0\n')
output.write('ic_uns_create_selection_edgelist 1\n')
output.write('ic_uns_subset_configure uns_sel_0 -draw_nodes
1\n')
output.write('ic_uns_uniqify uns_sel_0\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_create_selection_edgelist 0\n')
output.write('ic_uns_subset_delete uns_sel_0\n')
output.write('ic_uns_update_family_type visible {GEOM ORFN
BODY} {NODE LINE_2 TRI_3} update 0\n')
output.write('ic_boco_solver STL\n')
output.write('ic_solver_mesh_info STL\n')
output.write('ic_boco_solver\n')
output.write('ic_boco_save ' + workdir + '/pwRun/nastran.
fbc\n')
output.write('ic_boco_save_atr ' + workdir + '/pwRun/nastra
n.atr\n')
output.write('ic_save_tetin project1.tin 0 0 {} {} 0 0 1\n')
output.write('ic_uns_check_duplicate_numbers\n')
output.write('ic_uns_renumber_elements all 1 1\n')
output.write('ic_save_unstruct project1.uns 1 {} {} {}\n')
output.write('ic_uns_set_modified 1\n')
output.write('ic_boco_solver\n')
output.write('ic_boco_save project1.fbc\n')
output.write('ic_boco_save_atr project1.atr\n')
output.write('ic_cart_is_loaded\n')
output.write('ic_exec
/software/ansys/v120/icemcfd/linux64_amd
/icemcfd/output-interfaces/
stl ' + workdir + '/pwRun/project1.uns ' + workdir + '/pwRun
/tc.stl ascii no_shift\n')
output.write('ic_boco_unload\n')
output.write('ic_unload_mesh\n')
output.write('ic_uns_diag_reset_degen_min_max\n')
```

```
        output.write('ic_csystem_display all 0\n')
        output.write('ic_csystem_set_current global\n')
        output.write('ic_unload_tetin\n')
        output.write('ic_empty_tetin\n')
        output.write('ic_geo_set_modified 0\n')
        output.write('ic_cart_is_loaded\n')
        output.write('ic_csystem_display all 0\n')
        output.write('ic_csystem_set_current global\n')
        output.write('ic_geo_set_modified 0\n')
        output.write('ic_vsettings -delete all\n')

## Make PMN
output.write('ic_geo_new_family GEOM\n')
        output.write('ic_boco_set_part_color GEOM\n')
        output.write('ic_surface sphere GEOM srf.00 {{0 0 0} {0 0 1}
         1 0 180}\n')
        output.write('ic_set_dormant_pickable point 0 {}\n')
        output.write('ic_set_dormant_pickable curve 0 {}\n')
        output.write('ic_geo_new_family BODY\n')
        output.write('ic_boco_set_part_color BODY\n')
        output.write('ic_geo_build_bodies BODY\n')
        output.write('ic_geo_delete_family LUMP\n')
        output.write('ic_geo_delete_family SHEET\n')
        output.write('ic_geo_delete_family SHELL\n')
        output.write('ic_geo_set_family_params GEOM no_crv_inf prism
        0 emax 0.1 ehgt 0.0 hrat 0 nlay 0 erat 0 ewid 0 emin 0.0
        edev
         0.0 split_wall 0 internal_wall 0\n')
        output.write('ic_geo_params_blank_done part 1\n')
        output.write('ic_quad2 what surfaces entities {} element 2
        pr
        oj 1 conver 0.025 geo_tol 0 ele_tol 0.01 dev 0.0 improvement
        1 block 0.2 bunch 0 debug 0 adjust_nodes 0 adjust_nodes_max
        0
         try_harder 1 error_subset Failed_surfaces pattern 150 big 1
         board 0 remove_old -1 inner 0 simple_offset 0 enn 0
         b_smooth
          0 time_max 0 ele_max 0 four 0 merge_dormant 1 max_length
          0.
          0 max_area 0.0 min_angle 0.0 max_nodes 0 max_elements 0
```

```
  smoothdormant 0 breakpoint 0 freeb 0 n_threads 1 snorm 1
  shape
   0\n')
output.write('ic_batch_mode\n')
output.write('ic_get_ignore_size\n')
output.write('ic_uns_update_family_type visible {GEOM ORFN
BODY} {!NODE LINE_2 TRI_3 QUAD_4} update 0\n')
output.write('ic_uns_create_selection_subset 0\n')
output.write('ic_uns_create_selection_edgelist 1\n')
output.write('ic_uns_subset_configure uns_sel_0 -
draw_nodes 1
\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_subset_create\n')
output.write('ic_uns_subset_copy uns_sub_0 Selected 2d\n')
output.write('ic_uns_subset_copy uns_sel_0 uns_sub_0\n')
output.write('ic_uns_subset_add_from uns_sel_0 uns_sub_0\n')
output.write('ic_uns_subset_delete uns_sub_0\n')
output.write('ic_uns_uniqify uns_sel_0\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_create_selection_edgelist 0\n')
output.write('ic_uns_change_type uns_sel_0 quad tri 0\n')
output.write('ic_uns_subset_delete uns_sel_0\n')
output.write('ic_uns_update_family_type visible {GEOM ORFN
BODY} {!NODE LINE_2 TRI_3} update 0\n')
output.write('ic_uns_create_selection_subset 0\n')
output.write('ic_uns_create_selection_edgelist 1\n')
output.write('ic_uns_subset_configure uns_sel_0 -draw_nodes
1\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_subset_create\n')
output.write('ic_uns_subset_make_all uns_sel_0\n')
output.write('ic_uns_subset_subtract_from uns_sel_0
Selected\
n')
output.write('ic_uns_uniqify uns_sel_0\n')
output.write('ic_uns_subset_visible uns_sel_0 0\n')
output.write('ic_uns_create_selection_edgelist 0\n')
output.write('ic_uns_change_type uns_sel_0 quad tri 0\n')
output.write('ic_uns_subset_delete uns_sel_0\n')
```

```
        output.write('ic_uns_update_family_type visible {GEOM ORFN
        BODY} {!NODE LINE_2 TRI_3} update 0\n')
        output.write('ic_uns_create_selection_subset 0\n')
        output.write('ic_uns_create_selection_edgelist 1\n')
        output.write('ic_uns_subset_configure uns_sel_0 -draw_nodes
        1\n')
        output.write('ic_uns_uniqify uns_sel_0\n')
        output.write('ic_uns_subset_visible uns_sel_0 0\n')
        output.write('ic_uns_create_selection_edgelist 0\n')
        output.write('ic_uns_subset_delete uns_sel_0\n')
        output.write('ic_uns_update_family_type visible {GEOM ORFN
        BODY} {NODE LINE_2 TRI_3} update 0\n')
        output.write('ic_boco_solver STL\n')
        output.write('ic_solver_mesh_info STL\n')
        output.write('ic_boco_solver\n')
        output.write('ic_boco_save ' + workdir + '/pwRun/nastran.fbc
        \n')
        output.write('ic_boco_save_atr ' + workdir + '/pwRun/nastran
        .atr\n')
        output.write('ic_save_tetin project1.tin 0 0 {} {} 0 0 1\n')
        output.write('ic_uns_check_duplicate_numbers\n')
        output.write('ic_uns_renumber_elements all 1 1\n')
        output.write('ic_save_unstruct project1.uns 1 {} {} {}\n')
        output.write('ic_uns_set_modified 1\n')
        output.write('ic_boco_solver\n')
        output.write('ic_boco_save project1.fbc\n')
        output.write('ic_boco_save_atr project1.atr\n')
        output.write('ic_cart_is_loaded\n')
        output.write('ic_exec
        /software/ansys/v120/icemcfd/linux64_amd/icemcfd/output-
        interfaces/stl ' + workdir + '/pwRun/project1.uns ' +
        workdir + '/pwRun/
        unitSphere.stl ascii no_shift\n')

## Close File
        output.close()



        #This section creates a Pointwise batch script for initializ
```

```
        ation of the desired geometry
        output2 = file('pwRun/cellMechanics.PW.inp1.glf','w')

        output2.write('# Pointwise V17.0 Journal file \n\n')

        output2.write('package require PWI_Glyph 2.17.0\n\n')

output2.write('pw::Application setUndoMaximumLevels 5\n')
        output2.write('pw::Application reset\n')

        output2.write('pw::Application clearModified\n\n')

        output2.write('pw::Application setCAESolver {OpenFOAM} 3\n\n')

        output2.write('set _TMP(mode_10) [pw::Application begin GridI
        mport]\n')
        output2.write('  $_TMP(mode_10) initialize -type Automatic {'
         + workdir + '/pwRun/pmn.stl}\n')
        output2.write('  $_TMP(mode_10) read\n')
        output2.write('  $_TMP(mode_10) convert\n')
        output2.write('$_TMP(mode_10) end\n')
        output2.write('unset _TMP(mode_10)\n')

        output2.write('set _TMP(mode_10) [pw::Application begin GridIm
        port]\n')
        output2.write('  $_TMP(mode_10) initialize -type Automatic {'
        + workdir + '/pwRun/tc.stl}\n')
        output2.write('  $_TMP(mode_10) read\n')
        output2.write('  $_TMP(mode_10) convert\n')
        output2.write('$_TMP(mode_10) end\n')
        output2.write('unset _TMP(mode_10)\n')

        output2.write('pw::Display resetView -Z\n')
        output2.write('set _TMP(mode_10) [pw::Application begin Creat
        e]\n')
        output2.write('  set _TMP(PW_10) [pw::SegmentSpline create]\n')
        output2.write('  $_TMP(PW_10) addPoint {0 0 0}\n')
        output2.write('  $_TMP(PW_10) addPoint {' + str(xLen) + ' 0 0
        }\n')
        output2.write('  set _TMP(con_7) [pw::Connector create]\n')
```

```
output2.write('  $_TMP(con_7) addSegment $_TMP(PW_10)\n')
output2.write('  unset _TMP(PW_10)\n')
output2.write('  $_TMP(con_7) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Creat
e]\n')
output2.write('  set _TMP(PW_11) [pw::SegmentSpline create]\
n')
output2.write('  $_TMP(PW_11) addPoint {' + str(xLen) + ' 0
0}\n')
output2.write('  $_TMP(PW_11) addPoint {' + str(xLen) + ' '
+ str(yLen) + ' 0}\n')
output2.write('  unset _TMP(con_7)\n')
output2.write('  set _TMP(con_8) [pw::Connector create]\n')
output2.write('  $_TMP(con_8) addSegment $_TMP(PW_11)\n')
output2.write('  unset _TMP(PW_11)\n')
output2.write('  $_TMP(con_8) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(PW_12) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_12) addPoint {' + str(xLen) + ' '
+ str(yLen) + ' 0}\n')
output2.write('  $_TMP(PW_12) addPoint {0 ' + str(yLen) + '
0}\n')
output2.write('  unset _TMP(con_8)\n')
output2.write('  set _TMP(con_9) [pw::Connector create]\n')
output2.write('  $_TMP(con_9) addSegment $_TMP(PW_12)\n')
output2.write('  unset _TMP(PW_12)\n')
output2.write('  $_TMP(con_9) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
```

```
output2.write('  set _TMP(PW_13) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_13) addPoint {0 ' + str(yLen) + '
 0}\n')
output2.write('  $_TMP(PW_13) addPoint {0 0 0}\n')
output2.write('  unset _TMP(con_9)\n')
output2.write('  set _TMP(con_10) [pw::Connector create]\n')
output2.write('  $_TMP(con_10) addSegment $_TMP(PW_13)\n')
output2.write('  unset _TMP(PW_13)\n')
output2.write('  $_TMP(con_10) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(PW_14) [pw::SegmentSpline create]
\n')
output2.write('  set _DM(1) [pw::GridEntity getByName "dom-
3"]\n')
output2.write('  set _DM(2) [pw::GridEntity getByName "dom-
4"]\n')
output2.write('  pw::Display resetView -Z\n')
output2.write('  $_TMP(PW_14) addPoint {0 0 0}\n')
output2.write('  $_TMP(PW_14) addPoint {0 0 ' + str(zLen) +
 '}\n')
output2.write('  unset _TMP(con_10)\n')
output2.write('  set _TMP(con_11) [pw::Connector create]\n')
output2.write('  $_TMP(con_11) addSegment $_TMP(PW_14)\n')
output2.write('  unset _TMP(PW_14)\n')
output2.write('  $_TMP(con_11) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cr
eate]\n')
output2.write('  set _TMP(PW_15) [pw::SegmentSpline create
]\n')
output2.write('  $_TMP(PW_15) addPoint {0 0 ' + str(zLen)
+ '}\n')
output2.write('  $_TMP(PW_15) addPoint {0 ' + str(yLen) +
```

```
' ' + str(zLen) + '}\n')
output2.write('  unset _TMP(con_11)\n')
output2.write('  set _TMP(con_12) [pw::Connector create]\n')
output2.write('  $_TMP(con_12) addSegment $_TMP(PW_15)\n')
output2.write('  unset _TMP(PW_15)\n')
output2.write('  $_TMP(con_12) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cr
eate]\n')
output2.write('  set _TMP(PW_16) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_16) addPoint {0 ' + str(yLen) +
' ' + str(zLen) + '}\n')
output2.write('  $_TMP(PW_16) addPoint {0 ' + str(yLen) + '
 0}\n')
output2.write('  unset _TMP(con_12)\n')
output2.write('  set _TMP(con_13) [pw::Connector create]\n')
output2.write('  $_TMP(con_13) addSegment $_TMP(PW_16)\n')
output2.write('  unset _TMP(PW_16)\n')
output2.write('  $_TMP(con_13) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(PW_17) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_17) addPoint {0 ' + str(yLen) + '
' + str(zLen) + '}\n')
output2.write('  $_TMP(PW_17) addPoint {' + str(xLen) + '
' + str(yLen) + ' ' + str(zLen) + '}\n')
output2.write('  unset _TMP(con_13)\n')
output2.write('  set _TMP(con_14) [pw::Connector create]\n')
output2.write('  $_TMP(con_14) addSegment $_TMP(PW_17)\n')
output2.write('  unset _TMP(PW_17)\n')
output2.write('  $_TMP(con_14) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
```

```
output2.write('set _TMP(mode_10) [pw::Application begin Crea
te]\n')
output2.write('  set _TMP(PW_18) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_18) addPoint {' + str(xLen) + ' '
 + str(yLen) + ' ' + str(zLen) + '}\n')
output2.write('  $_TMP(PW_18) addPoint {' + str(xLen) + ' '
+ str(yLen) + ' 0}\n')
output2.write('  unset _TMP(con_14)\n')
output2.write('  set _TMP(con_15) [pw::Connector create]\n')
output2.write('  $_TMP(con_15) addSegment $_TMP(PW_18)\n')
output2.write('  unset _TMP(PW_18)\n')
output2.write('  $_TMP(con_15) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(PW_19) [pw::SegmentSpline create]
\n')
output2.write('  pw::Display resetView -Z\n')
output2.write('  $_TMP(PW_19) addPoint {' + str(xLen) + ' '
 + str(yLen) + ' ' + str(zLen) + '}\n')
output2.write('  $_TMP(PW_19) addPoint {' + str(xLen) + ' 0
 ' + str(zLen) + '}\n')
output2.write('  unset _TMP(con_15)\n')
output2.write('  set _TMP(con_16) [pw::Connector create]\n')
output2.write('  $_TMP(con_16) addSegment $_TMP(PW_19)\n')
output2.write('  unset _TMP(PW_19)\n')
output2.write('  $_TMP(con_16) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Crea
te]\n')
output2.write('  set _TMP(PW_20) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_20) addPoint {' + str(xLen) + ' 0
 ' + str(zLen) + '}\n')
```

```
output2.write('  $_TMP(PW_20) addPoint {' + str(xLen) + ' 0
 0}\n')
output2.write('  unset _TMP(con_16)\n')
output2.write('  set _TMP(con_17) [pw::Connector create]\n')
output2.write('  $_TMP(con_17) addSegment $_TMP(PW_20)\n')
output2.write('  unset _TMP(PW_20)\n')
output2.write('  $_TMP(con_17) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(PW_21) [pw::SegmentSpline create]
\n')
output2.write('  $_TMP(PW_21) addPoint {' + str(xLen) + ' 0
' + str(zLen) + '}\n')
output2.write('  $_TMP(PW_21) addPoint {0 0 ' + str(zLen) +
'}\n')
output2.write('  unset _TMP(con_17)\n')
output2.write('  set _TMP(con_18) [pw::Connector create]\n')
output2.write('  $_TMP(con_18) addSegment $_TMP(PW_21)\n')
output2.write('  unset _TMP(PW_21)\n')
output2.write('  $_TMP(con_18) calculateDimension\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Crea
te]\n')
output2.write('  set _TMP(PW_22) [pw::SegmentSpline create]\
n')
output2.write('  $_TMP(PW_22) delete\n')
output2.write('  unset _TMP(PW_22)\n')
output2.write('$_TMP(mode_10) abort\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('unset _TMP(con_18)\n')
output2.write('pw::Application setGridPreference Unstructur
ed\n')
output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('$_TMP(mode_10) abort\n')
```

```
output2.write('unset _TMP(mode_10)\n')
output2.write('set _CN(1) [pw::GridEntity getByName "con-8
"]\n')
output2.write('set _CN(2) [pw::GridEntity getByName "con-13
"]\n')
output2.write('set _CN(3) [pw::GridEntity getByName "con-1
5"]\n')
output2.write('set _CN(4) [pw::GridEntity getByName "con-5
"]\n')
output2.write('set _CN(5) [pw::GridEntity getByName "con-1
6"]\n')
output2.write('set _CN(6) [pw::GridEntity getByName "con-7
"]\n')
output2.write('set _CN(7) [pw::GridEntity getByName "con-1
0"]\n')
output2.write('set _CN(8) [pw::GridEntity getByName "con-9
"]\n')
output2.write('set _CN(9) [pw::GridEntity getByName "con-6
"]\n')
output2.write('set _CN(10) [pw::GridEntity getByName "con-
12"]\n')
output2.write('set _CN(11) [pw::GridEntity getByName "con-
11"]\n')
output2.write('set _CN(12) [pw::GridEntity getByName "con-
14"]\n')
output2.write('set _TMP(PW_23) [pw::Collection create]\n')
output2.write('$_TMP(PW_23) set [list $_CN(1) $_CN(2) $_CN
(3) $_CN(4) $_CN(5) $_CN(6) $_CN(7) $_CN(8) $_CN(9) $_CN(10
) $_CN(11) $_CN(12)]\n')
output2.write('$_TMP(PW_23) do setDimension 15\n')
output2.write('$_TMP(PW_23) delete\n')
output2.write('unset _TMP(PW_23)\n')

output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(edge_1) [pw::Edge create]\n')
output2.write('  $_TMP(edge_1) addConnector $_CN(11)\n')
output2.write('  $_TMP(edge_1) addConnector $_CN(1)\n')
output2.write('  $_TMP(edge_1) addConnector $_CN(8)\n')
output2.write('  $_TMP(edge_1) addConnector $_CN(7)\n')
```

```
output2.write('  set _TMP(dom_1) [pw::DomainUnstructured cr
eate]\n')
output2.write('  $_TMP(dom_1) addEdge $_TMP(edge_1)\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('unset _TMP(dom_1)\n')

output2.write('unset _TMP(edge_1)\n')
output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(edge_2) [pw::Edge create]\n')
output2.write('  $_TMP(edge_2) addConnector $_CN(7)\n')
output2.write('  $_TMP(edge_2) addConnector $_CN(5)\n')
output2.write('  $_TMP(edge_2) addConnector $_CN(12)\n')
output2.write('  $_TMP(edge_2) addConnector $_CN(10)\n')
output2.write('  set _TMP(dom_2) [pw::DomainUnstructured cre
ate]\n')
output2.write('  $_TMP(dom_2) addEdge $_TMP(edge_2)\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('unset _TMP(dom_2)\n')

output2.write('unset _TMP(edge_2)\n')
output2.write('set _TMP(mode_10) [pw::Application begin Cre
ate]\n')
output2.write('  set _TMP(edge_3) [pw::Edge create]\n')
output2.write('  $_TMP(edge_3) addConnector $_CN(11)\n')
output2.write('  $_TMP(edge_3) addConnector $_CN(10)\n')
output2.write('  $_TMP(edge_3) addConnector $_CN(2)\n')
output2.write('  $_TMP(edge_3) addConnector $_CN(6)\n')
output2.write('  set _TMP(dom_3) [pw::DomainUnstructured cre
ate]\n')
output2.write('  $_TMP(dom_3) addEdge $_TMP(edge_3)\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('unset _TMP(dom_3)\n')

output2.write('unset _TMP(edge_3)\n')
output2.write('set _TMP(mode_10) [pw::Application begin Crea
te]\n')
```

```
output2.write('  set _TMP(edge_4) [pw::Edge create]\n')
output2.write('  $_TMP(edge_4) addConnector $_CN(1)\n')
output2.write('  $_TMP(edge_4) addConnector $_CN(6)\n')
output2.write('  $_TMP(edge_4) addConnector $_CN(9)\n')
output2.write('  $_TMP(edge_4) addConnector $_CN(4)\n')
output2.write('  set _TMP(dom_4) [pw::DomainUnstructured cre
ate]\n')
output2.write('  $_TMP(dom_4) addEdge $_TMP(edge_4)\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('unset _TMP(dom_4)\n')

output2.write('unset _TMP(edge_4)\n')
output2.write('set _TMP(mode_10) [pw::Application begin
Create]\n')
output2.write('  set _TMP(edge_5) [pw::Edge create]\n')
output2.write('  $_TMP(edge_5) addConnector $_CN(3)\n')
output2.write('  $_TMP(edge_5) addConnector $_CN(9)\n')
output2.write('  $_TMP(edge_5) addConnector $_CN(2)\n')
output2.write('  $_TMP(edge_5) addConnector $_CN(12)\n')
output2.write('  set _TMP(dom_5) [pw::DomainUnstructured
create]\n')
output2.write('  $_TMP(dom_5) addEdge $_TMP(edge_5)\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('unset _TMP(dom_5)\n')

output2.write('unset _TMP(edge_5)\n')
output2.write('set _TMP(mode_10) [pw::Application begin
Create]\n')
output2.write('  set _TMP(edge_6) [pw::Edge create]\n')
output2.write('  $_TMP(edge_6) addConnector $_CN(3)\n')
output2.write('  $_TMP(edge_6) addConnector $_CN(4)\n')
output2.write('  $_TMP(edge_6) addConnector $_CN(8)\n')
output2.write('  $_TMP(edge_6) addConnector $_CN(5)\n')
output2.write('  set _TMP(dom_6) [pw::DomainUnstructured
create]\n')
output2.write('  $_TMP(dom_6) addEdge $_TMP(edge_6)\n')
output2.write('$_TMP(mode_10) end\n')
output2.write('unset _TMP(mode_10)\n')
```

```
output2.write('unset _TMP(dom_6)\n')

output2.write('unset _TMP(edge_6)\n')
output2.write('set _TMP(mode_10) [pw::Application begin
Create]\n')
output2.write('  pw::Display resetView -Z\n')
output2.write('$_TMP(mode_10) abort\n')
output2.write('unset _TMP(mode_10)\n')
output2.write('set _TMP(mode_10) [pw::Application begin
Create]\n')
output2.write('  set _TMP(block_1) [pw::BlockUnstructured
create]\n')
output2.write('  set _DM(3) [pw::GridEntity getByName
"dom-5"]\n')
output2.write('  set _TMP(face_1) [pw::FaceUnstructured
create]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(3)\n')
output2.write('  set _DM(4) [pw::GridEntity getByName
"dom-6"]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(4)\n')
output2.write('  set _DM(5) [pw::GridEntity getByName
"dom-7"]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(5)\n')
output2.write('  set _DM(6) [pw::GridEntity getByName
"dom-8"]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(6)\n')
output2.write('  set _DM(7) [pw::GridEntity getByName
"dom-10"]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(7)\n')
output2.write('  set _DM(8) [pw::GridEntity getByName
"dom-9"]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(8)\n')
output2.write('  $_TMP(block_1) addFace $_TMP(face_1)\n')
output2.write('  set _TMP(face_2) [pw::FaceUnstructured
create]\n')
output2.write('  $_TMP(face_2) addDomain $_DM(1)\n')
output2.write('  $_TMP(face_2) addDomain $_DM(2)\n')
output2.write('  $_TMP(block_1) addFace $_TMP(face_2)\n')
output2.write('  set _DM(9) [pw::GridEntity getByName
"dom-2"]\n')
```

```
        output2.write('  set _TMP(face_3) [pw::FaceUnstructured
        create]\n')
        output2.write('  $_TMP(face_3) addDomain $_DM(9)\n')
        output2.write('  set _DM(10) [pw::GridEntity getByName
        "dom-1"]\n')
        output2.write('  $_TMP(face_3) addDomain $_DM(10)\n')
        output2.write('  $_TMP(block_1) addFace $_TMP(face_3)\n')
        output2.write('$_TMP(mode_10) end\n')
        output2.write('unset _TMP(mode_10)\n')
        output2.write('unset _TMP(block_1)\n')

        output2.write('unset _TMP(face_3)\n')
        output2.write('unset _TMP(face_2)\n')
        output2.write('unset _TMP(face_1)\n')

output2.write('\n')
output2.write('set _TMP(mode_1) [pw::Application begin
Create]\n')
output2.write('  set _TMP(block_1) [pw::BlockUnstructured
create]\n')
output2.write('  set _DM(1) [pw::GridEntity getByName
"dom-1"]\n')
output2.write('  set _TMP(face_1) [pw::FaceUnstructured
create]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(1)\n')
output2.write('  set _DM(2) [pw::GridEntity getByName
"dom-2"]\n')
output2.write('  $_TMP(face_1) addDomain $_DM(2)\n')
output2.write('  $_TMP(block_1) addFace $_TMP(face_1)\n')
output2.write('$_TMP(mode_1) end\n')
output2.write('unset _TMP(mode_1)\n')
output2.write('unset _TMP(block_1)\n')
output2.write('pw::Application markUndoLevel {Assemble
Block}\n')
output2.write('\n')
output2.write('unset _TMP(face_1)\n')
output2.write('set _TMP(mode_2) [pw::Application begin
Create]\n')
output2.write('  set _TMP(block_1) [pw::BlockUnstructured
create]\n')
```

```
output2.write('  set _DM(3) [pw::GridEntity getByName
"dom-3"]\n')
output2.write('  set _TMP(face_2) [pw::FaceUnstructured
create]\n')
output2.write('  $_TMP(face_2) addDomain $_DM(3)\n')
output2.write('  set _DM(4) [pw::GridEntity getByName
"dom-4"]\n')
output2.write('  $_TMP(face_2) addDomain $_DM(4)\n')
output2.write('  $_TMP(block_1) addFace $_TMP(face_2)\n')
output2.write('$_TMP(mode_2) end\n')
output2.write('unset _TMP(mode_2)\n')
output2.write('unset _TMP(block_1)\n')
output2.write('pw::Application markUndoLevel {Assemble
Block}\n')
output2.write('\n')
output2.write('unset _TMP(face_2)\n')
output2.write('\n')
output2.write('set _BL(1) [pw::GridEntity getByName "blk-2"]\n')
output2.write('set _BL(2) [pw::GridEntity getByName "blk-3"]\n')
output2.write('set _BL(3) [pw::GridEntity getByName "blk-1"]\n')
output2.write('set _TMP(mode_5) [pw::Application begin
UnstructuredSolver [list $_BL(1) $_BL(2) $_BL(3)]]\n')
output2.write('  $_TMP(mode_5) run Initialize\n')
output2.write('$_TMP(mode_5) end\n')
output2.write('unset _TMP(mode_5)\n')
output2.write('pw::Application markUndoLevel {Initialize}\n')
output2.write('\n')
output2.write('set _DM(1) [pw::GridEntity getByName "dom-1"]\n')
output2.write('set _DM(2) [pw::GridEntity getByName "dom-2"]\n')
output2.write('set _DM(3) [pw::GridEntity getByName "dom-3"]\n')
output2.write('set _DM(4) [pw::GridEntity getByName "dom-4"]\n')
output2.write('set _TMP(PW_2) [pw::BoundaryCondition getByName
"Unspecified"]\n')
output2.write('set _DM(5) [pw::GridEntity getByName "dom-5"]\n')
output2.write('set _DM(6) [pw::GridEntity getByName "dom-6"]\n')
output2.write('set _DM(7) [pw::GridEntity getByName "dom-7"]\n')
output2.write('set _DM(8) [pw::GridEntity getByName "dom-8"]\n')
output2.write('set _DM(9) [pw::GridEntity getByName "dom-9"]\n')
output2.write('set _DM(10) [pw::GridEntity getByName
"dom-10"]\n')
```

```
output2.write('set _TMP(PW_3) [pw::BoundaryCondition create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_4) [pw::BoundaryCondition getByName
"bc-2"]\n')
output2.write('unset _TMP(PW_3)\n')
output2.write('$_TMP(PW_4) setName "Inflow_00"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_4) apply [list [list $_BL(3)
$_DM(5)]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_5) [pw::BoundaryCondition create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_6) [pw::BoundaryCondition getByName
"bc-3"]\n')
output2.write('unset _TMP(PW_5)\n')
output2.write('$_TMP(PW_6) setName "Pressure_00"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_6) apply [list [list $_BL(3)
$_DM(9)]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_7) [pw::BoundaryCondition create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_8) [pw::BoundaryCondition getByName
"bc-4"]\n')
output2.write('unset _TMP(PW_7)\n')
output2.write('$_TMP(PW_8) setName "Symmetry_00"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_8) apply [list [list $_BL(3) $_DM(6)]
[list $_BL(3) $_DM(8)]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_9) [pw::BoundaryCondition create]\n')
```

```
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_10) [pw::BoundaryCondition getByName
"bc-5"]\n')
output2.write('unset _TMP(PW_9)\n')
output2.write('$_TMP(PW_10) setName "Wall_00"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_10) apply [list [list $_BL(3) $_DM(1)
Same] [list $_BL(3) $_DM(2) Same]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_11) [pw::BoundaryCondition
create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_12) [pw::BoundaryCondition getByName
"bc-6"]\n')
output2.write('unset _TMP(PW_11)\n')
output2.write('$_TMP(PW_12) setName "Wall_01"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_12) apply [list [list $_BL(1) $_DM(1)
Opposite] [list $_BL(1) $_DM(2) Opposite]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_13) [pw::BoundaryCondition
create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_14) [pw::BoundaryCondition getByName
"bc-7"]\n')
output2.write('unset _TMP(PW_13)\n')
output2.write('$_TMP(PW_14) setName "Wall_02"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_14) apply [list [list $_BL(3) $_DM(3)
Same] [list $_BL(3) $_DM(4) Same]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
```

```
output2.write('set _TMP(PW_15) [pw::BoundaryCondition
create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_16) [pw::BoundaryCondition getByName
"bc-8"]\n')
output2.write('unset _TMP(PW_15)\n')
output2.write('$_TMP(PW_16) setName "Wall_03"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_16) apply [list [list $_BL(2) $_DM(3)
Opposite] [list $_BL(2) $_DM(4) Opposite]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_17) [pw::BoundaryCondition
create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_18) [pw::BoundaryCondition getByName
"bc-9"]\n')
output2.write('unset _TMP(PW_17)\n')
output2.write('$_TMP(PW_18) setName "wall_04"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_18) apply [list [list $_BL(3)
$_DM(10)]]\n')
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_19) [pw::BoundaryCondition
create]\n')
output2.write('pw::Application markUndoLevel {Create BC}\n')
output2.write('\n')
output2.write('set _TMP(PW_20) [pw::BoundaryCondition getByName
"bc-10"]\n')
output2.write('unset _TMP(PW_19)\n')
output2.write('$_TMP(PW_20) setName "wall_05"\n')
output2.write('pw::Application markUndoLevel {Name BC}\n')
output2.write('\n')
output2.write('$_TMP(PW_20) apply [list [list $_BL(3)
$_DM(7)]]\n')
```

```
output2.write('pw::Application markUndoLevel {Set BC}\n')
output2.write('\n')
output2.write('unset _TMP(PW_2)\n')
output2.write('unset _TMP(PW_4)\n')
output2.write('unset _TMP(PW_6)\n')
output2.write('unset _TMP(PW_8)\n')
output2.write('unset _TMP(PW_10)\n')
output2.write('unset _TMP(PW_12)\n')
output2.write('unset _TMP(PW_14)\n')
output2.write('unset _TMP(PW_16)\n')
output2.write('unset _TMP(PW_18)\n')
output2.write('unset _TMP(PW_20)\n')
output2.write('set _TMP(mode_6) [pw::Application begin CaeExport
[pw::Entity sort [list $_BL(3) $_BL(1) $_BL(2)]]]\n')
output2.write('  $_TMP(mode_6) initialize -type CAE {' + workdir
+ '/pwRun/constant/polyMesh}\n')
output2.write('  if {![$_TMP(mode_6) verify]} {\n')
output2.write('    error "Data verification failed"\n')
output2.write('  }\n')
output2.write('  $_TMP(mode_6) write\n')
output2.write('$_TMP(mode_6) end\n')
output2.write('unset _TMP(mode_6)\n')

        output2.write('pw::Application save {' + workdir +
        '/pwRun/EntireCase.pw}\n')
        output2.write('pw::Application exit\n')


output2.close()

incellMaker = file('pwRun/incellMaker','w')
        incellMaker.write('1 \n' + str(pmn_rad) + ' \n' +
        str(pmn_xcent) + ' ' + str(pmn_ycent) + ' ' + str(pmn_zcent)
        + ' ')
        incellMaker.close()

# Script Creation Complete

## Execute Scripts
```

```
os.system('cd ' + workdir + '/pwRun/;icemcfd -batch -script
cellMechanics.ICEM.inp1 > icem1.out')
        while not os.path.exists('pwRun/unitSphere.stl'):
                if os.path.exists('pwRun/unitSphere.stl'):
                        break
                pass

os.system('cd ' + workdir + '/pwRun/; python
/home/bjg25/apps/cellMakerDir/cellMaker3.py < incellMaker')
        while not os.path.exists('pwRun/pmn.stl'):
                if os.path.exists('pwRun/pmn.stl'):
                        break
                pass

os.system('cd ' + workdir + '/pwRun/; pointwise -b
cellMechanics.PW.inp1.glf')
        while not os.path.exists('pwRun/EntireCase.pw'):
                if os.path.exists('pwRun/EntireCase.pw'):
                        break
                pass


def runPW():

output = file('pwRun/cellMechanics.PW.inp2.glf','w')

        output.write('# Pointwise V17.0 Journal file\n\n')

        output.write('package require PWI_Glyph 2.17.0\n\n')

        output.write('pw::Application setUndoMaximumLevels 5\n')
        output.write('pw::Application reset\n')

        output.write('pw::Application clearModified\n\n')

        output.write('pw::Application reset -keep Clipboard\n')
        output.write('set _TMP(mode_6) [pw::Application begin
        ProjectLoader]\n')
        output.write('  $_TMP(mode_6) initialize {' + workdir +
        '/pwRun/EntireCase.pw}\n')
```

```
output.write('  $_TMP(mode_6) setAppendMode false\n')
output.write('  $_TMP(mode_6) load\n')
output.write('$_TMP(mode_6) end\n')
output.write('unset _TMP(mode_6)\n')
output.write('pw::Application resetUndoLevels\n')
output.write('set _DM(1) [pw::GridEntity getByName
"dom-3"]\n')
output.write('set _DM(2) [pw::GridEntity getByName
"dom-4"]\n')

output.write('set _TMP(mode_7) [pw::Application begin Modify
[list $_DM(1) $_DM(2)]]\n')
#output.write('  pw::Entity transform [pwu::Transform
translation [pwu::Vector3 subtract {0 1 0} {0 0 0}]]
[$_TMP(mode_7) getEntities]\n')
output.write('  pw::Entity transform [pwu::Transform
translation {' + str(xdisp) + ' ' + str(ydisp) + ' ' +
str(zdisp) + '}] [$_TMP(mode_7) getEntities]\n')
output.write('$_TMP(mode_7) end\n')
output.write('unset _TMP(mode_7)\n')

output.write('set _TMP(mode_8) [pw::Application begin Modify
[list $_DM(1) $_DM(2)]]\n')
output.write('  pw::Entity transform [pwu::Transform
rotation -anchor {' + str(c_xcent) + ' ' + str(c_ycent) + '
' + str(c_zcent) + '} {' + str(xRot) + ' ' + str(yRot) + ' '
+ str(zRot) + '} ' + str(rotMag) + '] [$_TMP(mode_8)
getEntities]\n')
output.write('$_TMP(mode_8) end\n')
output.write('unset _TMP(mode_8)\n')

output.write('pw::Application setGridPreference
Unstructured\n')
output.write('set _BL(1) [pw::GridEntity getByName
"blk-1"]\n')
output.write('set _BL(2) [pw::GridEntity getByName
"blk-2"]\n')
output.write('set _BL(3) [pw::GridEntity getByName
"blk-3"]\n')
output.write('set _TMP(mode_9) [pw::Application begin
```

```
        UnstructuredSolver [list $_BL(1) $_BL(2) $_BL(3)]]\n')
        output.write('  $_TMP(mode_9) run Initialize\n')
        output.write('$_TMP(mode_9) end\n')
        output.write('unset _TMP(mode_9)\n')

        output.write('set _TMP(mode_10) [pw::Application begin
        CaeExport [pw::Entity sort [list $_BL(1) $_BL(2)
        $_BL(3)]]]\n')
        output.write('  $_TMP(mode_10) initialize -type CAE {' +
        workdir + '/pwRun/constant/polyMesh} \n')
        #output.write('  $_TMP(mode_10) setAttribute
        CellCombineAnisotropic true\n')
        output.write('  if {![$_TMP(mode_10) verify]} {\n')
        output.write('    error "Data verification failed"\n')
        output.write('  }\n')
        output.write('  $_TMP(mode_10) write\n')
        output.write('$_TMP(mode_10) end\n')
        output.write('unset _TMP(mode_10)\n')
        output.write('pw::Application save {' + workdir +
        '/pwRun/EntireCase.pw}\n')
        output.write('pw::Application exit\n')

output.close()

        incobaltMod = file('pwRun/incobaltMod','w')
        incobaltMod.write('4\n')
        incobaltMod.close()

## Script Complete ##

tempPW = 0

if runnum > 0:
while (tempPW==0):
os.system('cd pwRun/;pointwise -b
cellMechanics.PW.inp2.glf')

time.sleep(20)

if os.path.exists('pwRun/constant/polyMesh/faces'):
```

```
                        tempPW = 1

        os.system('cp -r system/ pwRun/.')
        while not os.path.exists('pwRun/system/controlDict'):
                if os.path.exists('pwRun/system/controlDict'):
                        break
                pass

        os.system('cp foamMeshToCobalt pwRun/.')
        while not os.path.exists('pwRun/foamMeshToCobalt'):
                if os.path.exists('pwRun/foamMeshToCobalt'):
                        break
                pass

# Fix the immersed functionality!!! cobaltMod not working with
cobalt files generated by foamMeshToCobalt
os.system('cd pwRun/; . /opt/modules/Modules/3.2.5/init/bash ;
module load OpenFOAM/1.6-ext; ./foamMeshToCobalt')

#        os.system('cd pwRun/;python /home/bjg25/apps/cobaltMod2.py
< incobaltMod')
#        os.system('mv pwRun/cobalt.inp.new pwRun/cobalt.inp')
#        os.system('mv pwRun/cobalt.bc.new pwRun/cobalt.bc')

os.system('cp pwRun/cobalt.* nphaseRun/.')
        while not os.path.exists('nphaseRun/cobalt.inp'):
                if os.path.exists('nphaseRun/cobalt.inp'):
                        break
                pass




def runNPHASE():
        #This function handles all preprocessing necessary for
        NPHASE then runs the solver

        output1 = file('nphaseRun/centroid.dat','w')
        output1.write(' ' + str(c_xcent*gridScaling) + ' ' +
        str(c_ycent*gridScaling) + ' ' + str(c_zcent*gridScaling) +
```

```
' ' + str(c_rad*gridScaling) + '\n')
output1.close()

output1 = file('nphaseRun/rotation.dat','w')
output1.write(' ' + str(c_theta) + ' ' + str(c_iota) + ' ' +
str(c_psi) +  ' ' + str(c_omega) + ' ' + str(c_zeta) +  ' '
+ str(c_eta) + '\n')
output1.close()

inFump = file('nphaseRun/inFump','w')
if nprocs == 1:
        inFump.write(' ' + str(nprocs) + ' ' +
        str(gridScaling) + ' 0 1\n')
else:
        inFump.write(' ' + str(nprocs) + ' 0 ' +
        str(gridScaling) + ' 0 1\n')
inFump.close()

inEmerge = file('nphaseRun/inEmerge','w')
inEmerge.write(' ' + str(nprocs) + '\n1 \n')
inEmerge.close()

ndat = file('nphaseRun/nphase.dat','w')
ndat.write('#case title:\n\n')
ndat.write('iterations to perform ' + str(numIter) + '\n\n')
ndat.write('number of fields 1\n\n')

ndat.write('quasisteady simulation explicit\n')
ndat.write('numberofquasisteadytimesteps 1\n')
ndat.write('quasisteadytimestepinseconds ' + str(dt) + '\n')
ndat.write('quasisteadyfilewritefrequency 1\n')
ndat.write('quasisteadytimestepoffset 0\n\n')

ndat.write('pressure based convergence check ' +
str(convPress) + '\n\n')

ndat.write('employ julie biochemistry 1\n')
ndat.write('produce ensight output\nproduce ensight files
with immersed cells\nrestart file iteration write frequency
100\n\n')
```

```
        ndat.write('hard coded inlet 1\n\n')

        ndat.write('dont perform wall match logic\n\n')

#        ndat.write('adaptive meshing limit 1\n')
#        ndat.write('1 0 0 0.025 0. 0.\n')
#        ndat.write('1 1 1 5e-2 5e-2 5e-2\n\n')

        ndat.write('immersed boundaries face based 2\n')
        ndat.write('0 1 0.0 0.0 0.0\n')
        ndat.write('0 2 ' + str(c_xvel) + ' ' + str(c_yvel) + ' ' +
        str(c_zvel) + '\n\n')

        ndat.write('inlet patch 1 0\n')
        ndat.write('0.0042 0. 0. 1000. 1. .0 .0 0. 0. 0 0\n\n')
        ndat.write('pressure patch 1 0\n')
        ndat.write('0. 1000. 1. .0 .0 0. 0.\n\n')
#        ndat.write('wall patch 1 0\n')
#        ndat.write('0 0  0. 0. 0. 0.\n')
#        ndat.write('0 0.\n')
#        ndat.write('1 ' + str(c_xvel) + ' ' + str(c_yvel) + ' ' +
str(c_zvel) + '\n\n')
###################################
#### For small domain only: boundary condition on top wall
        ndat.write('wall patch 1 8\n')
        ndat.write('0 0  0. 0. 0. 0.\n')
        ndat.write('0 0.\n')
        ndat.write('1 ' + str(top_xvel) +' 0. 0.\n\n')
###################################
        ndat.write('constant fluid molecular viscosity 1e-3\n')
        ndat.write('constant fluid density 1000.\n\n')
        ndat.write('functionentry/exitechooff\n\n')
        ndat.write('relaxationfactorforu .95\n\n')
        ndat.write('solver choice for velocity components
        jacobiuvw\n')
        ndat.write('solver choice for pressure petsc\n')
        ndat.write('parallelstrategyforpressurecorrector:matrixlevel\n\n')
        ndat.write('initialize u field 0.00420\n')
        ndat.close()
```

```
rnout = file('nphaseRun/cn.run.nphase','w')
rnout.write('#!/bin/bash\n')
rnout.write('#PBS -l nodes=' + str(nNodes) + ':ppn=' +
str(ppn) + '\n')
rnout.write('#PBS -l walltime=1110:00:00\n')
rnout.write('#PBS -N stokes_cell\n')
rnout.write('#PBS -j oe\n')
rnout.write('##PBS -q batch\n\n')
rnout.write('#MP2BIN=/software/mpich2/1.2.1p1/intel/bin\n\n')
rnout.write('#cd $PBS_O_WORKDIR\n')
rnout.write('echo $PBS_O_WORKDIR\n\n')
rnout.write('export I_MPI_PIN_MODE=mpd\n\n')
rnout.write('# Boot the MPI2 engine.\n')
rnout.write('mpdboot --totalnum=1 --verbose --rsh=ssh --
file=${PBS_NODEFILE}\n\n')
rnout.write('############################\n# Run your
executable\n############################\n\n')
rnout.write('mpiexec -genv I_MPI_FABRICS ofa-v2-ib0 -genv
I_MPI_FALLBACK_DEVICE 0  -ppn ' + str(ppn) + ' -np ' +
str(nprocs) + '  nphase > n.out\n\n')
rnout.write('mpdallexit\n')
rnout.close()

output3 = file('nphaseRun/cobalt.bc','a')
output4 = file('nphaseRun/cobalt.test','w')

for x in range(0, num_imm):
   if x == 0:
      output3.write( str(num_imm) + '\n')
      output4.write( str(num_imm) + '\n')
   output3.write( str( 4+(x*2) ) + ' ' + str( 4+(x*2)+1 ) +
   '\n')
   output4.write( str( 4+(x*2) ) + ' ' + str( 4+(x*2)+1 ) +
   '\n')

output3.close()
output4.close()

os.system('cp immersed_convert nphaseRun/immersed_convert')
```

```
        os.system('cd nphaseRun;chmod 755
        immersed_convert;./immersed_convert')

        os.system('cp nphaseRun/cobalt.inp.immersed
        nphaseRun/cobalt.inp')

        output2 = file('nphaseRun/cobalt.bc','w')

output2.write('################################\n')
output2.write('Boundary condition file for COBALT\n')
output2.write('converted from foam grid\n')

output2.write('################################\n')
output2.write('1\n')
output2.write('Inflow_00\n')
output2.write('Inflow_00\n')
output2.write('Methods: User Created BC\n')
output2.write('User data supplied here - see COBALT doc!\n')

output2.write('################################\n')
output2.write('2\n')
output2.write('Pressure_00\n')
output2.write('Pressure_00\n')
output2.write('Methods: User Created BC\n')
output2.write('User data supplied here - see COBALT doc!\n')

output2.write('################################\n')
output2.write('3\n')
output2.write('Symmetry_00\n')
output2.write('Symmetry_00\n')
output2.write('Methods: User Created BC\n')
output2.write('User data supplied here - see COBALT doc!\n')
output2.write('################################\n')
output2.write('8\n')
output2.write('Wall_04\n')
output2.write('Wall_04\n')
output2.write('Methods: User Created BC\n')
output2.write('User data supplied here - see COBALT doc!\n')
output2.write('################################\n')
```

```
output2.write('9\n')
output2.write('Wall_05\n')
output2.write('Wall_05\n')
output2.write('Methods: User Created BC\n')
output2.write('User data supplied here - see COBALT doc!\n')
output2.write('##################################\n')
        output2.close()

        os.system('cp nphase nphaseRun/nphase')

        os.system('cd nphaseRun;/user/rfk102/NPHASE-PSU/FUMP/fump <
        inFump')

        test_file='nphaseRun/unphase.grid' + "%03d" % (nprocs-1)
        while not os.path.exists(test_file):
                if os.path.exists(test_file):
                        break
                pass

        os.system('cd nphaseRun;chmod 755
        cn.run.nphase;./cn.run.nphase > n.out')

        while not os.path.exists('nphaseRun/nphase_has_completed'):
                if os.path.exists('nphaseRun/nphase_has_completed'):
                        break
                pass

def adaptDt(dt,Fx,Fy,Fz,adDt_old,adF_old):
adForce=math.sqrt(Fx*Fx + Fy*Fy + Fz*Fz);

if runnum == 0:
adDt_old=dt;
adF_old=adForce;

dt_temp= min(adDt_old * (adF_old/adForce),10e-6);
dt=dt_temp;

adDt_old=dt;
adF_old=adForce;
```

```
def runEMERGE():
#Handles file operations needed for post-processing
        inEmerge = file('nphaseRun/inEmerge','w')
        inEmerge.write(str(nprocs) + '\n1 \n')
        inEmerge.close()

#        os.system('cd nphaseRun;/user/rfk102/NPHASE-
PSU/EMERGE/emerge < inEmerge')
        os.system('cd nphaseRun;/home/bjg25/apps/emerge < inEmerge')
        while not os.path.exists('nphaseRun/engold.uvw00.Evec'):
                if os.path.exists('nphaseRun/engold.uvw00.Evec'):
                        break
                pass


        os.system('cp nphaseRun/engold.geo ensightSave/engold.geo.
        %06d' % (runnum))
        os.system('cp nphaseRun/engold.p00.Esca
        ensightSave/engold.p00.Esca.%06d' % (runnum))
        os.system('cp nphaseRun/engold.u00.Esca
        ensightSave/engold.u00.Esca.%06d' % (runnum))
        os.system('cp nphaseRun/engold.v00.Esca
        ensightSave/engold.v00.Esca.%06d' % (runnum))
        os.system('cp nphaseRun/engold.w00.Esca
        ensightSave/engold.w00.Esca.%06d' % (runnum))
        os.system('cp nphaseRun/engold.uvw00.Evec
        ensightSave/engold.uvw00.Evec.%06d' % (runnum))
        os.system('cp nphaseRun/n.out nphaseSave/n.out.%04d' %
        (runnum))
        os.system('cp nphaseRun/nphase.out nphaseSave/nphase.out.
        %04d' % (runnum))


def updateHistory():
        output = file('resid.print','a')
        input = file('nphaseRun/resid.print','r')

        for line in input:
                info_line2 = line
                tmp1,fld,ru,rv,rw,rp,ra,rh,rk,re,gmass =
```

```
                string.split(info_line2)
                tmp2 = tmpCntIter + int(tmp1)
                output.write('%9d %s %s %s %s %s %s %s %s %s %s\n' %
                (tmp2,fld,ru,rv,rw,rp,ra,rh,rk,re,gmass))

        input.close()
        output.close()


        motionfile = file('6dofresTC.txt','a')

        motionfile.write('Iteration #: ' + str(runnum) + '\n')
        motionfile.write('dt: ' + str(dt) + '\n')
        motionfile.write('Input Forces [uN]         :     %3.5e
        %3.5e    %3.5e \n' % (Fx,Fy,Fz))
        motionfile.write('Input Torques [rad/s^2]   :     %3.5e
        %3.5e    %3.5e \n' % (Mz,Mx,My))
        motionfile.write('New Displacement [um]  (r):     %3.5e
        %3.5e    %3.5e \n' % (xdisp,ydisp,zdisp))
        motionfile.write('New Rotations [rad/s]  (r):     %3.5e
        %3.5e    %3.5e \n' % (c_psi,c_iota,c_theta))
        motionfile.write('New Velocity [m/s]     (a):     %3.5e
        %3.5e    %3.5e \n' % (c_xvel,c_yvel,c_zvel))
        motionfile.write('New Centroid           (a):     %3.5e
        %3.5e    %3.5e \n' % (c_xcent,c_ycent,c_zcent))
        motionfile.write('New Orientation        (a):     %3.5e
        %3.5e    %3.5e \n\n' % (c_eta,c_zeta,c_omega))


        motionfile.close()

def createTransient(runnum,dt):
        #This function creates the case file needed to visualize the
        study as a transient flow
        output = file('ensightSave/
        cellMechanics_transient.case','w')

        output.write('# BOF:  cellMechanics_transient
        .case\n\nFORMAT\n\n')
        output.write('type:   ensight gold\n\nGEOMETRY\n\n')
```

```
        output.write('model:  engold.geo.******\n\nVARIABLE\n\n')
        output.write('scalar per element: Pressure
        engold.p00.Esca.******\n')
        output.write('scalar per element: X-velocity_00
        engold.u00.Esca.******\n')
        output.write('scalar per element: Y-velocity_00
        engold.v00.Esca.******\n')
        output.write('scalar per element: Z-velocity_00
        engold.w00.Esca.******\n')
        output.write('vector per element: Velocity_00
        engold.uvw00.Evec.******\n\n')
        output.write('TIME\ntime set: 1\n')
        output.write('number of steps:  ' + str(runnum)  + '\n')
        output.write('filename numbers: \n')

        temp = 0
        while temp < runnum:
                output.write('\t' + str(temp)  + '\n')
                temp = temp + 1

        output.write('time values: \n')

        temp = 0
temp2 = 0
        while temp < runnum:
temp2 = temp2 + dtAd[temp]
                output.write('\t' + str(temp2)  + '\n')
                temp = temp + 1
        output.write('# EOF:  cellMechanics_transient.case\n')

        output.close()



####################################
####################################
####################################
####################################
####################################
```

```
###########
#Create required files and directories

os.system('rm -rf pwRun/')
os.system('rm -rf nphaseRun/')
os.system('rm -rf ensightSave/')
os.system('rm -rf nphaseSave/')
os.system('rm -rf constant/')
os.system('rm -f inFump')
os.system('rm -f run.nphase')
os.system('rm -f resid.print')
os.system('rm -f cellMechanics.*')
os.system('rm -f cellcfd_has_finished')
os.system('rm -f 6dofresTC.txt')
os.system('rm -f timer.txt')
os.system('mkdir nphaseRun')
os.system('mkdir nphaseSave')
os.system('mkdir pwRun')
os.system('mkdir ensightSave')
os.system('mkdir pwRun/constant')
os.system('mkdir pwRun/constant/polyMesh')
os.system('chmod 755 *Run/ *Save/ pwRun/constant/
pwRun/constant/polyMesh/')
os.system('touch resid.print')


motionfile = file('./6dofresTC.txt','w')
motionfile.close()

timefile = file('./timer.txt','w')
t1 = time.asctime()
timefile.write('\nScript start time = ' + str(t1) + '\n\n')
timefile.close()


###########
# Initialize computational parameters

nprocs = 12
```

```
nNodes = 1
ppn = 12

gridScaling = 1e-6

numIter = 1000
masterIter = 0
tmpCntIter = 0
masterDt = 0
tmpCntDt = 0

tol=1e-10

############
# Initialize domain parameters

xLen = 60.0
yLen = 32.0
zLen = 42.0

top_xvel = 4.749705e-3

############
# Initialize case parameters

num_imm=2

# Rad and centroid locations in um
c_rad = 8.0

c_xcent = 20.0
c_ycent = 10.0
c_zcent = 21.0

# Velocity in m/s
c_xvel = 0.0
c_yvel = 0.0
c_zvel = 0.0
c_velMag = 0.0
```

```
c_omega = 0.0
c_zeta = 0.0
c_eta = 0.0

c_theta = 0.0
c_iota = 0.0
c_psi = 0.0
c_rotMag = 0.0

xRot = 0.0  # dth
yRot = 0.0  # dio
zRot = 0.0  # dpsi
rotMag = 0.0 # Simultaneous orthogonal rotation angle

xVec = 0.0 # X component of simultaneous orthogonal rotation angle
vector
yVec = 0.0 # Y component of simultaneous orthogonal rotation angle
vector
zVec = 0.0 # Z component of simultaneous orthogonal rotation angle
vector

xdisp = 0.0
ydisp = 0.0
zdisp = 0.0

pmn_rad = 4

pmn_xcent = 30.0
pmn_ycent = 2.5
pmn_zcent = 21.0

# Computes cell mass
# Density in units of kg/um^3
c_volume = 4.0*math.pi*math.pow(c_rad,3)/3.0
c_density = 1.087e-15
c_mass = c_density * c_volume
c_mass = round(c_mass,14)

# Computes cell moment of inertia in kg*um^2
I_z = (2.00 / 5.00) * c_mass * math.pow(c_rad,2)
```

```
I_y = (2.00 / 5.00) * c_mass * math.pow(c_rad,2)
I_x = (2.00 / 5.00) * c_mass * math.pow(c_rad,2)
I_z = round(I_z,14)
I_y = round(I_y,14)
I_x = round(I_x,14)
I = [I_z,I_y,I_x]

dt = 5e-6
dtAd = []
tol = 1e-10
Fx = 0
Fy = 0
Fz = 0
Mx = 0
My = 0
Mz = 0

adDt_old =  0.0
adF_old  =  0.0

###########
#Run Case!

runFlag = 0
maxRuns = 500

convPress = 1.00e-6

ddt = 1   #Flag for adaptive dt. 1=on, 0=off

initMesh()

#while runFlag < 1:
while runnum < maxRuns:

timefile = file('./timer.txt','a')
        timefile.write('Iteration ' + str(runnum) + '\n')


        t1 = time.asctime()
```

```
        timefile.write('---PW start time = ' + str(t1) + '\n')
        runPW()


        t1 = time.asctime()
        timefile.write('---Nphase start time = ' + str(t1) + '\n')
        runNPHASE()


        t1 = time.asctime()
        timefile.write('---Emerge start time = ' + str(t1) + '\n')
runEMERGE()

##############################
# Save important information #
##############################


        filename = 'nphaseRun/6DofInfo.dat'
        input = file(filename,'r')
        for line in input:
                info_line = line
                ndt,nit,cx,cy,cz,u,v,w,th,io,ps,om,ze,et =
                string.split(info_line)
        input.close()


        filename = 'nphaseRun/qsForces.out'
        input = file(filename,'r')
        for line in input:
                info_line = line
                xf,yf,zf,zm,xm,ym = string.split(info_line)
        input.close()


dt=float(ndt)
dtAd.append(dt)

xdisp = float(cx)/gridScaling - c_xcent
ydisp = float(cy)/gridScaling - c_ycent
```

```
zdisp = float(cz)/gridScaling - c_zcent

        c_xcent = float(cx)/gridScaling
        c_ycent = float(cy)/gridScaling
        c_zcent = float(cz)/gridScaling

        c_xvel = float(u)
        c_yvel = float(v)
        c_zvel = float(w)

        xRot = float(th) - c_theta
        yRot = float(io) - c_iota
        zRot = float(ps) - c_psi

rotMag = math.sqrt(xRot*xRot + yRot*yRot + zRot*zRot)
if (rotMag >= tol):
xVec = xRot/rotMag
yVec = yRot/rotMag
zVec = zRot/rotMag

if (rotMag < tol):
xVec = 0.0
yVec = 0.0
zVec = 0.0

        c_theta=float(th)
        c_iota=float(io)
        c_psi=float(ps)

        c_omega=float(om)
        c_zeta=float(ze)
        c_eta=float(et)

Fx = float(xf)
Fy = float(yf)
Fz = float(zf)

Mz = float(zm)
Mx = float(xm)
My = float(ym)
```

```
        updateHistory()

if ddt == 1:
adaptDt(dt,Fx,Fy,Fz,adDt_old,adF_old)

os.system('rm pwRun/constant/polyMesh/*')

        t1 = time.asctime()
        timefile.write('---iteration complete time = ' + str(t1) +
        '\n')
        timefile.close()

        runnum = runnum + 1

createTransient(runnum,dt)

kinout = file('little_guy_has_finished','w')
kinout.write('cellcfd has finished\n')
kinout.close()

print 'cellcfd.py is complete'

exit
```

Appendix E

MATLAB Adhesion Confirmation Model

This file is the MATLAB routine described in Section 4.2. It generates arrays which represent the hypothetic

```
function[] = adhesion

epsilon = 1.2;
pmnface_x = [0, epsilon/4, epsilon/2, 3*epsilon/4, 0, epsilon/4,
epsilon/2, 3*epsilon/4];
tcface_x = [0, epsilon/4, epsilon/2, 3*epsilon/4, 0, epsilon/4,
epsilon/2, 3*epsilon/4];
pmnface_y = zeros(1,8);
tcface_y = ones(1,8) * epsilon/3;
pmnface_z =
[0,0,0,0,1/4*epsilon,1/4*epsilon,1/4*epsilon,1/4*epsilon,];
tcface_z =
[0,0,0,0,1/4*epsilon,1/4*epsilon,1/4*epsilon,1/4*epsilon,];
pmnface = [pmnface_x;pmnface_y;pmnface_z]';
tcface = [tcface_x;tcface_y;tcface_z]';

facearea = (1/4*epsilon)^2;

faces_t = length(tcface_x);
faces_p = length(pmnface_x);
```

```
pmn_dens = 45;
mlcs_p = ones(1,8) * (pmn_dens*facearea);
sts = 1;


kon_not = 3000;
kons = 0;
distance = zeros(faces_t,faces_p);
square = [0,0,0];
contact_area = 0;
delta_contact = 0;
kon_local_old = zeros(faces_t,faces_p);
kon_local = zeros(faces_t,faces_p);
kon_global = 0;
kon_average = 0;
dist_average = 0;
xid = zeros(faces_t,faces_p);
yid = zeros(faces_t,faces_p);

for i=1:faces_t
    for j=1:faces_p
        xid(i,j) = i;
        yid(i,j) = j;
        for d=1:3
            square(d) = (tcface(i,d)-pmnface(j,d))^2;
        end
        distance(i,j) = sqrt(square(1)+square(2)+square(3));
        if distance(i,j) < epsilon
            delta_contact = facearea;
            kon_local_old(i,j) = mlcs_p(j) * kon_not * exp(-
            sts*(distance(i,j)-epsilon)^2);
            kons = kons + 1;
            kon_average = kon_average + kon_local_old(i,j);
            dist_average = dist_average + distance(i,j);
        end
    end
    contact_area = contact_area + delta_contact;
    delta_contact = 0;
end
```

```
kon_average = kon_average / kons;
dist_average = dist_average / kons;
kon_global = contact_area * pmn_dens * kon_not * exp(-
sts*(dist_average - epsilon)^2);

correction = kon_global / kon_average;

for i=1:faces_t
    for j=1:faces_p
        kon_local(i,j) = kon_local_old(i,j) * correction;
    end
end

figure
plot(distance,kon_local,'o')
title('Local affinity vs local distance')
ylabel('k_on')
xlabel('distance')

figure
surf(xid,yid,kon_local)
view( -45, 60)
title('Local affinity between each face pair')
xlabel('tumor cell face')
ylabel('pmn cell face')
zlabel('k_on')

end
```

Bibliography

[1] Kumar, V., A. K. Abbas, and J. C. Aster (2012) *Robbins basic pathology*, Saunders.

[2] Hoskins, M. (2008) *Development of a Computational Fluid Dynamics Tool to Explore the Interactions Between Cancer Cells and Leukocytes*, Ph.D. thesis, The Pennsylvania State University.

[3] Alberts, B. (1998) *Essential cell biology: an introduction to the molecular biology of the cell*, vol. 1, Garland Pub.

[4] Burkett, R. D. (2006), "Digestive System Lab," .
URL `http://faculty.southwest.tn.edu/rburkett/Digest34.jpg`

[5] Bell, G. I. (1978) "Models for the Specific Adhesion of Cells to Cells," *Science*, **200**, pp. 618–627.

[6] Bell, G. I., M. Dembo, and P. Bongrand (1984) "Cell adhesion. Competition between nonspecific repulsion and specific bonding," *Biophysical Journal*, **45**(6), pp. 1051–1064.

[7] Bongrand, P. and G. Bell (1984) "Cell-cell adhesion: parameters and possible mechanisms," *Cell Surface Dynamics: Concepts and Models. AS Perelson, C. DeLisi, and F. Wiegel, editors. Marcel Dekker, Inc., New York*, pp. 459–493.

[8] Chang, K.-C. and D. A. Hammer (1996) "Influence of direction and type of applied force on the detachment of macromolecularly-bound particles from surfaces," *Langmuir*, **12**(9), pp. 2271–2282.

[9] Chtcheglova, L. A., G. T. Shubeita, S. K. Sekatskii, and G. Dietler (2004) "Force spectroscopy with a small dithering of AFM tip: a method of direct and continuous measurement of the spring constant of single molecules and molecular complexes," *Biophysical journal*, **86**(2), pp. 1177–1184.

[10] Dembo, M. (1994) "On peeling an adherent cell from a surface," *Lectures on Mathematics in the Life Sciences*, **24**, pp. 51–77.

[11] Dembo, M., D. Torney, K. Saxman, and D. Hammer (1988) "The reaction-limited kinetics of membrane-to-surface adhesion and detachment," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **234**(1274), pp. 55–83.

[12] Dong, C., M. J. Slattery, S. Liang, and H.-H. Peng (2005) "Melanoma Cell Extravasation under Flow Conditions is Modulated by Leukocytes and Endogenously Produced Interleukin 8," *Mol Cell Biomech*, **2**(3), pp. 145–159.

[13] Dudjak, L. A. (1992) "Cancer Metastasis," *Seminars in Oncology Nursing*, **8**(1), pp. 40–50.

[14] Elder, D. (1999) "Tumor Progression, Early Diagnosis and Prognosis of Melanoma," *Acta Oncologica*, **38**(5), pp. 535–547.

[15] Evans, E. A. (1983) "Bending elastic modulus of red blood cell membrane derived from buckling instability in micropipet aspiration tests," *Biophysical Journal*, **43**(1), pp. 27–30.

[16] Hammer, D. A. and S. M. Apte (1992) "Simulation of cell rolling and adhesion on surfaces in shear flow: general results and analysis of selectin-mediated neutrophil adhesion," *Biophysical Journal*, **63**, pp. 35–57.

[17] Hanahan, D. and R. A. Weinberg (2000) "The Hallmarks of Cancer," *Cell*, **100**, pp. 57–70.

[18] Health, P. (2012), "Melanoma," .

[19] Jadhav, S., C. D. Eggleton, and K. Konstantopoulos (2005) "A 3-D computational model predicts that cell deformation affects selectin-mediated leukocyte rolling," *Biophysical journal*, **88**(1), pp. 96–104.

[20] Johnson, J. P. (1999) "Cell adhesion molecules in the development and progression of malignant melanoma," *Cancer and Metastasis Reviews*, **18**, pp. 345–357.

[21] Ma, Y., J. Wang, S. Liang, C. Dong, and Q. Du (2010) "Application of Population Dynamics to Study Heterotypic Cell Aggregations in the Near-Wall Region of a Shear Flow," *Cellular and Molecular Bioengineering*, **3**(1), pp. 3–19.

[22] McGary, E. C., D. Chelouche Lev, and M. Bar-Eli (2002) "Cellular Adhesion pathways and Metastatic Potential of Human Melanoma," *Cancer Biology & Therapy*, **1**(5), pp. 459–465.

[23] Migliorini, C., Y. Qian, H. Chen, E. B. Brown, R. K. Jain, and L. L. Munn (2002) "Red blood cells augment leukocyte rolling in a virtual blood vessel," *Biophysical journal*, **83**(4), pp. 1834–1841.

[24] Nicolson, G. (1989) "Metastatic tumor cell interactions with endothelium, basement membrane and tissue," *Current Opinion in Cell Biology*, **1**, pp. 1009–1019.

[25] Orsello, C., D. Lauffenburger, and D. Hammer (2001) "Molecular properties in cell adhesion: a physical and engineering perspective," *Trands in Biotechnology*, **19**(8), pp. 310–316.

[26] Siegel, R., D. Naishadham, and A. Jemal (2012) "Cancer Statistics, 2012," *CA: A Cancer Journal for Clinicians*, **62**, pp. 10–29.

[27] Simon, S. I. and C. E. Green (2005) "Molecular mechanics and dynamics of leukocyte recruitment during inflammation," *Annu. Rev. Biomed. Eng.*, **7**, pp. 151–185.

[28] Slattery, M. J. and C. Dong (2003) "Neutrophils influence melanoma adhesion and migration under flow conditions," *International journal of cancer*, **106**(5), pp. 713–722.

[29] Springer, T. (1990) "Adhesion receptors of the immune system," *Nature*, **346**, pp. 425–434.

[30] Tegenfeldt, J. O., C. Prinz, H. Cao, S. Chou, W. W. Reisner, R. Riehn, Y. M. Wang, E. C. Cox, J. C. Sturm, P. Silberzan, et al. (2004) "The dynamics of genomic-length DNA molecules in 100-nm channels," *Proceedings of the National Academy of Sciences of the United States of America*, **101**(30), pp. 10979–10983.

[31] Wang, W. and M. R. King (2012) "Multiscale Modeling of Platelet Adhesion and Thrombus Growth," *Annals of Biomedical Engineering*, **40**(11), pp. 2345–2354.

[32] Zaccai, G. (2000) "How soft is a protein? A protein dynamics force constant measured by neutron scattering," *Science*, **288**(5471), pp. 1604–1607.