Implementing a Numerical Package to Model Collective Cell Migration

David Stonko[1], Michelle Starz-Gaiano[2] and Bradford E. Peercy[1]

[1] Department of Mathematics and Statistics, University of Maryland, Baltimore County
[2]Department of Biological Sciences, University of Maryland, Baltimore County

January 27, 2014

Abstract

In this article we present our three dimensional mathematical model of collective cell migration and describe the method by which we developed and implemented this package on the High Performance Computing Facility at UMBC. Specifically, we describe our motivation and the results of a new interface to define initial conditions for the three dimensional system, specify implementation constraints and solutions for the HPCF regarding post-processing, and outline several implementation changes that we made in order to achieve improved computation time.

1 Introduction

1.1 Motivation

Cell migration plays an important role in normal and pathological development [2, 7]. It is also an important factor for a proper immune response, wound healing, embryonic development and a number of other processes. In some processes individual cells migrate independently of other cells in the system. In other scenarios, groups of cells migrate together in clusters. Moreover, collective cell migration plays an important part in morphological changes during development, repair to injured tissue, and angiogenesis [4, 7]. When gone awry it can lead to cancer metastasis and other human syndromes. Despite the prevalence of cell migration throughout biology, little is known about the underlying mechanism by which cells coordinate their movements. This is especially true of collective cell migrations in which motile and non-motile cells translocate together. This

motivates the development of a model to capture the biophysical interactions in a cluster of cells that migrate together.

To investigate this mechanism we turn to a specific cell migration process within the fruit fly *Drosophila melanogaster*. Humans and *Drosophila* have a high degree of conservation in genes that have shown to be associated with disease [8]. Specifically, the fly can provide information on genetic processes involved in cell migration. Fruit flies also have a short generation time, high fecundity, tissues that can be observed live under a microscope, and are amenable to numerous genetic techniques [11]. This makes *Drosophila* especially useful for elucidating genetic mechanisms of physiological processes, which ultimately allows researchers to gain valuable insight into the workings of human systems.

1.2 Mathematical Background and Modeling Approach

Mathematically, work has been done to model the mechanism of cell migration in a number of ways and from various perspectives [4, 10, 12]. Complex individual cell models have been developed, but are too computationally expensive to track a cluster or would be difficult to implement for this system due to a lack of quantitative data. Cell automata models such as Cellular Potts Models exist. These models use energy minimization techniques and typically include growth dynamics that may not be applicable to how we intend to model this system. Other models have been developed that approach cell migration from the context of the dynamics of cell sorting [3]. However, these models were also developed to capture morphological changes at the cellular level. This perspective may not be directly applicable, and would add complexity that could be computationally expensive.

Another typical approach used to model problems like this would be a PDE continuum model. However, in our system the biologists have observed that there are a specific number of discrete border cells that behave differently from other cells in the system. This is why we ultimately decided to model the cells with a system with ODEs, where each cell has a corresponding ODE that tracks the position of it in time. Yamao developed a force-based mathematical model that is of this form [4]. This model simulates the migration of cells by computing the forces of adhesion, repulsion, migration and stochasticity in a related system of loosely connected cells. However, this construct does not capture the migration of a tightly connected cluster of cells. Also, this model was predicated on a system with a single cell type. These differences motivated the construction of a new mathematical model that tracks the position of cells as they change due to forces from adjacent cells and that provides the necessary flexibility to capture the heterogeneous cell population intrinsic to the developing egg chamber.

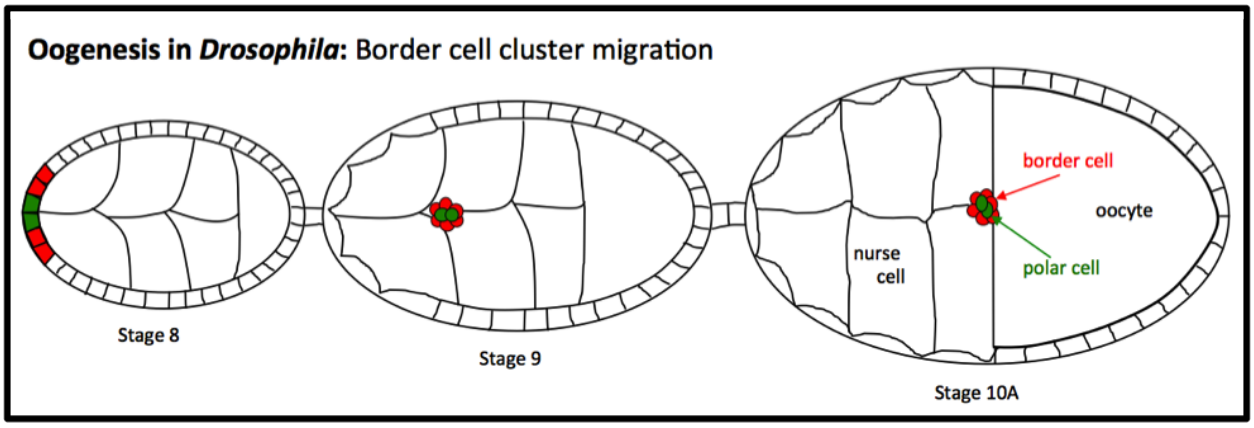

Figure 1: Oogenesis from stage 8 to early stage 10, which includes the border cell cluster migration. The egg chambers are approximately 200nm in length and this process takes approximately 4-5 hours [9]. At the start of stage 8 of Drosophila oogenesis, polar cells (green) lie in the follicular epithelium of developing egg chambers. The anterior polar cells recruit neighboring epithelial cells to differentiate into border cells (red). In stage 9 the border and polar cells form a cluster that becomes autonomous of the epithelium. These cells migrate as a collective and by stage 10 have traversed the egg chamber and arrive at the developing oocyte.

Previously we have developed a two-dimensional mathematical model of the plane through which the cluster migrates [5] based on this approach. It allowed us to make some hypotheses regarding migratory behavior, but also identified a need to scale our model into three dimensions so that we may tackle the questions that we initially set out to investigate, namely formation, rotation, and behavior of the cluster during migration. Therefore, we expanded our model and altered our mathematical framework so that we could capture these phenomena.

The expansion of our model from two to three dimensions imposed both numerical and mathematical challenges, which I outline in detail within this article. The goal of the remainder of this section is to constitute an abridged explanation of the necessary biological background and to summarize the two-dimensional mathematical model that we have previously developed.

1.3 Collective cell migration

During oogenesis in *Drosophila*, epithelial cells delaminate, form a cluster, and translocate as a collective in the direction of the oocyte, as depicted in Figure 1. Initially, these cells lie in the follicular epithelium at the anterior end of the egg chamber. Two polar cells (red), secrete a diffusible ligand that recruits nearby follicle cells to differentiate into border cells (blue). The polar cells and

border cells coalesce and migrate toward the posterior of the egg chamber, where the developing oocyte sits. This cluster then migrates between the nurse cells, which fill the space within the egg chamber. Effective migration of the cluster across the egg chamber is essential to the development of the egg. The border cells carry with them the nonmotile polar cells and move in response to a chemoattractive gradient in the egg chamber [7].

1.4 Mathematical construct

Previous work has been done to create a mathematical model of individually migrating neural crest cells [4], as previously discussed. We have developed a similar mathematical model that tracks the repulsion, adhesive and migratory forces between cells and includes a small degree of stochastic fluctuation of each Identical Math Cell, or IMC, according to Figure 2. Our formulation is different from previous models because we have applied the model to a heterogeneous cell population by assigning the appropriate properties to different types of IMCs and we aggregate numerous IMCs to capture the dynamics of the nurse cells, which are much larger than the migratory cells. We accomplish this aggregation by increasing the adhesion between the individual IMCs that make up each nurse cell. Within these IMCs, stochastic forces and reactive forces are maintained to account for the observed integrity and malleability of nurse cells. After determining the important forces of our system and developing the construct depicted in Figure 2, we were able to write down differential equations to track the position of each IMC in time. These differential equations capture how the forces cause the position of the cells, $U = [x, y]^T$ in two dimensions or $U = [x, y, z]^T$ in three dimensions, to change in time as the forces acting upon each cell changes. We compute these differential equations with a forward Euler method. These equations and functional definitions define how we model the system, and follows from the construct outlined in Figure 2. These equations are shown in Figure 3.

These equations and definitions constitute our two-dimensional mathematical model. The last three definitions describe how forces are defined based on the cell types that are interacting. Specifically, Equation 1 represents the change in cell position (cell position is U), in time, t. The change in cell position, U_i, is the sum of all forces on a particular cell, i. This is computed by summing the adhesive forces, $F^a_{i,j}$, and repulsive forces, $F^r_{i,j}$, between cell i and all of cell i's neighbors (which are the j's), and by accounting for stochastic fluctuation, F^s_i. Additionally, if cell i, or cell j is a migratory cell, then there will also be a migratory force, $F^m_{i,j}$, that is taken into account. The migratory force is 0 if the migrating cell is not a border cell ($i \notin$ B: set of border cells) or the cell it is interacting with is not a nurse cell ($j \notin$ N$_k$: the list of nurse cells). Moreover, the strength of

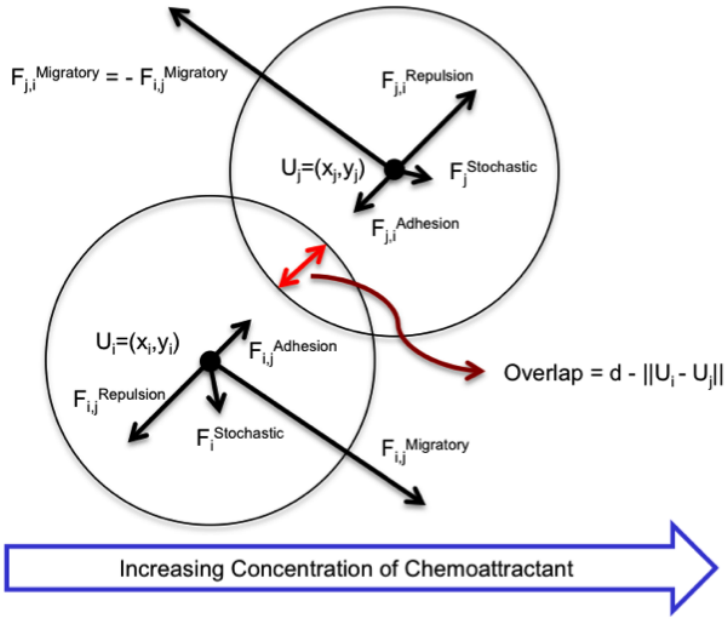

Figure 2: The forces between two adjacent IMCs. The repulsive force pushes the cells apart with equal magnitude and opposite direction. The adhesive force acts to attract IMCs. The migratory force directs migratory cells up the gradient forcing adjacent cells in the opposite direction. The stochastic force randomly perturbs the position of each IMC and is not dependent on position. These dynamics result in overlap of the domains of neighboring IMCs corresponding to the diameter of an IMC minus the 2-norm of the x, y positions of each IMC.

$$\mu \frac{dU_i}{dt} = \sum_{j \in A_j} (F^a_{i,j} + F^r_{i,j}) + \sum_{j \in M_j} F^m_{i,j} + F^s_i, \tag{1}$$

$$F^a_{i,j} = C^a_{i,j} \rho^\epsilon_{i,j} H(\rho^\epsilon_{i,j})(d_{i,j}), \tag{2}$$

$$F^r_{i,j} = C^a_{i,j} \rho^0_{i,j} H(\rho^0_{i,j})(-d_{i,j}), \tag{3}$$

$$F^m_{i,j} = C^m \sigma_i \left(d^\perp_{i,j} \frac{\nabla f_{chemo}}{||\nabla f_{chemo}||} \right), \tag{4}$$

$$d_{i,j} = \frac{(U_i - U_j)}{||U_i - U_j||}, \qquad d_{i,j} \cdot d^\perp_{i,j} = 0, \tag{5}$$

$$\rho^\epsilon_{i,j} = D(1+\epsilon) - ||U_i - U_j||, \tag{6}$$

$$F^s_i = C^s \zeta(i). \tag{7}$$

$$C^a_{i,j} = \begin{cases} \beta^a & i,j \text{ different cell type} \\ \beta^a M^a_B & i \text{ or } j \in B \\ \beta^a M^a_P & i,j \in P \\ \beta^a M^a_{P,B} & i \text{ or } j \in P \text{ and } i \text{ or } j \in B \\ \beta^a M^a_E & i,j \in E \\ \beta^a M^a_N & i \text{ or } j \in N_1 \text{ or } N_2 \text{ or ... or } N_k \end{cases} \tag{8}$$

$$C^r_{i,j} = \begin{cases} \beta^r & i,j \text{ different cell type} \\ \beta^r M^r_{B,N_k} & i \text{ or } j \in B \text{ and } i \text{ or } j \in N_k \\ \beta^r M^r_P & i \text{ or } j \in P \end{cases} \tag{9}$$

$$C^m_{i,j} = \begin{cases} 1 & i \in B, j \in N_k \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Figure 3: Equations used to compute the forces and the resulting displacements of the IMCs. Equation (1) is an ordinary differential equation that captures the change in the position of cell i due to the forces applied to cell i. These forces depend on the position of the cell, and the surrounding cells. U_i is the position of cell i, and $C^a_{i,j}, C^r_{i,j}, C^m_{i,j}$ are linear scalars that depend on IMC type (described below the equations), and $\zeta(i)$ is a stochastic force generator that has x- and y-components (also z components in the three dimensional model) taken from a Gaussian distribution with mean zero and unit standard deviation. f_{chemo} is the distribution of chemoattractants in space and affects the magnitude and direction of the migratory force. A_i is the set of all IMCs in the domain. σ_i determines the sign of the migratory force and is $+1$ for migratory IMCs and is -1 for non-migratory IMCs. The vector $d_{i,j}$ is the unit vector from cell j in the direction of cell i. D is the diameter of an IMC. The value $\rho^\epsilon_{i,j}$ is the overlap of domains of cells i and j for the repulsive force and extended for the adhesive force.

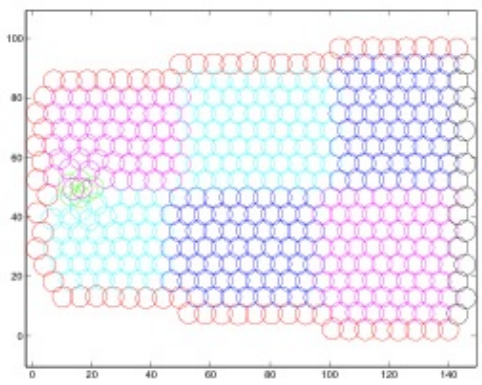

Figure 4: The initial conditions of the system, with manually defined IMC type and position. The cluster is located on the anterior end of the chamber, inside of the epithelium.

adhesion,C_a and strength of repulsion,C_r, is dictated by which type of cells are interacting (e.g. two polar cells adhere more strongly to one another than a polar cell and a nurse cell). The adhesion and repulsive forces are also governed by the $\rho^{\epsilon}_{i,j}H(\rho^{\epsilon}_{i,j})$ term. Here ϵ represents the distance beyond the edge of the cell that the force can act. For example, the repulsion force is always such that $\epsilon = 0$, because the force will not act when the cells are not in contact (i.e. the distance between the surface of the cells is more than 0). $\rho_{i,j}$ is the overlap of the domains of cells i and j. The direction of these forces comes from $d_{i,j}$, which is the unit vector from cell i in the direction of cell j.

We non-dimensionalized this system for use in our computational model, so that we could better understand the components of the system and how changing any one component influences other aspects of the system [5]. Here we defined the initial position and type of each IMC manually. This was possible because of our limited number of IMCs. The end result was a two-dimensional egg chamber that captured some of the dynamics of the system, as seen in Figure 4. However, the two-dimensional model did not capture the rotation of the border cell cluster as it migrated. Our desire to investigate this phenomenon motivated our construction of the three-dimensional model. Moreover, when we entered three dimensions it was no longer feasible to manually define the initial conditions for each IMC in the code directly, which motivated us to develop a new way to specify initial conditions, the result of which is outlined in Section 2.

1.5 Implementation Overview

The file package that we have developed to execute this computation is divided into three subpackages. The first subpackage is a GUI which is used to define the initial conditions of the cells in

the system (which is detailed in the next section). This package outputs a data file of the specified initial conditions. The second subpackage is where the model is implemented and computed. It outputs a binary file that saves each IMC's position in space at each time point that the model is computed, and is where we hope to achieve the numerical speed up. The third part of the package goes back over this binary file and plots all of the IMCs at each time point and then saves these plots as image files, which can be concatenated into a video. This process required a new approach when implemented on the HPCF, as described in Section 4. An overview of the file package is shown in Figure 5.

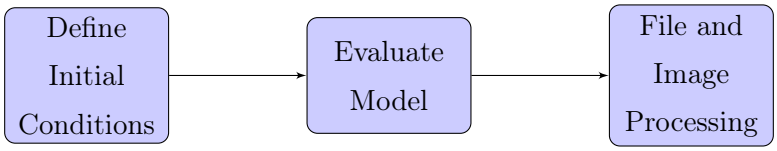

Figure 5: Package overview. The first subpackage has been used to create a library of initial conditions which a user can direct the second subpackage to use. The second part of the package uses the given initial conditions to solve the set of coupled ODEs that define the cells' movements and save the positions. The third subpackage accepts the data provided by the second and produces images of the domain as it evolves.

2 Objective: Specification of Initial IMC Positions in 3D

Our mathematical model tracks the position of each Identical Math Cell, or IMC, in time. However, in order for this calculation to occur we must first have well defined initial conditions. In two dimensions, we were able to specify the initial cell position and type by looping over the cells in the plane and manually setting these parameters. However, in three dimensions, we have many more cells, so hardcoding initial conditions was no longer sensible. Thus, our first challenge of the transition from two to three dimensions was to find a way to determine the initial conditions of all of the IMCs.

To solve this problem we developed a graphical user interface to interact with Matlab. This GUI takes directed input from the user and outputs the desired initial conditions into a file, which is then uploaded to our computational engine to undergo calculation. This methodology also enabled us to maintain the necessary flexibility so that we can perform all projected computational experiments.

Matlab offers a function, **ginput()**, that enables user interaction by allowing a user to click on a

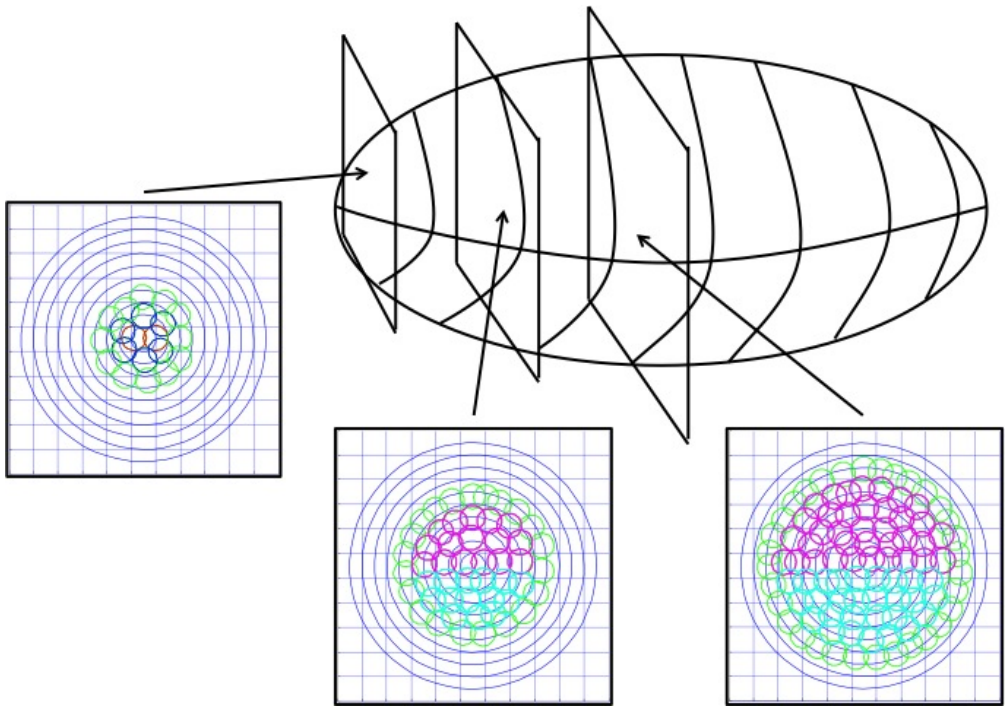

Figure 6: Graphical representation of how we utilize our GUI to define the three dimensional positions of the IMCs by placing the cells directly into two dimensional slices along the third dimension.

figure and then save the x and y position of each click in vectors. Our goal was to use this function to place IMCs by allowing a user to click on the position of all the cells in each 'z-slice', save those initial conditions, and then step through the remaining slices until all of the IMCs have been placed. Figure 6 shows the three dimensional representation of this process for three the slices of the egg chamber.

However, a call to **ginput()** forces Matlab to stay in the function until the user reached a breakpoint, or presses the return key to leave. This was problematic because we wanted to plot each IMC as the user clicked on its location so that the user could visualize where they had placed the IMC. To address this issue we obtained source code for **ginput()** and rewrote part of it so that it would be useful to us. Part of my revision included adding a call to **circplot(x,y,r)**, which plots a circle around a point centered at x, y with radius r after each click, which allows the user to see where the circle actually sits in the plane. We then integrated this code into a cell specifier function, which is used to define the positions of the IMCs when called by a higher level function, which organizes the output and operates the GUI. This function can create a set of initial conditions that can be manually loaded into the script that computes the model, or it can be called by this

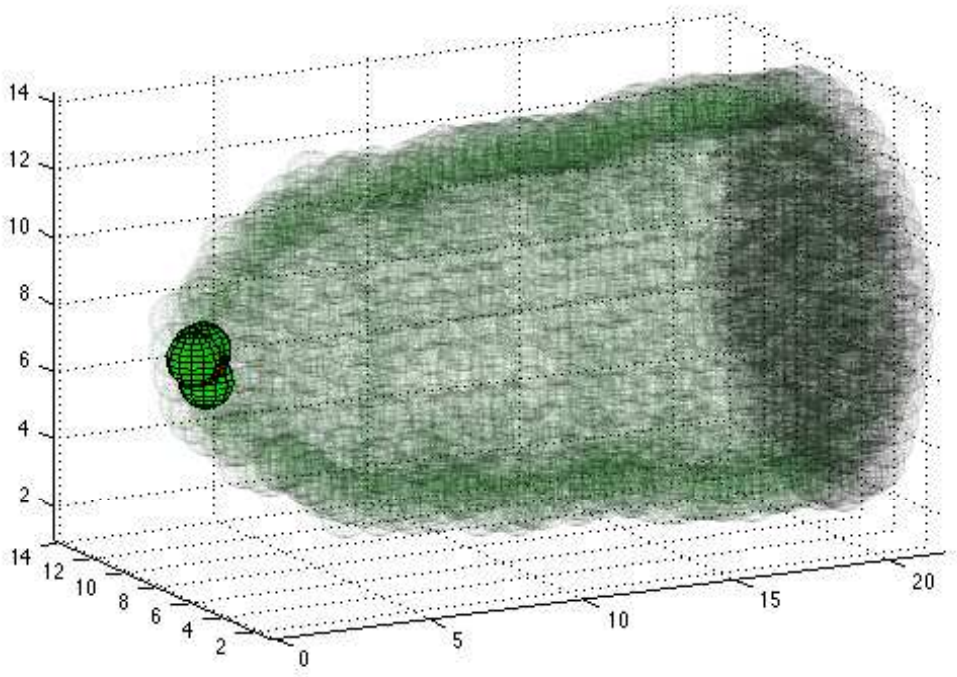

Figure 7: The final result after fully executing first subpackage. A semi-lateral view of the computational egg chamber, with the epithelial cells plotted in transparent green and the cluster of six border cells plotted in opaque green. The polar cells are plotted as well, but are located at the center of the cluster, mostly out of view.

script to create new initial conditions. Specifically, the call: 'ThreeD-CPM-gen(10,15,15,1)' will ask the user to create new ICs with the GUI of size 10 (depth) by 15 (length) by 15 (width), but the call 'ThreeD-CPM-gen(10,15,15,'default.mat')' uses previously created ICs that are saved in a file called 'default.mat' in the same folder. After fully implementing this code and executing the subpackage we obtain initial conditions like those plotted in Figure 7 and a **.mat** file with the coordinates of each IMC in the system.

3 Objective: Numerical Challenges and Implementation

Once we developed and fully implemented the software package we recognized that there was bottleneck in runtime during the middle step, where the simulations are being generated. Indeed, the implementation used to build initial conditions had undergone a complete revision when scaled from the two dimensional model to the three dimensional model. During this revision runtime was able to

	Method 1	Method 2
Runtime (H:M:S)	18:18:58	09:21:27
Runtime (normalized %)	100.0	51.09

Table 1: Runtime for our computational package. These run times correspond to a computational timespan from $t = 0$ to $t = 1$ in the forward Euler step and all associated overhead with the function.

be virtually eliminated (the runtime is faster than a user can click, and is therefore not a problem). Additionally, we rarely utilized this part of the package once a library of initial conditions were created and stored in data files. On the other end we find that the post-processing is in fact very slow. However, high quality post processing is only performed intermittently. Also, due to the constraints placed on the code by the system, which have been previously discussed[1], there is probably little to be gained by devoting a large amount of time to speeding up this process. However, we found that there was was both a need and the capability to speed up the part of the code that performs the simulation.

In order to gauge the improvement in runtime of successive modifications in the code we have made each modification and measured the run time for the overhead and computation from $t = 0$ to $t = 1$ with a time step of 0.004. Table 1 shows the improvement in run time as successive modifications have been made in both seconds and as a percentage relative to the initial runtime in Method 1. Method 1 was the code before the if/else statements were optimized and Method 2 was after. Specifically, there are if/elseif statements that alter the forces between certain IMCs, which we manipulate so that they are biologically accurate. To improve this aspect of the code we have partially vectorized the calls to **ismember(X,Y)**, which returns a logical if X is a member of Y. In fact, counterintuitively, this change actually *increased* the number of if-statements, but *decreased* the total runtime because it got responses from the statements faster. We also made some improvements to the structure of the code that calculates migratory force, but this is not considered to be a major factor because this portion of code is infrequently entered.

Ultimately, we saw some improvement of the runtime, as shown in Table 1. However, the nature of the ODEs forces the time step to remain small. No matter how much we improve the runtime of a single time step we remain limited by the sheer number of time steps required to complete a simulation. For this reason, I believe that this improvement in runtime is indeed significant. Though,

[1] Another constraint on the post-processing was that I desired to have a method which worked on the cluster and on my local machine, where I am running Matlab R2011a on a 64bit Mac Intel). This led me to shy away from options not available before 2012 and limited my rendering options.

an even greater improvement may be achieved if the stiffness of the functions were fully characterized and cleverly subverted. By this, we mean that each ODE could be individually optimized to take larger time steps during intervals in which it is experiencing little change. However, since all of the ODEs are coupled, they need to step together in time. Then, because of the large number of ODEs it is likely that at least one is changing rapidly at any time during the simulation, which in turn drives the need for a global small time step.

4 Objective: Implementing Image Processing on Tara

Figure 5 depicts an overview of the structure of the software package. During post-processing we loop over the binary file produced by the previous section of code. During each iteration we plot the cells and use Matlab's function **getframe()** to save the frame to the end of an **avi** file using one of various methods, namely **avifile()** or **videowriter()**. However, this method (specifically **getframe()**, which each of these functions rely on) requires a figure to be rendered onto the screen in order to operate the undocumented Matlab function **hardcopy()**, which is what Matlab uses to actually obtain figure properties from figures. Because tara operates Matlab without a display or desktop, the method that we had been using would not work so we needed to find a new way to process our simulation.

Previously, users on tara have created each figure and exported the figures as **.png**, **.jpg**, or **.tif** files (unpublished). However, when Matlab exports figures on tara it is unable to extract the transparency data (which is imperative to have for proper interpretation of the results in the project, as shown in Figure 8) known as 'AlphaData' when it saves the figure in these file types because a figure has not been rendered. When searching for a solution I found that many people have encountered this problem with Matlab, and most used the aforementioned **getframe()** function to solve their problem [2]. However, as discussed, this method would not work on tara.

Instead, I decided to use a function **plot2svg()** which takes a figure and exports to a Scalar Vector Graphics, or **svg**, file. This filetype does not rely on the undocumented **hardcopy()** function to get the transparency data and instead saves the data from each patch independently, so it works with tara[3]. Once the figure is exported to an **svg**, I use system commands on tara to convert the **svg** file into a **png**, which I can later turn into a video. Previously, several tara users have used a bash script to turn an image sequence into a movie. However, I have found that using an application

[2]Other workarounds include changing the renderer, but tara does not have the OpenCL option.

[3]This function is available on the Mathworks file exchange:

`http://www.mathworks.com/matlabcentral/fileexchange/7401-scalable-vector-graphics-svg-export-of-figures`

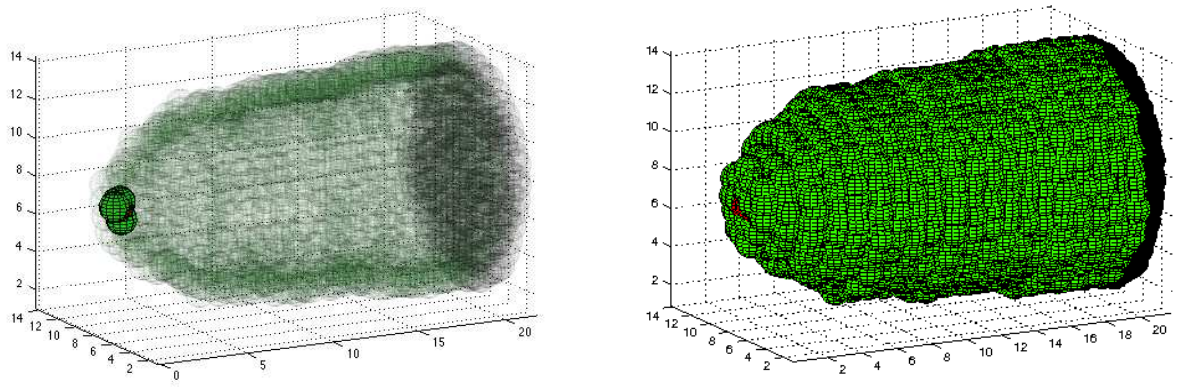

Figure 8: (A) Domain with appropriate transparency values. One may interpret the dynamics that are occuring as the cluster makes its way through the center of the chamber. (B) An example of a figure produced by tara, without appropriate transparency values. As the cluster moves into the chamber it will no longer be visible.

like QuickTime Pro 7 or an open source application like Blender works well and allows for video editing and scaling; available: `http://www.blender.org`. With **plot2svg()** in hand, a user on tara can implement the following code to obtain an image sequence of **png** or **jpg** files with appropriate transparency. The first block of code goes before the loop that creates each successive image to create a directory and enable certain settings, and the second block of code goes inside of the loop to save and process each image.

```
% this code sets up the directory and changes several settings, and
% goes before the start of the loop that will plot each image.
c = clock; % obtains time to create a directory with current time and date
directory = strcat('movie','-',date,'-',num2str(c(4)),'-',num2str(c(5)));
mkdir(directory)
fig = figure;
set(gcf,'visible','off');
set(gcf,'Renderer','Zbuffer') % tara does not have all available renderers.

% this code goes at the end of the for loop that produces each image,
% where it will save each file as an svg, convert this svg to a png, and
% then remove the old svg file.
filename = sprintf('/img%d.svg', frame); % creates next file in image sequence
whereto = [directory filename];
```

```
plot2svg(whereto);
whereto2 = [whereto(1:end-3) 'png'];
system(['convert ' whereto ' ' whereto2]); % tells system to convert file type
system(['rm ' whereto]); %removes old file, which has large size
```

This code will save the **svg** file with the corresponding filename during each iteration. Then it will call tara to convert the **svg** file to a **png** file with the same file identifier, and remove the initial **svg** file. After the entire loop has executed the result will be a folder of an image sequence for the simulation. This folder will contain frames that can be converted into a movie. Two such frames of a run are shown in Figure 9. The result is comparable to that of the package before changes were made, except that these additional steps, namely the call to **plot2svg()** and system calls to tara, increase the total processing time. However, this appears to be the best viable option. Additionally, this image sequence can easily be converted by an encoder or with the previously discussed software. If QuickTime Pro 7 is available, simply click 'File', 'Open Image Sequence' and save the file in the preferred file type.

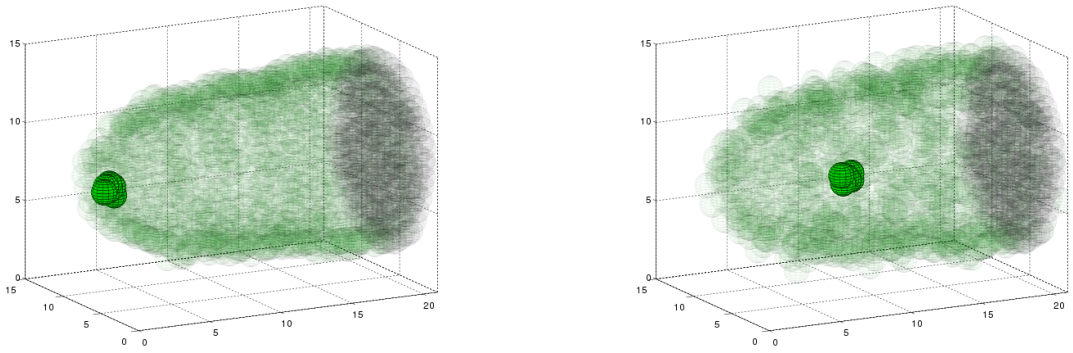

Figure 9: Two frames from an image sequence of a run with six border cells, shown in green, and two polar cells, which are shown in red, but not visible because they are the center of the migrating cluster. This is representative of the final product of the numerical package. These are frames 64 and 539, which correspond to 0.1 and 1.1 hours, respectively.

5 Conclusions

In this article we have described our three dimensional mathematical model of collective cell migration in *Drosophila melanogaster* and described the method by which we developed and implemented this package on the High Performance Computing Facility at UMBC.

We have explained how we created a new interface to define initial conditions for the three dimensional system, which may be useful to others who model cell migration or to those developing a GUI for any purpose that requires feedback beyond what the normal capabilities of **ginput()** are. We have also specified implementation constraints and solutions specific to the High Performance Computing Facility at UMBC that may be helpful to others in need of a method to output image files with transparency data on tara, or on any cluster with the same rendering constraints. Lastly, we have outlined several modifications to the computational package that have successfully decreased the computational runtime to under 50% of what we began with.

Thus, in the course of this project we have created a fully functional, system specific, implementation of our mathematical model that can be used to simulate a number modeling scenarios all while decreasing the computational expense and improving the end result.

References

[1] Dafni Hadjieconomou, Shay Rotkopf, Cyrille Alexandre, Donald M. Bell, Barry J. Dickson, and Iris Salecker, *Flybow: genetic multicolor cell labeling for neural circuit analysis in Drosophila melanogaster*, Nature Methods **8** (2011), 260–266, DOI 10.1038/nmeth.1567.

[2] Michelle Starz-Gaiano, Mariana Melani, Xiaobo Wang, Hans Meinhardt, and Denise J. Montell, *Feedback Inhibition of JAK/STAT Signaling by Apontic Is Required to Limit an Invasive Cell Population*, Developmental Cell **14** (2008), no. 5, 726 - 738, DOI 10.1016/j.devcel.2008.03.005.

[3] Takuya T. Maeda, Itsuki Ajoki, and Kazunori Nakajima, *Computational cell model based on autonomous cell movement regulated by cell-cell signalling successfully recapitulates the "inside and outside" pattern of cell sorting*, BMC Systems Biology **1** (2007), no. 43, DOI 10.1186/1752-0509-1-43.

[4] Masataka Yamao, Honda Naoki, and Shin Ishii, *Multi-Cellular Logistics of Collective Cell Migration*, PLoS ONE **6** (2011), no. 12, DOI 10.1371/journal.pone.0027950.

[5] David Stonko, *Force-Based Biophysical Model of Border Cell Migration: Unraveling the mechanism of collective cell migration*, Senior Thesis (2013), available at `http://www.math.umbc.edu/~dstonko1/THESIS_FINAL_STONKO.pdf`.

[6] Michelle Starz-Gaiano, Mariana Melani, Hans Meinhardt, and Denise J. Montell, *Interpretation of the UPD/JAK/STAT morphogen gradient in Drosophila follicle cells*, Cell Cycle **8** (2009), 2918 - 2926, DOI 10.4161/cc.8.18.9547.

[7] Denise J. Montell, Wan H. Yoon, and Michelle Starz-Gaiano, *Group choreography: mechanisms orchestrating the collective movement of border cells*, Nature **13** (2012), no. 10, 631-645, DOI 10.1038/nrm3433.

[8] Lawrence T. Reiter, Lorraine Potocki, Sam Chien, Michael Gribskov, and Ethan Bier, *A Systematic Analysis of Human Disease-Associated Gene Sequences In Drosophila melanogaster*, Genome Research **11** (2001), no. 6, 1114-1125, DOI 10.1101/gr.169101, available at `http://genome.cshlp.org/content/11/6/1114.full.pdf+html`.

[9] Mohit Prasad and Denise J. Montell, *Cellular and Molecular Mechanisms of Border Cell Migration Analyzed Using Time-Lapse Live-Cell Imaging*, Developmental Cell **12** (2007), no. 6, 997 - 1005, DOI 10.1016/j.devcel.2007.03.021.

[10] Robin N. Thompson, Christian A. Yates, and Ruth E. Baker, *Modeling Cell Migration and Adhesion During Development*, Bulletin of Mathematical Biology **74** (2012), no. 12, 2793-2809, DOI 10.1007/s11538-012-9779-0.

[11] Anna C-C Jang, Michelle Starz-Gaiano M., and Denise Montell J., *Modeling migration and metastasis in Drosophila.*, J Mammary Gland Biol Neoplasia **103** (2007), no. 14.

[12] R. Rangarajan and M. H. Zaman, *Modeling cell migration in 3D: Status and challenges*, Cell Adhesion & Migration **2** (2008), no. 2, 106–109.

Acknowledgements

I would like to thank Dr. Matthias K. Gobbert for his instruction on this project, part of which constituted a final project for Math 620 - Numerical Analysis. I would also like to thank UMBC Graduate Student Zana Coulibaly for his guidance with regards to post-processing on the HPCF and his support with running scripts on tara.

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258 and CNS–1228778) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `http://www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.