

THE OHIO STATE UNIVERSITY

GrowFlesh:

The Lightweight, Customizable, Epithelial Tissue Simulator

Author:

Melvyn Ian DRAG
B.A. Mathematics

Advisors:

Dr. Marcos Manuel SOTOMAYOR
Dr. Edward OVERMAN

Presented in partial fulfillment of the requirements for the degree

Master of Mathematical Sciences

in the Graduate School of The Ohio State University

April 5, 2015

Copyright by

Melvyn Ian Drag

2015

Abstract

Epithelial tissue is introduced, current models of epithelial development are described, and then a new implementation of a long existing model is presented. The code for the model is thoroughly examined and the algorithms and data structures underlying it are presented. Some topological insights are made about the nature of epithelial tissue, and some discussion of which numerical methods are used is made.

a mis escuincles Alfie and Andy

Acknowledgements

I would like to thank Dr. Marcos Sotomayor, and the entire Sotomayor lab (I won't mention names for fear of leaving someone out) for having me on their team. Dr. Sotomayor gave me a very fancy computer on which to develop my software, and offered great suggestions about how to improve my scientific writing and presentations. The other lab members were very welcoming to me, and made me feel at home in an entirely foreign environment! On my free time I was able to do a couple of interesting lab experiments, and I learned all about how hearing works. In addition, Dr. Sotomayor graciously funded part of my trip to a programming conference in Texas, and offered me many more opportunities for academic and professional growth which I was simply not able to accept! Thank you all for the good times, the resources, and the opportunities.

I would like to thank Dr. Overman for his letters of recommendation when I needed them, his patience with me in what turned out to be the one of the hardest and most rewarding classes I've ever taken, and especi

Vita

Contents

1	Why Study Epithelial Tissue?	4
2	Overview of the Model	7
2.1	Introduction	7
2.2	The Nagai-Honda Model	8
2.2.1	How the Vertices Move	8
2.2.2	Academic Dishonesty, or Obvious Modeling?	12
2.2.3	Topological Changes to the Mesh	12
2.2.4	Quality, Not Quantity	14
3	Further Remarks About Epithelial Tissue and the Honda-Nagai Model	17
3.1	The Euler Characteristic and Its Implications	17
3.2	Why the Specified Topology Is Not Perfect	18
3.3	Feedback Mechanisms and Proliferation	19
3.4	Additional Curiosities	20
4	GrowFlesh	21
4.1	About Epithelium	21
4.2	Sample Configuration Files	22
4.2.1	config.txt	22
4.2.2	parameters.hpp	23
4.2.3	changemesh.txt	23
4.3	Image Gallery	23
4.4	Why <i>Epithelium</i> is a good piece of software	24
5	The Design of <i>GrowFlesh</i>	34
5.1	[Highly Simplified] Pseudocode	34
5.2	Classes	35
5.2.1	The Cell Class	35

5.2.2	The Coordinate Class	37
5.3	Initial Mesh Design	37
5.4	A Relational Database	39
5.5	Moving the Vertices	40
5.6	Embarassing Parallelism and CUDA	41
5.7	Computing Topological changes.	42
5.7.1	The T1 Swap	42
5.7.2	The implementation of the T2 swap.	44
5.8	Error Tolerance of the Algorithm	44
A	Valuable Programming Lessons Learned	49
A.1	GitHub	49
B	How Several Programming Languages Were Used In Harmony	51

List of Figures

1.1	The Types of Epithelial Tissue	4
1.2	Some existing models of epithelial tissue.	6
2.1	The Giant's Causeway	9
2.2	Potential Energy as a Function of Distance from Equilibrium.	10
2.3	A T1 Swap	13
2.4	A T2 Swap	14
2.5	The Distribution of Cell Shapes As a Function of Time [21]	16
3.1	Square Cells.	19
4.1	Beta Dominant Output	26
4.2	Equal Parameter Output	27
4.3	Gamma Dominant Output	28
4.4	Lambda Dominant Output	29
4.5	Missing Parameter Output	30
4.6	Large Area Output	31
4.7	A Selection of Interesting Tissues	32
4.8	A screenshot of the simulator	33
5.1	A 5x5 Hexagonal Mesh.	38
5.2	The Relational Database	39
5.3	Getting Cells in Order	41
5.4	A Rotated Mesh for Error Analysis.	45

Chapter 1

Why Study Epithelial Tissue?

Epithelial tissue covers the interior and exterior surfaces of our bodies. Skin, the lining of the esophagus and intestines, the urethra, the lining of the lungs and all of the bronchioles in the lungs are all made up of epithelial tissue. In this way, we can think of epithelial tissue as being the envelope in which our contents are packaged [15]; epithelial tissue is our interface with the outside world. This definition of course extends to all animals and even to plant life, as plants have a single layered epithelial tissue that covers their exposed leaves, flowers, roots, and stems.

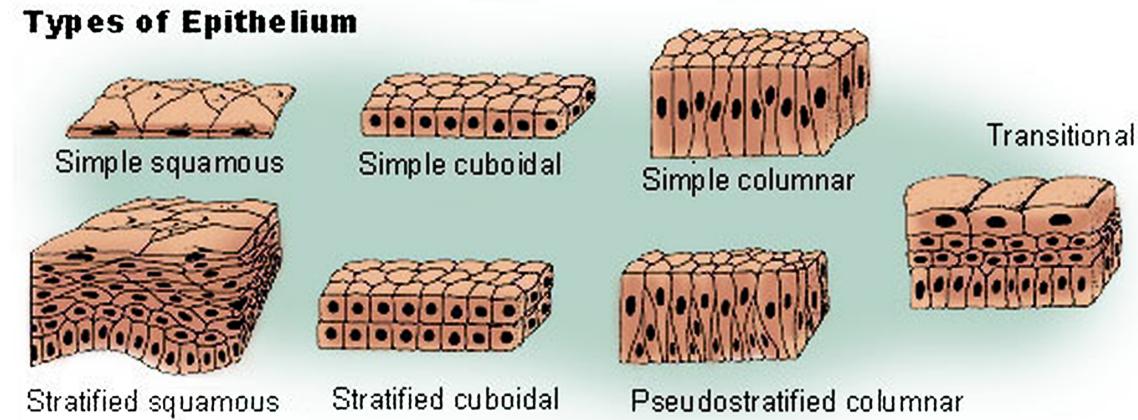


Figure 1.1: The Types of Epithelial Tissue

Botanical epithelia are rather simple and homogeneous - a single layer of cells covering exposed surfaces. As Figure 1.1 shows, however, there are many types of epithelial tissue in animals which vary in the number of layers they include, how the cells are shaped, and how tall the cells are. Each of these types of cells are found in a different region of the body where they perform a specific function. For example, the simple squamous epithelium is no more than one layer of cells thick, and the cells are all very flat, much flatter than they are wide. These cells are therefore well suited to allow diffusion across themselves. As such,

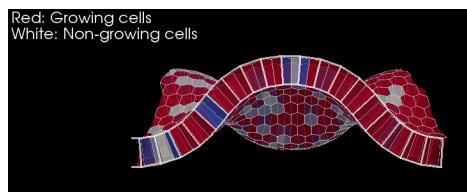
simple squamous tissue is found in the walls of blood vessels, and in the alveoli in the lungs, where the diffusion of oxygen occurs. On the other hand, columnar cells are much taller than they are wide, and are thus well suited to absorption. These cells are found in the intestines where they absorb nutrients from passing food. Stratified squamous epithelia are several layers thick line the esophagus and mouth and serve to protect against abrasion.

What all of these tissues have in common, however, is how amenable they are to computational modeling. The simplest case is that of simple epithelia, which typically have near-uniform height, and very little difference in appearance between their apical and basal faces. This means that the cells can easily be approximated by two dimensional meshes, since the top and bottom of the cells move in tandem and the surface where two cells touch can be approximated by a line. Slightly more difficult is the modeling of stratified tissue. In this case, the tissue develops in three dimensions, since underlying cells affect the cells on top of them. These cannot be modeled in two dimensions, but can be modeled by a solid composed of three dimensional polytopes. The theory behind these models is well developed, but the technical details of implementing such a model has made it so that very exist, and are often quite limited ¹.

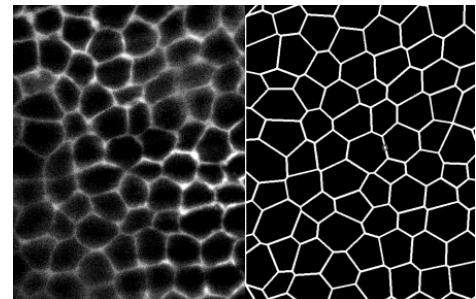
To give you an idea of what current epithelial tissue simulators have produced, I am putting some images from recent papers.

So, the question still remains: “Why Study Epithelial Tissue?”. The study of biological tissues has intrinsic value, much as mapping constellations or studying the properties of natural numbers. Beyond its inherent worth, current research in epithelial tissue is producing great results in the field of epithelial tissue morphogenesis and diversification as well as wound healing. The Honda-Nagai model which we will discuss in great detail in this paper successfully reproduced the dynamics of the healing of wounds to cats’ corneas[20]. This model has also been able to reproduce all of the essential dynamics of epithelial tissue [21]. So, in conclusion, we *should* be modeling epithelial tissue computationally because it is easy to do, effective, and the community is currently very active and passionate about the field.

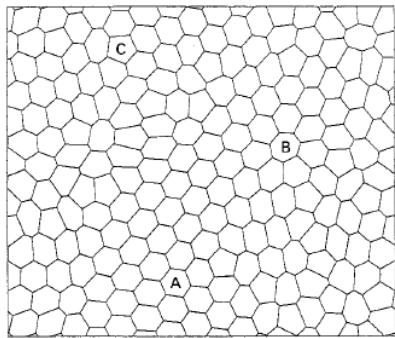
¹Even a leading epithelial tissue simulator, Chaste, still does not have stable 3D modeling capabilities



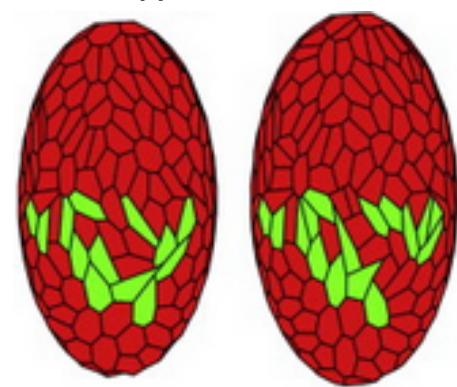
(a) Simple Squamous Tissue Bending in 3D[25]



(b) Comparison of Living Tissue and Simulation[1]



(c) Equilibrium Mesh[21]



(d) Equilibrium Mesh on a 3d Surface [8]

Figure 1.2: Some existing models of epithelial tissue.

Chapter 2

Overview of the Model

*The world was so recent that many things still lacked names,
and to mention them one had to point with a finger.*

- Gabriel Garcia Marquez

2.1 Introduction

A two dimensional **vertex dynamics model** of epithelial tissue is made up of vertices and edges. A cell is represented as a convex hull made up of cells and vertices. This model presupposes that the movement of cells in epithelial tissue can be approximated by the movement of the vertices which make up the cell. Some force is hypothesized to be the guiding force between epithelial cell movement, and this force is applied to all of the vertices in the mesh of cells, producing some final state of the tissue. The three dimensional model is an extension of the two-dimensional model, but the mesh comes to include not only vertices and edges, but also cell faces.

Epithelial vertex dynamics has been a lively field of research since the 1970s because of several heartening results in the field. Some researchers have had success modelling the morphogenesis of *Drosophila* wing growth, whereas other researchers have successfully reproduced the dynamics of corneal wound healing. Unfortunately, these results have not come from one standard force, but from a variety of different hypothesized forces.

As an example of the different approaches to force modelling, let us consider the Weliky-Oster and Honda-Nagai forces. The Weliky-Oster model specifies explicit forces which act upon the vertices due to a tension associated with boundary lengths. In two dimensions this contributes two tensions from each edge in a cell surrounding a vertex. The model also includes a cortical pressure due to the osmotic pressure inside a cell, which tends to push a vertex away from the interior. The Honda-Nagai Model (which I have recreated) takes a

much different approach and instead specifies a potential energy function for the tissue. In this model the cells move passively as they attempt to minimize the potential energy of the tissue[8].

While both of these models successfully reproduce the topological and geometric properties of epithelial tissue, I have chosen to focus my efforts on the Nagai-Honda model. So, let us look at the model in some depth.

2.2 The Nagai-Honda Model

2.2.1 How the Vertices Move

A very basic result from physics is the relationship between force and potential energy, and this is one of the building blocks of the Nagai-Honda Model. Given a force vector \vec{F} and scalars U , for potential energy and W for work, we can derive the relation:

$$\vec{F} = (F_x, F_y, F_z) \quad (2.1)$$

$$W = -\Delta U(\vec{x}) = \int_{x_0}^x F_x dx + \int_{y_0}^y F_y dy + \int_{z_0}^z F_z dz \quad (2.2)$$

$$\nabla(-\Delta U(\vec{x})) = \nabla \left(\int_{x_0}^x F_x dx + \int_{y_0}^y F_y dy + \int_{z_0}^z F_z dz \right) \quad (2.3)$$

$$-\nabla U(\vec{x}) = \vec{F} \quad (2.4)$$

The second building block of the Nagai-Honda Model is the equation of motion. In 1989, K. Kawasaki showed that the dynamics of grain growth can be reduced to a first order system given by:

$$\eta \frac{dr_i}{dt} = F_i \quad (2.5)$$

where F_i denotes the force applied to vertex i and the left hand side is the velocity of the vertex multiplied by a positive drag coefficient, η [16].

A very important paper in the field of natural cellular structures, of which the study of epithelial tissue is a small subset, is *Soap, Cells, and Statistics*. In this paper, leading researchers in the field argue that there must be some natural mechanism underlying the development of epithelial tissue, columnar basalt formations, soap froths, grin growths, and other cellular structures, as they exhibit a great deal of similarity. For example, consider



Figure 2.1: The Giant’s Causeway

the images of epithelial tissue presented throughout this paper next to the image of The Giant’s Causeway in Northern Ireland. Some differences include the exact distribution of cell shapes, the presence of chemicals in biological tissues versus the absence of growth inducing chemicals in geological structures, and the active migration of biological cells versus the entirely passive movement of soap froths; still, the equilibrium patterns of these diverse structures leads one to believe that there must be some underlying principle. The Honda-Nagai model builds upon this idea and borrows ideas from the field of grain growth.

Putting these building blocks together, we have a force acting on the vertices in the mesh, and an equation of motion which explains how the vertices move. Now all that is needed is a free energy function from which to derive our force (and this force should be similar to the force found in other cellular structures). The free energy function which provides the applied force was taken in part from the grain growth model, which asserts that grains want to achieve a target volume and perimeter via some still unknown mechanism. As previously mentioned, biological cells have idiosyncratic mechanisms working on them as well. As such, the model was extended to include a purely biological factor, *differential adhesion*. It is well known that cells have a proclivity to attach more tightly to certain cells than to others due to the presence of certain cadherin proteins in the membranes of the cells. This tendency is called differential adhesion.

The first two energy terms assume that the cell is elastic, and that the cell wants to return to a target shape. Therefore, the first two energy terms are of the elastic potential

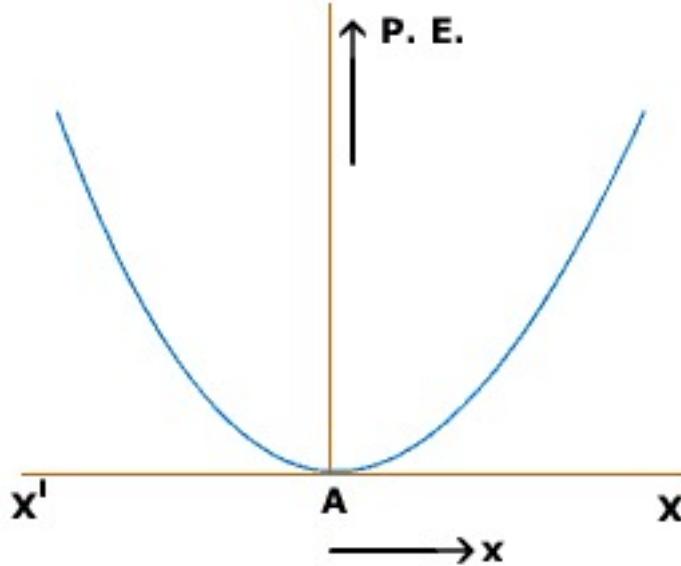


Figure 2.2: Potential Energy as a Function of Distance from Equilibrium.

form:

$$C(x - x_0)^2 \quad (2.6)$$

where x is some physical quantity, and C is some constant. The plot of this energy is therefore a parabola with a minimum at $x = x_0$ (A replaces x_0 in Figure 2.2) and the farther x is from x_0 , the more potential energy there will be in the cell. The last energy term is an adhesion energy, which is proportional to the amount of interfacial surface area between a cell and its neighbor. Here are the three energy terms:

1. The deformation energy term

$$U_D = \lambda(A - A_0)^2 \quad (2.7)$$

where A_0 is a target area for a cell, and lambda is some positive constant.

2. The membrane surface energy

$$U_S = \beta(C - C_0)^2 \quad (2.8)$$

where C is the cell perimiter, and C_0 is a target perimeter.

3. The cell-cell adhesion energy

$$U_A = \sum_{j=1}^n \gamma_j d_j \quad (2.9)$$

where n is the number of vertices in the cell, γ is some constant for the boundary in question between one cell and another, and d is the distance between one vertex and the next in a counter clockwise fashion. Note that in two dimensions the boundary is a distance d , but in three dimensions it would have to be the area of a cell face. Also take note of the fact that the gamma term could be implemented in various ways. I have chosen to assign a “stickiness” to each cell, and then the gamma term is calculated as the average of the stickiness of the two cells.

As seen in [9], this force on each vertex i is given by the negative gradient of the potential:

$$F_i = - \sum_{l \in N_i} (2\lambda(A_l - A_{0_l})\nabla_i A_l + 2\beta(C_l - C_{0_l})(\nabla_i d_{l,I_l-1} + \nabla_i d_{l,I_l}) + \gamma_{l,I_l-1}\nabla_i d_{l,I_l-1} + \gamma_{l,I_l}\nabla_i d_{l,I_l}) \quad (2.10)$$

where l is the l th cell containing vertex i , given a counter clockwise orientation. I_l is the local index of node i in element l .

The way to derive this force is not described in much of the literature, and only is partially described in [?]. I will explain the force in some detail here.

The area of a cell is given by Gauss's Shoelace Formula:

$$A = \frac{1}{2} \left| \sum_{i=1}^N (x_i y_{i+1} - x_{i+1} y_i) \right| \quad (2.11)$$

where $N+1 = 1$. Therefore, the gradient is given by:

$$\nabla_i A_l = \frac{1}{2} \left\langle y_{I+1}^l - y_{I-1}^l, x_{I-1}^l - x_{I+1}^l \right\rangle \quad (2.12)$$

where the superscripts l denote that x, y are in cell l . The subscripts are local indices in the cell l , and the orientation of vertices is counterclockwise. The circumference is given by:

$$C = \sum_{j=1}^N d_j = \sum_{j=1}^N \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2} \quad (2.13)$$

Therefore

$$\nabla_i C = \nabla_i d_{i-1} + \nabla_i d_i \quad (2.14)$$

and

$$\nabla_i d_{l,j} = \frac{1}{d_{l,j}} \left\langle x_{j+1} - x_j, y_{j+1} - y_j \right\rangle \quad (2.15)$$

Substituting the above values into the equation:

$$-\nabla_i U = -\nabla_i(U_D + U_S + U_A) = F_i \quad (2.16)$$

gives the force described above.

2.2.2 Academic Dishonesty, or Obvious Modeling?

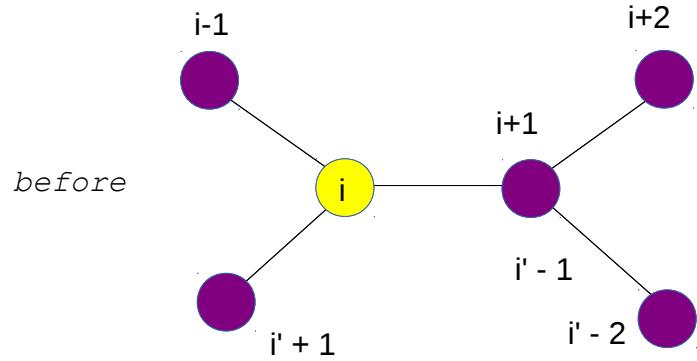
I am not at the point yet in my understanding of the physics of epithelial tissue to comment upon whether this model ought to be intuitively obvious to the physicist, or whether it is a carefully chosen model which has discarded many extraneous possibilities. A recent article [?] gives the equation for the potential in the mesh as:

$$E_i = \sum_{\alpha} \frac{K}{2} (A_{\alpha} - A_{\alpha}^0)^2 + \sum_{i,j} \Lambda_{ij} l_{ij} + \sum_{\alpha} \frac{\Gamma}{2} L_{\alpha}^2 \quad (2.17)$$

Where i denotes vertex i , α denotes a cell of which i is on the border, the A and A_0 terms denote area and target area, and the l_{ij} denotes the length of the boundaries on which i lies, and the L denotes the perimeter. The H. Honda or T. Nagai are not cited in the bibliography to this paper by several well known and respected scientists, so either this model is so obviously adequate that it isn't necessary to cite the original author, or this is a case of academic dishonesty.

2.2.3 Topological Changes to the Mesh

There is empirical evidence that nearly all vertices in a sheet of epithelial tissue have degree three. Since the coordination number of the almost all vertices is three, this has led many modelers to consider what sort of topological changes can occur in mesh of these types of cells without changing the connectivity. As it turns out, there are three changes which can occur. The first is called a T1 swap and is illustrated in Figure ???. This is also called a "neighbor exchanging swap" because, as you can see, two cells which were adjacent cease to be neighbors and two cells that weren't adjacent become neighbors. The T1 swap occurs when two vertices become critically close to each other, and instead of allowing the force to drive the vertices into each other we rotate their edge by 90 degree. In nature this should correspond to two vertices getting very close, colliding, and then flattening out into an edge. Our model performs this action discretely as a simplifying measure, however, because otherwise for a moment a vertex would have degree four, and this would be difficult to



<A T1 SWAP>

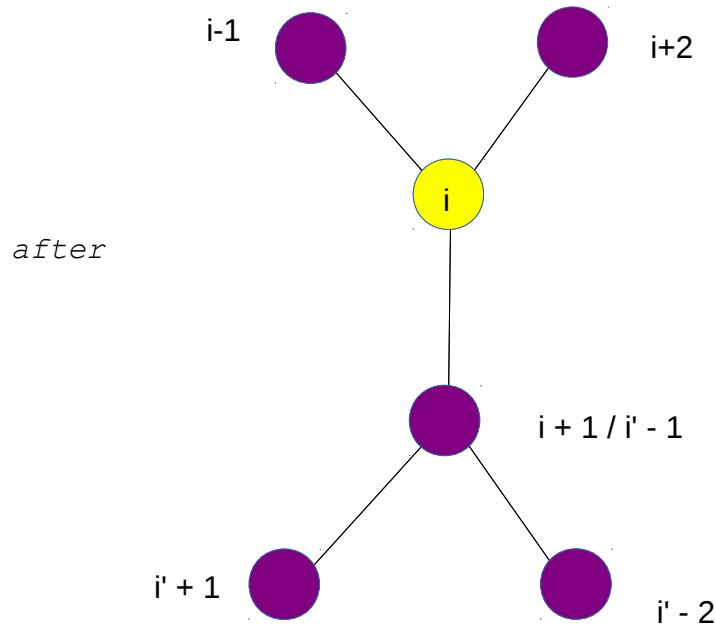


Figure 2.3: A T1 Swap. The numbers reflect the counterclockwise storage of the vertices. The prime serves to distinguish an index in the top cell from an index in the bottom cell.

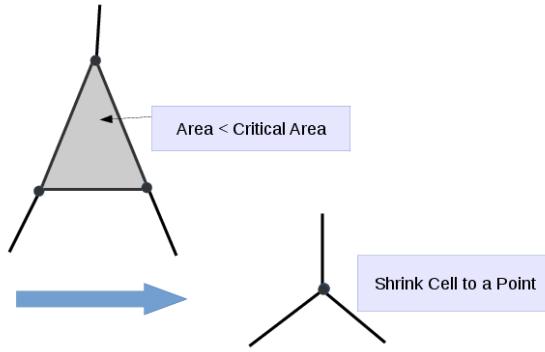


Figure 2.4: A T2 Swap

handle with out a very complex data structure. The second topological change is the T2 swap, which is also known as the "cell removal swap." This occurs when a triangular cell becomes too small and is deleted and replaced by a single vertex. The last swap is cell division, occasionally referred to as the T3 swap.

Cell division was not a part of the original Honda-Nagai Model, but it is now a very hot topic in computational tissue morphogenesis; it deserves mention. The challenge with implementing the T3 swap is that there are infinitely (within the bounds of floating point arithmetic) many choices about where to divide a cell, and there are several competing opinions (though no unanimously accepted theory) about how the dvision is oriented. Some cells divide along their longer axis, which is known as the 'Hertwig's Long Axis Rule', but global tissue stress and local cell geometry are also thought to affect the orientation of cell division [28][11]. The computational implementation of T3 swap is trivial, as the swap occurs by placing two new vertices in the interior of opposite edges and then connecting them by a new edge. The trouble is that it is not clear which edges ought to have vertices implanted, or where to insert these vertices, and the choice of where to divide a cell in a proliferating tissue can have profound effects upon the geometric appearance of a tissue [22]. There will be more discussion of cell division in the section of the paper dealing with the technical details of the implementation of *Epithelium*.

2.2.4 Quality, Not Quantity

The equations in this model are dimensionless. I will not undertake a discussion of how to derive the dimensionless model from the dimensional model, but for the curious reader this

is all laid out in [21]. Typically, one would not choose the values of the parameters α , β and γ , but would instead have some dimensional biological data and go through the necessary conversion steps to convert these parameters to the simpler ones. Interestingly, in vertex dynamics literature I have read and cited in the bibliography, the only clearly stated parameters appear in [9], and these are said to be taken from several papers which the authors (though I myself did not find these values explicitly stated).

There is little difference between equilibrium states in the Nagai-Honda model, as can be seen in the section of this paper dealing with the output of the code. Still, it has been shown that different parameter values coupled with other mesh changing operations (such as oriented cell division) can cause drastically different types of morphogenesis [10]. For example, drosophila wings, with their highly oriented divisions, have been shown to contain approximately 80% hexagonal cells whereas cucumber epidermis contains approximately 47% hexagons [22]. While all epithelial tissue has a strong tendency towards achieving an equilibrium dominated by hexagons, the width of the distribution of cell shapes differs by cellular structure and, hence, by parameter choices [31].

For those interested I have included in this paper charts which display the equilibrium meshes for different parameter values, along with a graph of the equilibrium distributions of cell sizes. These tables are found in chapter TITLE OF CHAPTER HERE. What you will see in the charts is that my implementation of the Honda-Nagai Model reproduces the results of the authors' original paper in that given a wide variety of parameterizations the mesh tends towards six sided cells. Here is one graphic from [21] in which they illustrate the distribution of cell shapes as a function of time.

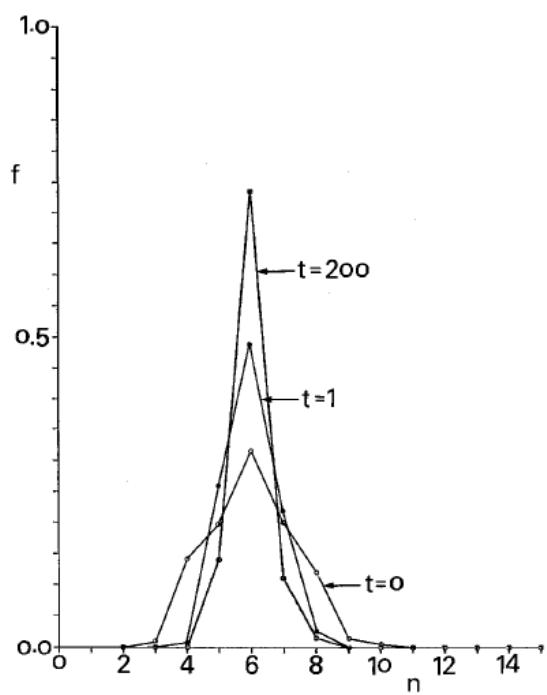


Figure 2.5: The Distribution of Cell Shapes As a Function of Time [21]

Chapter 3

Further Remarks About Epithelial Tissue and the Honda-Nagai Model

3.1 The Euler Characteristic and Its Implications

In this section I will expand upon some observations made in [31]. **Euler's Formula** is an equation which relates the number of edges, faces, and vertices in a graph or polyhedron. An invariant χ relates the faces, edges, and vertices as follows:

$$\chi = V - F + E \tag{3.1}$$

The invariant depends upon the graph or polyhedron in question. We will ignore the exact value of χ and comfort ourselves with the fact that it is a constant. The majority of current vertex dynamics models assume that vertices will have a coordination number of 3, and are based upon empirical results from biology. While there are certain models which consider *rosettes*, for now we will assume that all vertices connect to exactly three other vertices. Then we notice that all edges have two vertices, and that all vertices are connected to three edges. Initially, our intuition tells us that there should be three times as many edges as vertices, which leads us to the incorrect:

$$3V = E \tag{3.2}$$

But then we notice that if we consider all of the vertices in the mesh, we count each edge twice, so we divide the conjectured number of edges by two and then simplify to get:

$$3V = 2E \tag{3.3}$$

Similarly, if we consider how to relate the number of edges to the faces in the mesh, we conjecture that the number of edges is equal to the number of cells with 3 sides plus the

number of cells with 4 sides plus the number of cells with 5 sides plus . . . each multiplied by the number of edges per cell. THis gives us:

$$\sum_{k=3}^N kF_k = E \quad (3.4)$$

Where N is the highest number of edges encountered by any cell in the mesh. But in this way we have again counted all of the edges twice, so the true number of edges must be the summation above divided by 2. We simplify the equation to :

$$\sum_{k=3}^N kF_k = 2E \quad (3.5)$$

Now, we are able to reduce Euler's Forula to one variable using the relationships given above.

$$V - F + E = \chi \quad (3.6)$$

$$\frac{2E}{3} - F + E = \chi \quad (3.7)$$

$$\frac{5E}{3} - F = \chi \quad (3.8)$$

$$\frac{\sum_{k=3}^N kF_k}{6} - F = \chi \quad (3.9)$$

$$\left(\frac{\sum_{k=3}^N kF_k}{F} - 6 \right) F = 6\chi \quad (3.10)$$

Biological cells are very small, and an epithelial tissue is composed of many [NUMBER?] cells, so we assume that $F \rightarrow \infty$ and then immediately notice that the expression in parentheses must tend to zero as F goes to infinity, or else the left hand side of the above equation will not approach the constant 6χ . So we know that the algebraic mean of the number of vertices per face must be 6. Of course we have no reason at this point to assume a distribution which guarantees that the majority of cells in the tissue will have exactly six edges and will not, say consist of a mixture of 5 and 7 sided cells. Nevertheless, empirical evidence shows a strong central tendency in the distribution of cell shapes. Whenever a cell tries to stray from the mean, there are means (such as a T3 swap, described in the next section) of recentering the distribution at 6 sides.

3.2 Why the Specified Topology Is Not Perfect

The choice to impose degree three on each vertex is not one hundred percent consistent with nature, but has been a part of most models of epithelial tissue. This is because empirical

evidence shows that the *majority* of vertices in a tissue will have out degree three. It is a simplifying assumption that *rosettes* (epithelial cells organized radially about one vertex which has degree greater than or equal to 4) do not change the global dynamics of the development of an epithelial tissue. Recent computer vision developments [17] have made it easier to detect rosettes in epithelial tissue samples and may in the future provide information about the number of these formations, or evidence that rosettes are an important feature of epithelial tissue.

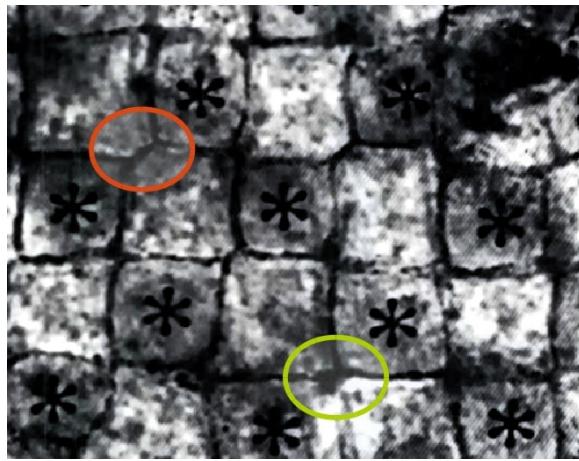


Figure 3.1: An image of Japanese quail epithelium. This tissue is patterned like a checkerboard and has been modeled in the past as a tissue with degree three vertices. It is possible that modelling this tissue as degree four vertices would provide better approximations of the dynamics. In particular, notice that the orange vertices clearly can be modeled with degree three, whereas the green vertex could be one degree four vertex or two degree three.[32]

3.3 Feedback Mechanisms and Proliferation

This model is limited in that it includes neither mechanical nor chemical feedback, and does not specify how cells proliferate. These are two extensions which can and have been made to the model by certain researchers. On the other hand, the dynamics of some successful models such as [29] have the patterning of cells specified entirely by morphogens. In these models, chemicals called ‘morphogens’ are what drive cells to proliferate, and the low concentrations of the morphogen on the fringe of the tissue explain why the tissue eventually ceases to grow as the tissue reaches a certain size. Other models, such as those described in [2] are based

upon the vertex dynamics models such as the Honda-Nagai model, but include additional mechanical feedback terms which limit the growth of cells.

3.4 Additional Curiosities

The vast majority of models assume that exactly three cells meet at any vertex, except vertices on the boundary. In the language of topology, one would say that all interior vertices in the tissue have degree three. Empirical observations show, however, that more than three cells can meet at any junction under certain circumstances [8]. This means that models ought to be extended to include the formation of these ‘rosettes’.

Chapter 4

GrowFlesh

“All models are wrong, but some are useful” - G. Pox

4.1 About Epithelium

For my Master’s Thesis I have implemented the Nagai-Honda Model for epithelial tissue development. My software is easy to install, takes up less than 50Mb, comes in parallel and non-parallel versions (Version 1.0.0, Version 1.0.1) and has very few dependencies (and they are likely already installed on mac and linux computers).

My code allows users to specify all parameters of interest in a couple of well commented configuration files, can handle simulations of arbitrarily large size, and can generate beautiful animations and plots of epithelial tissue development as well as useful plots of various quantities which are of interest to the researcher.

On top of all of this, the source code is highly modularized and allows for ambitious users to easily extend the code to meet their needs. For example, alternate numerical integrators can easily replace the existing one, new mesh generators can replace the square mesh I have developed, and all data is output in very simple formats(which users with scripting experience can transform to fit into the graphical utilities of their choice). In addition, the cell and vertex classes are well documented and can be extended to output new data, as users may need. What follow are a few graphs to give some flavor of what *Epithelium* can do.

Mention how Chaste was a step in the right direction, but the dependencies are too many and the it is difficult to get started. Epithelium is lightwieght and the code is easy to read, a better introductory tool.

4.2 Sample Configuration Files

4.2.1 config.txt

In this file, the user can specify some global properties about the mesh, and some important quantities for how the simulation will proceed. Most of the quantities are self explanatory. The dimension of the mesh is explained in the Code Design chapter of this paper. The swap length , upper bound, and Max Swaps parameters are used for performing random transformations to the mesh in the beginning of the simulation. The swap length is the edge length below which a swap is performed. The Max Swaps is the maximum number of random perturbations the code will make to the mesh. The upper bound is an integer which is upper bound of the range for a random number generator. The random swaps occur when a ‘1’ is generated, and we choose random numbers from [1:upper bound].

OFF is the file format given to the plotting program geomview to plot the mesh. You can specify how often the code prints an image. It must be a multiple of ten, so if you want more closely spaced images you can just drop the maximum step size for the numerical integrator. The numerical integrator takes adaptively sized steps, and this will be discussed in more detail later.

```
13 # Dimension of mesh MUST BE ODD!!!!  
1 # Plot the energy in the system? [1/0]  
1 # Make a movie in the end? [1/0]  
0.01 # Maximum step size  
1000 # Number of iterations  
10 # frequency of OFF file output. Must be a multiple 10!  
.1 # delta minimum vertex separation.  
1000 # Max swaps  
1.5 # swap length  
2 # upper bound random number generator  
1 # Make a bar graph of cell sides? [1/0]  
0 # Make a histogram of the areas? [1/0]  
0 # Make a histogram of the perimeters? [1/0]
```

4.2.2 parameters.hpp

In this header the user can define some global parameter values for the cells in the mesh. Unfortunately, you must recompile every time a parameter is changed. This could be rewritten such that parameters are read in from a configuration file, but that is just not the way the code was developed.

```
const double beta = 0.1;
const double lambda = 0.1;
const double pi = 3.141592;
const double t_gamma = 1;
const double t_area = 14.0;
```

4.2.3 changemesh.txt

This file is a configuration file, and does not require you to recompile when you change it. You can fine tune the properties of the cells in the mesh with this file.

```
# This file contains number of gamma changes ,
# followed by the number of area changes ,
# followed by the indices and new values
2 3
16 4.0
17 5.1
3 1.0
4 1.0
10 3.7
```

4.3 Image Gallery

This selection of graphs captures the properties of the mesh with various choices of parameter values. The target area for each cell was set to 4.0, unless otherwise specified. A couple of plots have outliers in the number of sides the cells have, or in the perimeters attained. This is due to the individual specification of cell properties for a handful of cells. In this way, the distribution of cell properties was more finely controlled. The images come in groups of four. There are two parameter studies per page. On the right you will see the

equilibrium area and perimeter reached after applying the force, and the on the right you will see the equilibrium distribution of cell shapes along with a plot of the decreasing energy in the mesh as a function of time.

Following the parameter studies several interesting tissues are presented.

4.4 Why *Epithelium* is a good piece of software

To close our discussion of *Epithelium* I would like to discuss why it is a great piece of software for *you*. *GrowFlesh* has only a couple of dependencies which are likely already installed on your computer if you are interested in computational science. You will need a compiler supporting the C++11 standard, python2.7, the matplotlib plotting library for python and the numpy python library. The only packages which will likely not be installed on your computer are the geomview geometric visualization software and the imagemagick image conversion program. Geomview and ImageMagick are free, and are such good programs that they have been around for more than 20 years. Of course, all of the animation, and plot information is outputted from this software in file formats that are easy to adapt to the user's choice of visualization software. The energy and histogram files are space separated lists and the mesh information is outputted in the OFF file format, which features a list of coordinates followed by a list of polygons which are defined counterclockwise as a list of vertex indices from the preceding list of vertices. A sample mesh file is shown below to convince you of how simple this file format is, and to suggest that you could write a small script to change this file to a different format that will be acceptable to your choice of visualization software.

```
{appearance {+edge}  
OFF  
# This is the OFF format for a square.  
# We have the numVerts, numFaces, and a zero  
# Then a list of vertex coordinates  
# Then a list of faces described counterclockwise  
# by their vertices.  
4 1 0  
1 0 0  
1 1 0
```

0	1	0		
0	0	0		
4	1	2	3	4

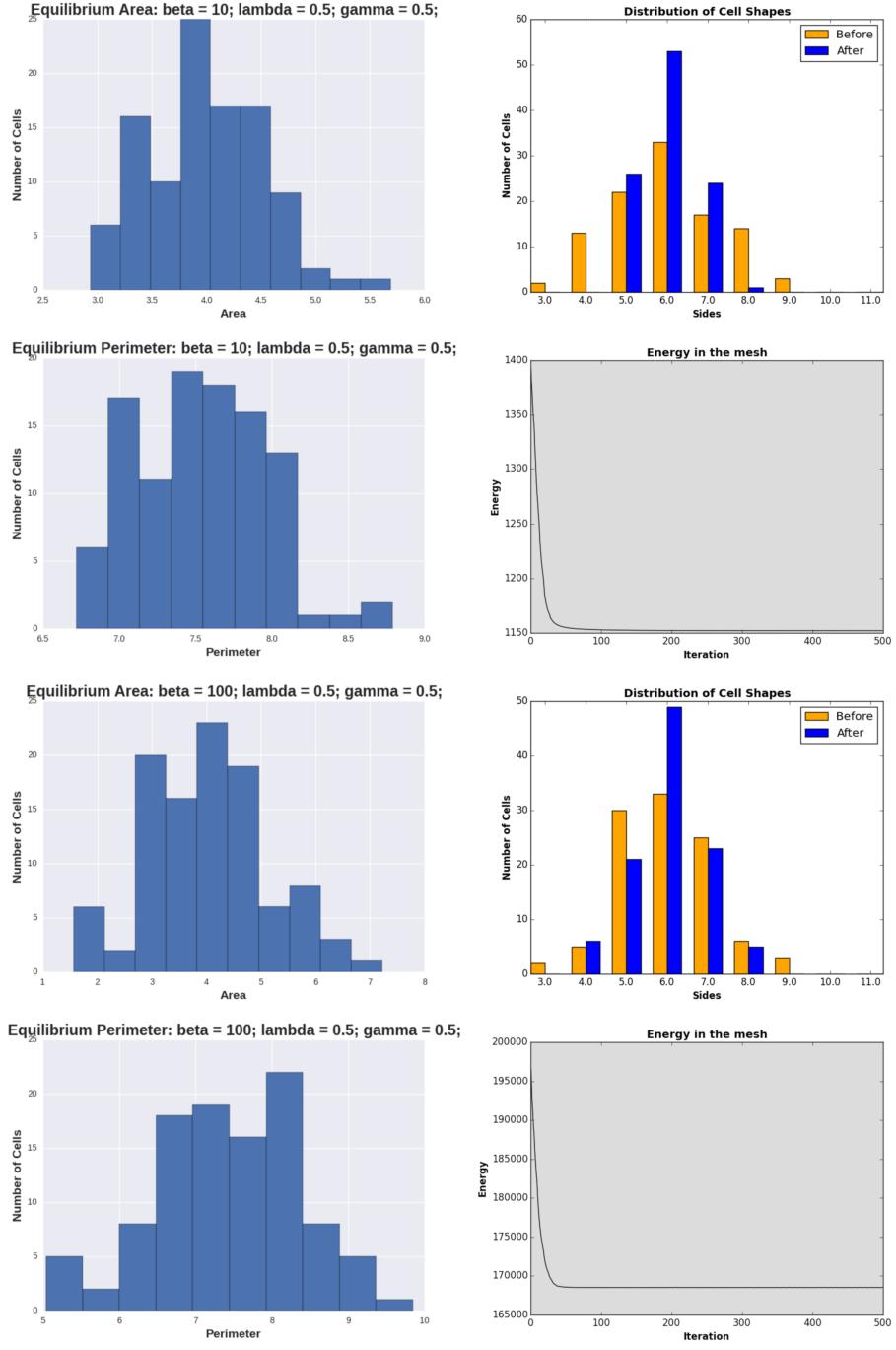


Figure 4.1: These graphs show the equilibrium reached when beta is the dominant parameter.

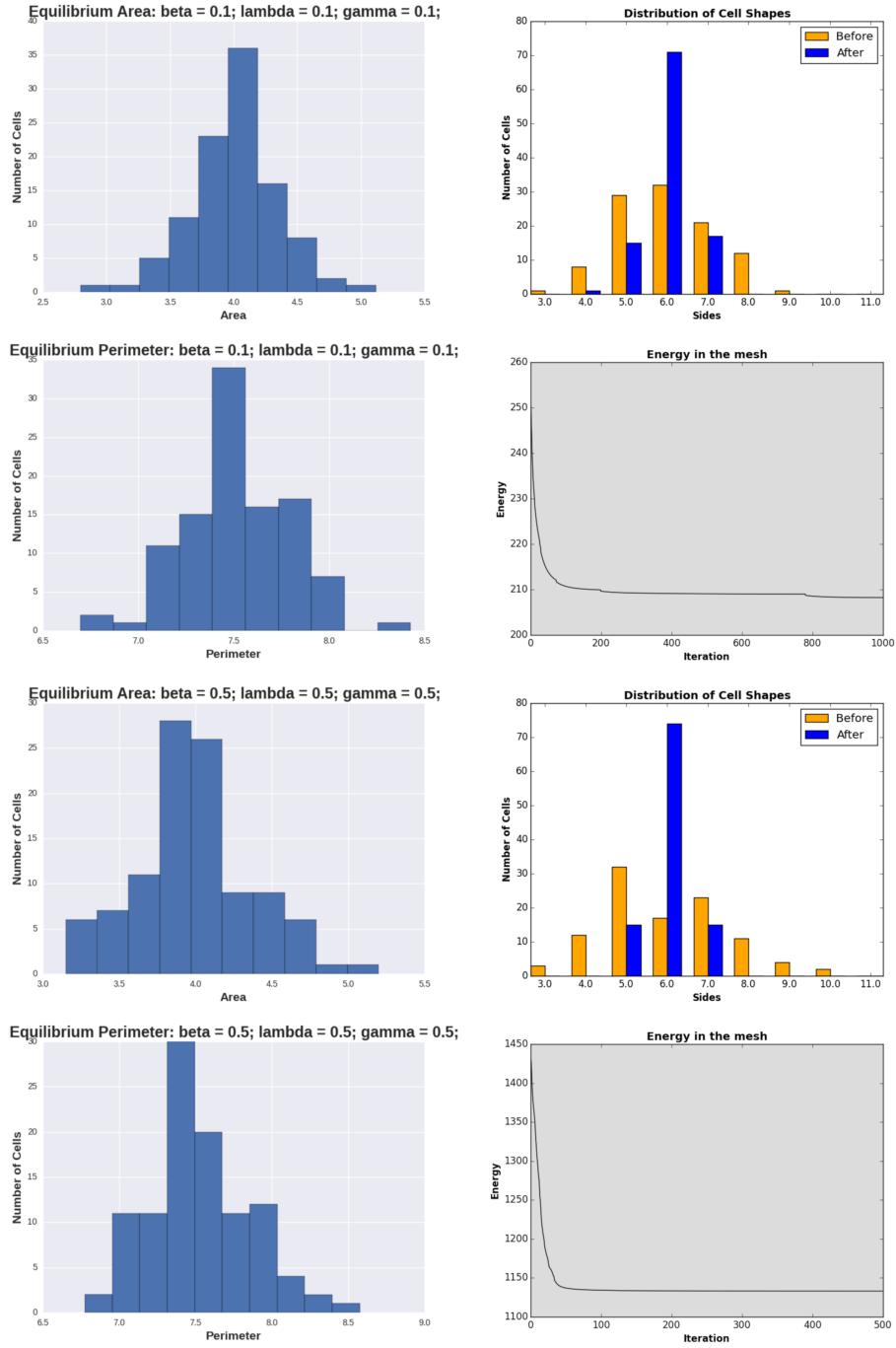


Figure 4.2: These graphs show the equilibrium reached when all parameters are set equal.

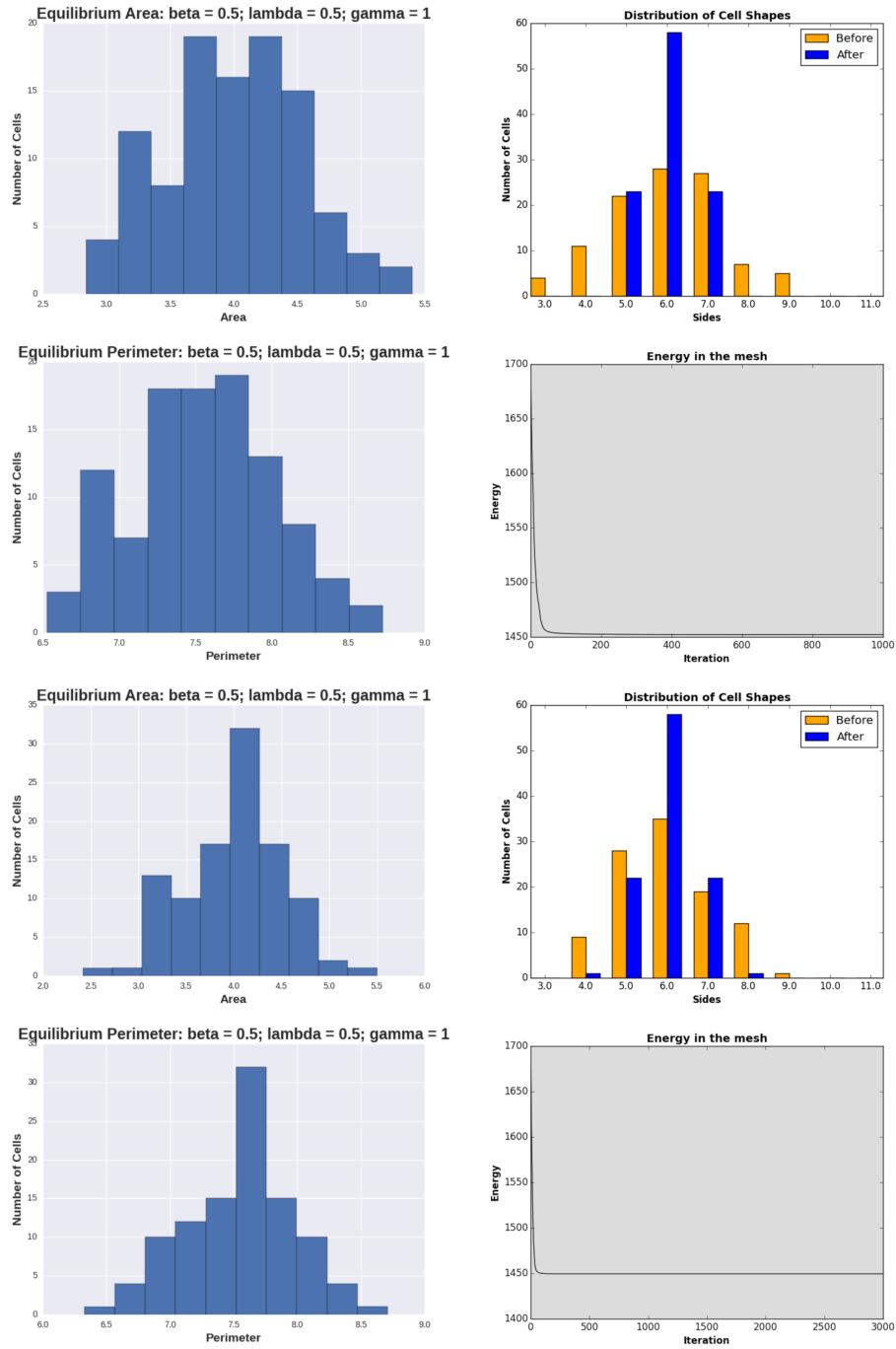


Figure 4.3: These graphs show the equilibrium reached when gamma is the dominant parameter.

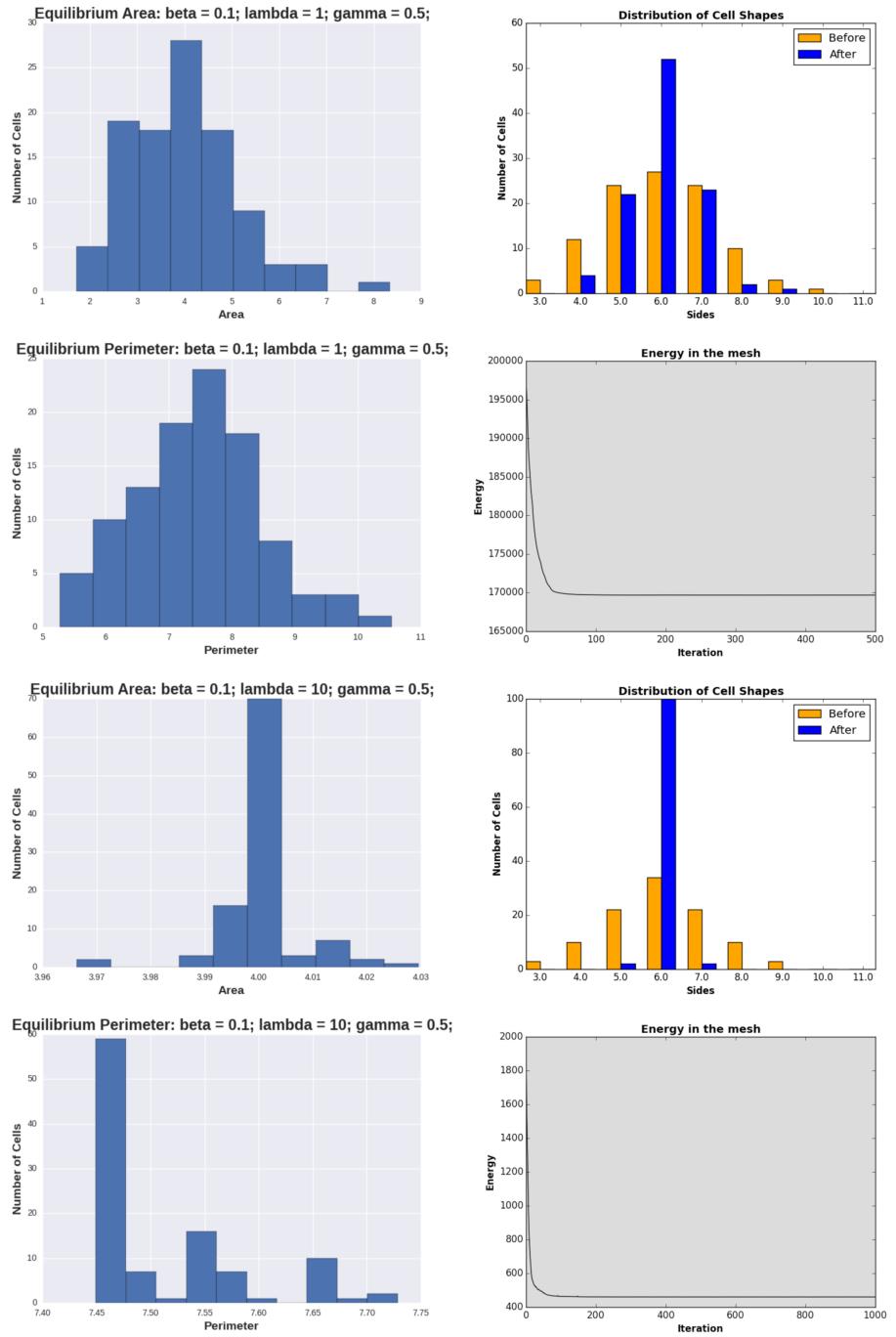


Figure 4.4: These graphs show the equilibrium reached when lambda is the dominant parameter.

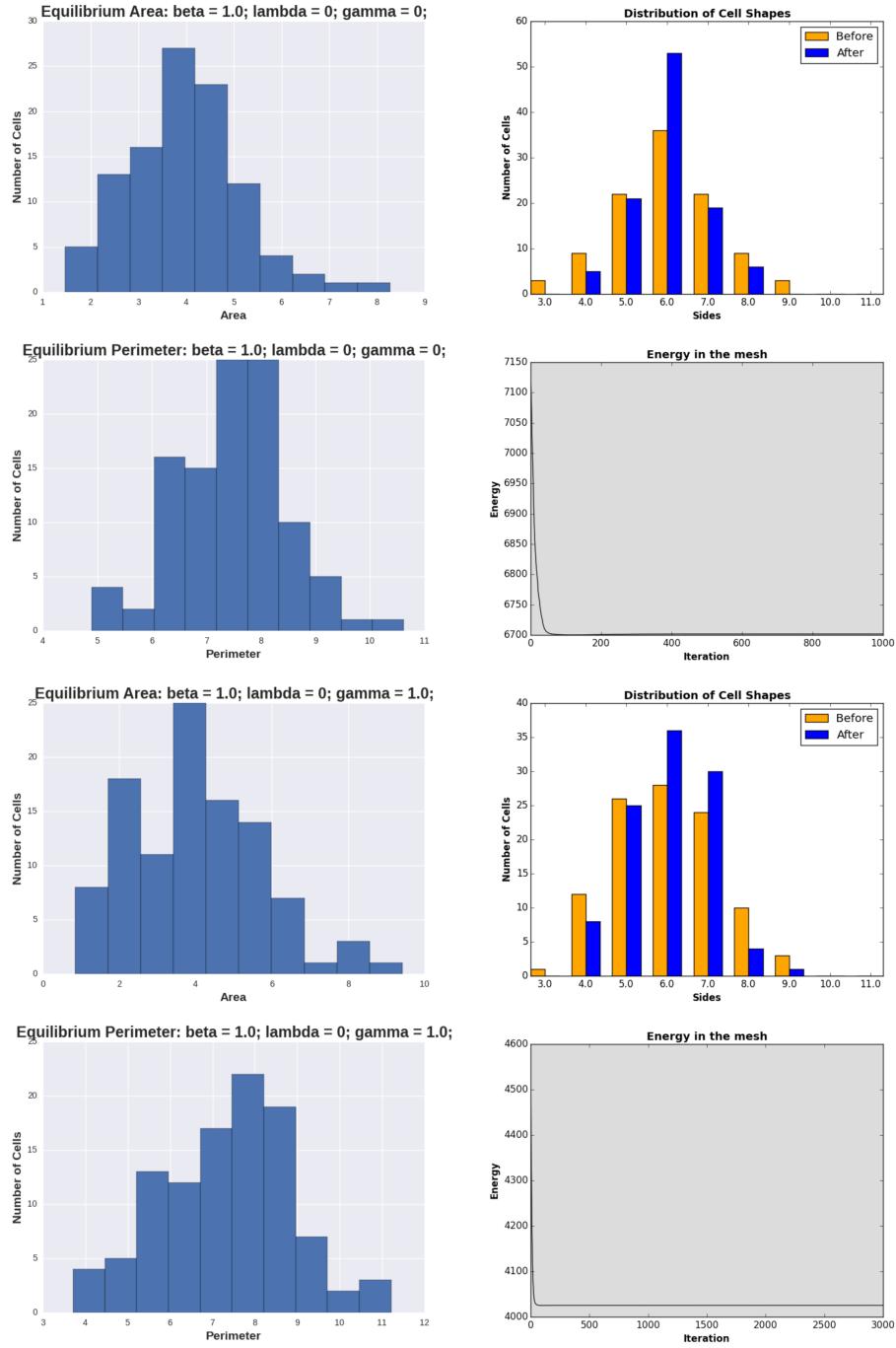


Figure 4.5: These graphs show the equilibrium in the absence of a parameter.

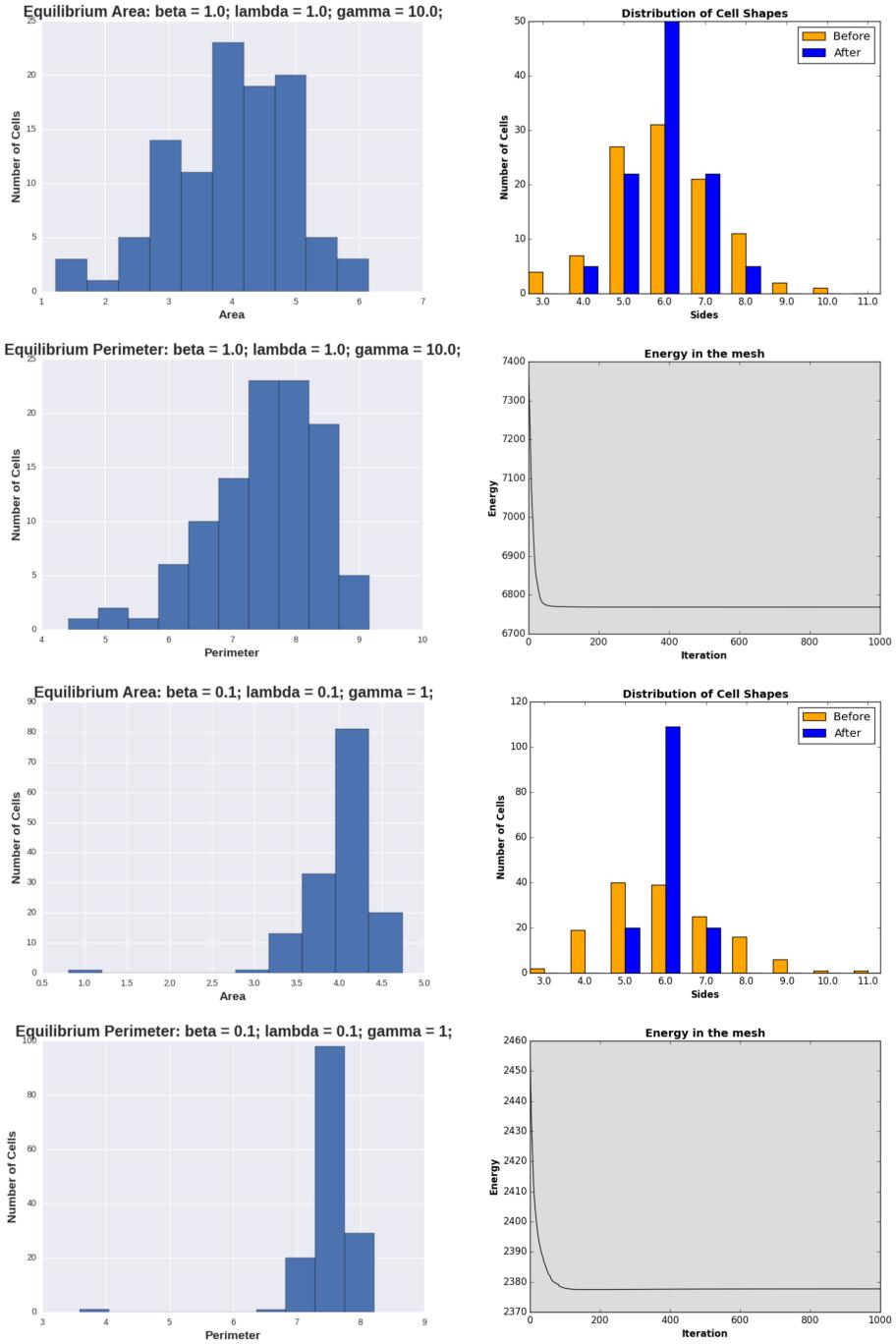
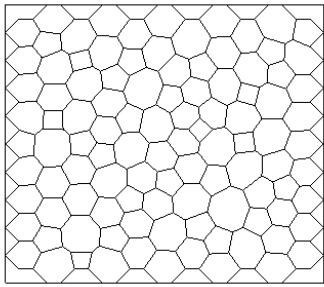
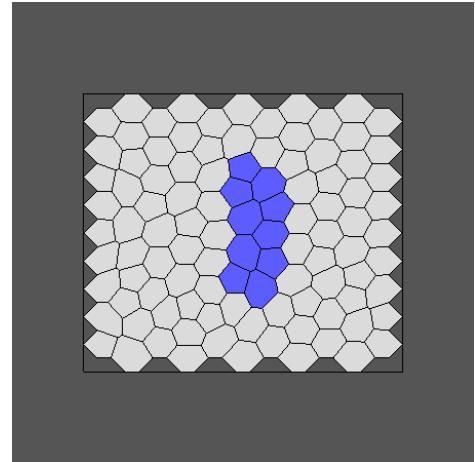


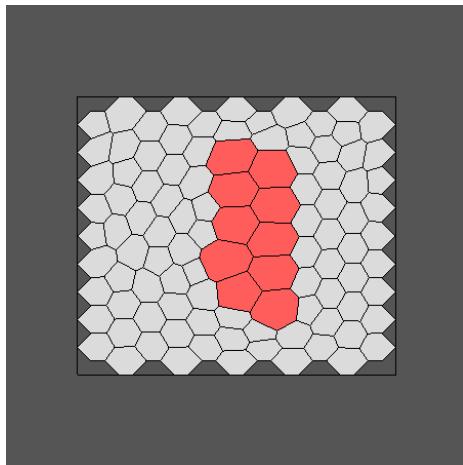
Figure 4.6: These graphs show the equilibrium reached when the target area is set to be larger than the available space in the mesh, and there is also a graph showing another gamma-dominant equilibrium.



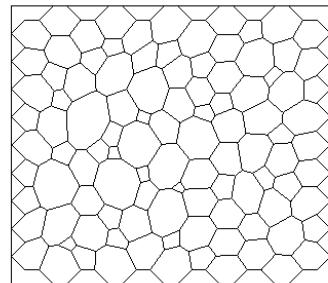
(a) A Final Mesh With White Background.



(b) A final mesh where some cells have target area < 4.0



(c) A final mesh where some cells have target area > 4.0



(d) A mesh during an integration, before equilibration.

Figure 4.7: A Selection of Interesting Tissues

```
*****
***** Nagai-Honda Model Simulation *****
*****  
  
Dimension of mesh: 11  
Number of iterations 100  
Maximum step size for the integration 0.01  
Delta for T1 swap 0.1  
An image will be printed every 10 iterations.  
The maximum number of initial T1 swaps is: 1000  
61 initial random T1 swaps occurred!  
The number of cells in this simulation is 104  
The number of coordinates in this simulation is 252  
  
Starting the integration...  
    Current iteration: 0  
The computation finished in 5.5279 seconds!  
The final error in the mesh is 1.12347e-14 units measured  
Plotting the system energy...  
Making animation...  
Making a bar graph...  
Making area and perimeter histograms...  
  
*****Simulation Complete*****
```

Figure 4.8: A screenshot of the simulator

Chapter 5

The Design of *GrowFlesh*

In this chapter I will explain the internals of *GrowFlesh*. I will outline the data structure I used for the model, and some of the interesting algorithms I designed to calculate inter- and intra- cellular forces. It is interesting to note that a number of datastructures and algorithms were scrapped in the beginning of the modeling process because of the complicated relationship between and vertices. The model I have implemented is a vertex dynamics model, and, as such, one would think that the main object of interest should be a vertex. Vertices require information about the cells which contain them in order to be moved. With this as a starting idea, I developed a sophisticated vertex class which contained cells as member variables. Apart from the software bloat coming from many vertices containing massive amounts of cell information, another issue was that cells are made of vertices, and, as such, cells ought to contain vertices as member variables! Unfortunately this bilateral inclusion is not supported by C++. In this chapter I will describe how this issue was circumvented through the use of tables which mimic the behavior of a relational database.

5.1 [Highly Simplified] Pseudocode

Here I will briefly outline how the code works. All of the functions are explained in some detail later in this chapter.

```
mesh_variables $<$- read_configs()
mesh $<$- make_mesh()
random_alterations(mesh)
copy(mesh, rotate_mesh)
rotate(rotate_mesh)
print(simulation_info) # So the user can verify all parameters.
for i = 1:num_iters
    if (iter%print_freq == 0)
```

```

    print(OFF_file)
    temp_mesh = NagaiHondaForce(mesh) # if any displacement in temp_mesh,
                                    # the displacement is recalculated
    temp_rotate_mesh = NagaiHondaForce(rotate_mesh)
    mesh $<-$ mesh + temp_mesh
    rotate_mesh $<-$ rotate_mesh + temp_rotate_mesh
    performT1(mesh)
    performT2(mesh)
    performT1(rotate_mesh)
    performT2(temp_rotate_mesh)
    rotate_back(rotate_mesh)
    compare_mesh(mesh, rotate_mesh)
    print(graphs and error analysis)

```

5.2 Classes

GrowFlesh uses two classes to organize data. They are the cell and coordinate classes.

5.2.1 The Cell Class

The cell class contains a number of useful functions and data members to make the code easy to read and understand. All cell information could have been stored in arrays, but the OO structure makes the code more readable. All cells know their index, which vertices (coordinates) make them up, they are able to calculate their area and perimeter, can modify their constituent vertices, can tell you whether or not they contain a vertex, and can print out a graphical, color coded representation of themselves to an OFF file.

```

public:
cell(int index, vector<int>& AssociatedVertices,
     double target_area = t_area, double gamma = t_gamma)
{
    assert(index >= 0);
    m_AssociatedVertices = AssociatedVertices;
    m_index = index;
    m_target_area = target_area;
    m_target_perimeter = sqrt(pi * m_target_area);
}

```

```

    m_gamma = gamma;
}

cell(){} // Default constructor

vector<$int$> GetVertices(){return m_AssociatedVertices;};
int GetIndex(){return m_index;};
void SetIndex(int index){m_index = index;};
void SetTargetArea(double area){m_target_area = area;};
double GetTargetArea(){return m_target_area;};
double GetTargetPerimeter(){return m_target_perimeter;};
double ComputeArea(double * X, double * Y);
double ComputePerimeter(double * X, double * Y);
void PrintCell(ofstream &OffFile);
int ContainsVertex(int index);
void SetGamma(double gamma){m_gamma = gamma;};
double GetGamma(){return m_gamma;};
void InsertVert(int v1, int v2);
void EraseVert(int index)
{
    vector<$int$>::iterator it = find(m_AssociatedVertices, index);
    m_AssociatedVertices.erase(it);
};
void ReplaceVert(int before, int after)
{
    vector<$int$>::iterator it = find(m_AssociatedVertices, before);
    *it = after;
};
void SetVertices(vector<$int$> vertices)
{
    m_AssociatedVertices = vertices;
};
int GetNumSides(){return m_AssociatedVertices.size();};

private:
vector<$int$> m_AssociatedVertices; // Stored counterclockwise
int m_index;

```

```

    double m_target_area;
    double m_target_perimeter;
    double m_gamma;
};


```

5.2.2 The Coordinate Class

The coordinate class stores the index of a coordinate, and whether or not the vertex will move during the integration. *GrowFlesh* will run with two types of meshes: meshes with border, and meshes without. A mesh with border is a mesh with fixed border elements. A mesh without border is a mesh which was generated with periodic boundary conditions and which maintains the periodicity throughout the calculations done by the program. Border vertices will not move, whereas interior vertices will be able to move; the coordinate class allows us to specify whether or not a vertex is interior or exterior. Another benefit of this class is that it allows us to easily extend the code to include forces acting on individual vertices, and to specify other types of vertices besides interior and exterior. This code was developed with eyes to the future. Another interesting idea to explore in computational biology is active cell migration, and the cell class allows us to specify which vertices want to move actively.

```

class coordinate
{
public:
    coordinate(int idx, bool t) : index(idx), IsInner(t){};
    coordinate() {index = -1; IsInner = 0;};
    int index;
    bool IsInner;
    inline bool operator==(const coordinate& rhs)
    {return index == rhs.index;};
};


```

5.3 Initial Mesh Design

The `hex_mesh()` function generates an $n \times n$ mesh of cells, where the dimension represents the number of cells touching the boundary.

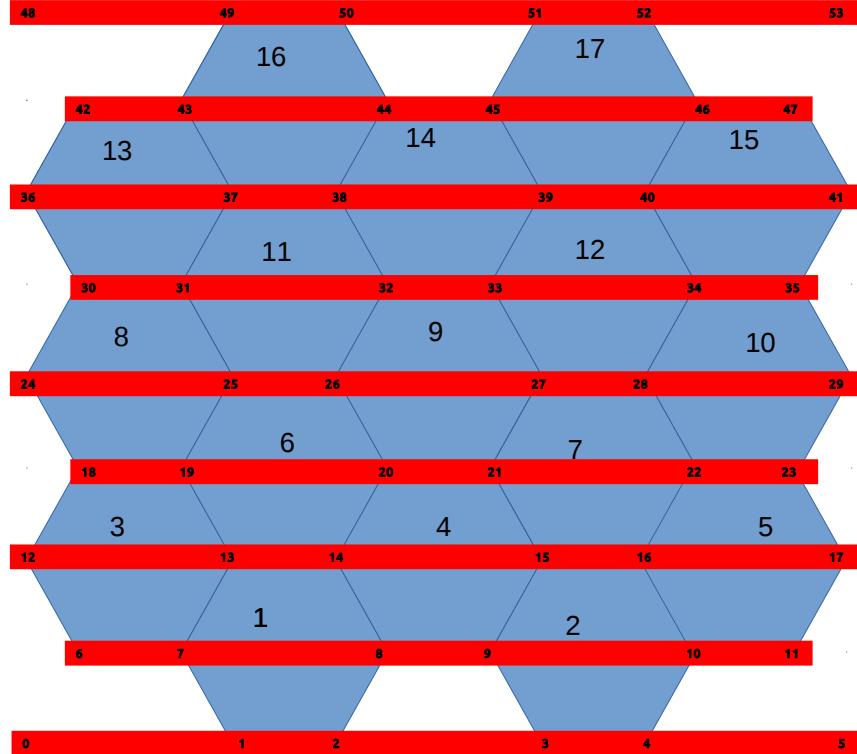


Figure 5.1: A 5x5 hexagonal mesh. The cell indices are written in the cells and the vertex indices are written on the red bars.

5.4 A Relational Database

A popular way to store data since the 1970's is to store data in group of tables which are connected via *keys*. This type of database is popular for reasons which will become apparent by means of a simple example.

Consider a business which sells a number of products, and wants to keep track of their customers, the customer's orders, the customer's addresses, and information about the products ordered. A wasteful way to store this data is to store a large table with a column for customer. Next to every customer's name is the customer's address. Next to the customers address is the customer's order number. Next to the customer's order number is an item in that order. And, next to each item ordered is the item information. This method of storing data is terribly redundant, for each customer you would need $\sum_c \sum_{o_c} |o_c|$ rows in the table, and many rows would have the customer's information repeated. Similarly, item information would be repeated in every row that features this item.

A better idea is to break this data up into several tables which together define a *schema*. In this example, the following tables define a good schema:

1. Customer information. columns: name, address, order number
2. Order numbers: columns: order number, item
3. Items: columns: item, item description

We are now guaranteed that the customer information is not duplicated for every item in an order, and that item information is not duplicated every time an item is in an order.

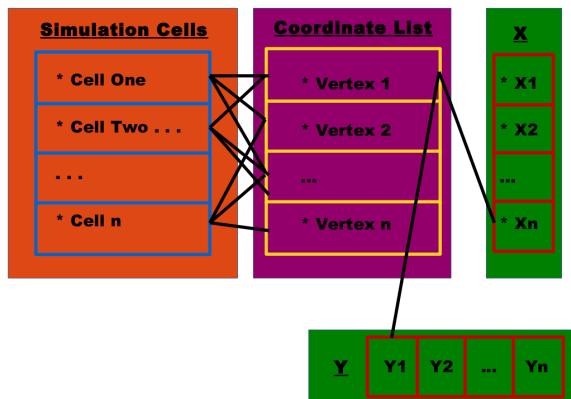


Figure 5.2: The Relational Database

My data structure is inspired by the relational database model. I have six tables, including the simulationCells, coordinateList, X, Y, tempX and tempY tables. The cell and coordinate tables are implemented as 1d vectors of cell and coordinate indices, whereas the position tables are implemented as 1d arrays for ease of passing these structures to CUDA C functions (will talk more about CUDA C later). The cells can extract coordinate information from the coordinateList table via the *index* key, and the coordinateList can access the position information from the X and Y arrays via their own *index*. The temporary X and Y arrays store temporary position information about the vertices before the mesh positions are updated. This choice saves memory because the cells, coordinates, and coordinate locations are stored independently of each other, and there is no data redundancy.

5.5 Moving the Vertices

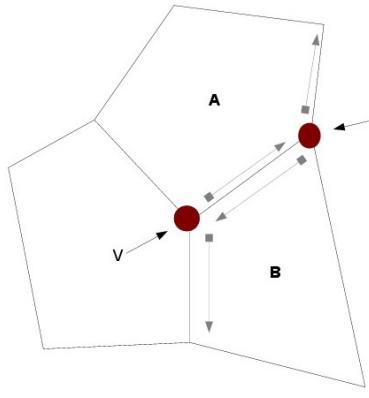
GrowFlesh loops over the vertices, computes the force applied to each vertex and then computes a displacement due to the force using the Euler Method. In their original paper, H. Honda and T. Nagai described the use of a Modified Runge Kutta Method to move the vertices, but this method would result in extra unnecessary computations at each time step. The *Error Tolerance* section of this chapter describes how the Euler Method is numerically stable enough for this application. The same decision to use then Euler Method was made by the research group at Oxford that developed CHASTE, the *other* leading software for implementing the Nagai-Honda Model.

A displacement is calculated and stored in the temporary X and Y arrays. No vertex is permitted to move more than one half of the minimum delta separating vertices (the δ under which a T1 swap will occur) during an integration. By imposing this restriction we are ensuring that we will not miss the event of two vertices coming critically close and a T1 swap occurring. Also, this prevents vertices from passing each other and invalidating the mesh. To ensure that no vertex moves too much, we store verify each displacement as we put it in the temporary X and Y arrays. If a displacement is too large, then the entire array of temporary displacements is erased, the time step is halved, and we begin the integration again. We could label the integrator as ‘fault tolerant’.

Another important aspect of the numerical integration is that cell and vertex information must be processed in counterclockwise order. In Figure 5.3 I illustrate how a vertex can find

the cells surrounding it in this order. When we are integrating a vertex i , the vertex first searches in the cell vector for a cell which contains it, and then the vertex finds the next vertex in that cell. Then, we have an edge. Some other cell contains that edge if it contains both of the vertices. That cell must be clockwise from the first. The cells are stored in the reverse order of which they are uncovered by this algorithm before the integration starts. This is a very expensive step ($O(n^3)$) in the computation, and ought to be optimized.

5.6 Embarassing Parallelism and CUDA



Algorithm for orienting cells counter-clockwise around vertex “ v ”

The numerical integrations and the vertex location updates exhibit what Cleve Moler describes as “embarrassing parallelism”. Computing the displacement of vertex a does not depend upon the computation of the displacement of vertex b during a given time step. Similarly, the vertex locations can all be updated in parallel since the update is simply a vector sum operation of the X and Y arrays with the temporary X and Y arrays, respectively.

Figure 5.3: Getting Cells in Order

A popular hardware choice for parallel programming in last decade has been the NVIDIA GPU. CUDA is the NVIDIA extension of the C programming language which can run certain parts of C code on the GPU, while still running the serial and i/o operation on the CPU. I wrote some parts of the code in this language, and these parts were very simple to implement. The GPU is a collection of small processors, and the vertices are small, simple objects, so I mapped each vertex in the mesh to one of the hundreds of small processors to get some computational speedup.

Specifically, CUDA was employed to update vertex locations, since the sum of 1d arrays is well suited

to a GPU. I also parallelized the rotation of the mesh and the reduction step which found the maximum error between the mesh and the rotated mesh. An interesting possibility is that of parallelizing the force computations. I was unable to do this with the code as it is because the cell data uses C++ vectors, while cuda only likes to take large C arrays as input. There is an interesting library called *thrust* which can strips the vectors down to arrays and does all of the memory allocations for you, but the issue then becomes one of compiling the cells into one big vector with clever structure. This is left as a future project.

CUDA was used to multiply each vertex in the mesh by the rotation matrix:

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

5.7 Computing Topological changes.

5.7.1 The T1 Swap

The PerformT1s() function loops over all of the cells in the mesh, and checks the edge lengths in that cell. If an edge is critically small, then the other three cells involved in the junction are found. One of the two clashing vertices is deleted from the main cell, and the other clashing vertex is deleted from the neighboring cell. The midpoint of the critical edge is calculated, and then a perpendicular line is drawn through it. The vertices are then moved a distance $\delta/2$ into the interior of the cells (δ is the minimum vertex separation, and the vertices now have this minimal spacing). Then, these two vertices are inserted into the cells which previously hadn't been neighbors, making them adjacent after the swap. Figure ?? offers a visual aid to understanding the swap.

There are 8 cases to consider when inserting moving the vertices the distance $\delta/2$. I and Ip1 are the critically close vertices, where Ip1 comes before I in clockwise order in a cell. mp is the midpoint of $(I, \bar{I}p1)$. dx and dy are the displacements which will give a minimal separation after the swap.

1. Given:

$I.x < Ip1.x$ AND $I.y == Ip1.y$ **Then:**

$I \mapsto mp + (0, -dy)$ AND $Ip1 \mapsto mp + (0, dy)$

2. Given:

$I.x > Ip1.x$ AND $I.y == Ip1.y$ **Then:**

$I \mapsto mp + (0, dy)$ AND $Ip1 \mapsto mp + (0, -dy)$

3. Given:

$I.x < Ip1.x$ AND $I.y < Ip1.y$ **Then:**

$I \mapsto mp + (dx, -dy)$ AND $Ip1 \mapsto mp + (-dx, dy)$

4. Given:

$I.x > Ip1.x$ AND $I.y > Ip1.y$ **Then:**

$I \mapsto mp + (-dx, dy)$ AND $Ip1 \mapsto mp + (dx, -dy)$

5. Given:

$I.x < Ip1.x$ AND $I.y > Ip1.y$ **Then:**

$I \mapsto mp + (-dx, -dy)$ AND $Ip1 \mapsto mp + (dx, dy)$

6. Given:

$I.x > Ip1.x$ AND $I.y < Ip1.y$ **Then:**

$I \mapsto mp + (dx, dy)$ AND $Ip1 \mapsto mp + (-dx, -dy)$

7. Given:

$I.x == Ip1.x$ AND $I.y > Ip1.y$ **Then:**

$I \mapsto mp + (dx, -dy)$ AND $Ip1 \mapsto mp + (-dx, dy)$

8. Given:

$I.x == Ip1.x$ AND $I.y < Ip1.y$ **Then:**

$$I \mapsto mp + (dx, -dy) \text{ AND } Ip1 \mapsto mp + (-dx, dy)$$

5.7.2 The implementation of the T2 swap.

The PerformT2s() function looks at every cell in the mesh and checks its area. If the area is critically small, then the cell is deleted from the mesh. The centroid of the triangle is calculated, and this becomes the collapsed triangle. Then, one of the three vertices has its position updated to the centroid coordinate and any cell which contained one of the other two coordinates is assigned the centroid coordinate index as a replacement.

5.8 Error Tolerance of the Algorithm

GrowFlesh has been empirically shown to be numerically stable; the code outputs an error measurement at the end of a simulation to give the user a sense of the stability. There was no clear way to provide an analytical proof of stability but at least we will always have an error measure to strengthen our confidence in the code. *GrowFlesh* runs two simulations at the same time, one on the mesh made by hex_mesh(), and another on the same mesh which has been rotated 45 degrees. Then, at the end of a simulation, the rotated mesh is rotated back and corresponding vertices are compared by index using the Euclidean norm. In practice we see near machine epsilon error for every simulation.

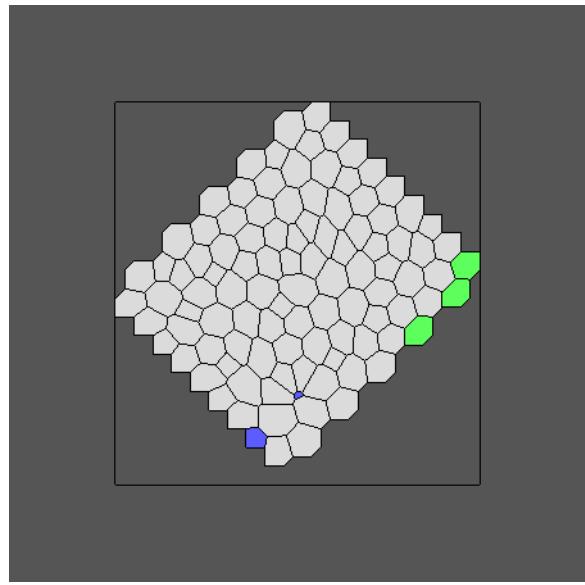


Figure 5.4: A Rotated Mesh for Error Analysis.

Bibliography

- [1] K. Bambardekar et. al. Direct Laser Manipulation Reveals the Mechanics of Cell Contacts In Vivo *PNAS* **112** 1416-1421.
- [2] Buchmann, A. Alber, M., Zartman, J. *Sizing it up: The mechanical feedback hypothesis of organ growth regulation.* Seminars in Cell and Developmental Biology, 2014.
- [3] Chaste Tutorial https://chaste.cs.ox.ac.uk/chaste/tutorials/release_2.1/UserTutorials.html
- [4] Drasdo, D. *Bucking Instabilities of One Layered Growing Tissues.* Physical Review Letters 84.
- [5] Durand, M., Stone, H. Relaxation Time of the Topological T1 process in a Two Dimensional Foam. arXiv.0
- [6] Epithelial Tissues http://www.botany.uwc.ac.za/sci_ed/grade10/mammal/epithelial.htm
- [7] Farhadifar, R., Roper, J., Algouy, B., Eaton, S., Jullcher, F. The Influence of Cell Mechanics, Cell-Cell Interactions, and Proliferation on Epithelial Packing. *Current Biology* **17** 2095-2104. (2007)
- [8] Fletcher, A. , Osterfield, M., Baker, R., Shvartsman, S. *Vertex Models of Epithelial Morphogenesis.* Biophysical Journal 106, June 2014.
- [9] Fletcher, A., Osborne, J., Maini, P, Gavaghan, D. *Implementing vertex dynamics models of cell populations in biology within a consistent computational framework.* Prog. Biophys. Mol. Biol. 113: 299 - 326.
- [10] Gibson, W., Gibson, M. *Cell Topology, Geometry, and Morphogenesis in Proliferating Epithelia.* Current Topics in Developmental Biology **89** 87-114. (2009)
- [11] T. Gillies and C. Cabernard. Cell Division and Orientation in Animals. *Current Biology* **21** 599-609

- [12] P. Grassia, C. Oguey, R. Saepitomi. Relaxation of the topological T1 process in a two-dimensional foam. *The European Physical Journal E* **35** (2012)
- [13] H. Honda. Description of Cellular Patterns by Dirichlet Domains: The Two-Dimensional Case. *Journal of Theoretical Biology* **72** 523-543. (1978)
- [14] H. Honda, H. Yamanaka, M. Dan-Sohkawa. A Computer Simulation of Geometrical Configurations During Cell Division. *Journal of Theoretical Biology* **106** 423-435. (1984)
- [15] Honda, H. Essence of Shape Formation in Animals. *Forma*, **27** S1-S8. (2012)
- [16] K. Kawasaki, T. Nagai, K. Nakashima. Vertex models for two-dimensional grain growth. *Philosophical Magazine Part B* **60** 399 - 421. (1989)
- [17] K. Liu, S. Ernst, V. Lecaudey, O. Ronneberger. Epithelial Rosette Detection in Microscopic Images.
- [18] Luebke, D. and Owens, J. *Intro to Parallel Programming* <https://www.udacity.com/course/cs344> 2013.
- [19] Marshall et. al. What Determines Cell Size? *BMC Biology* **10** (2012)
- [20] T. Nagai and H. Honda. Wound Healing Mechanism in Epithelial Tissues Cell Adhesion to Basal Lamina. *Proceedings of the 2006 WSEAS Int. Conf. on Cellular & Molecular Biology, Biophysics & Bioengineering***2006** (pp111-116)
- [21] Nagai, T. Honda, H. A dynamic model for the formation of epithelial tissues. *Philosophical Magazine, Pt. B*.**81**(2001)
- [22] R. Nagpal, A. Patel, M. Gibson. Epithelial Topology. *BioEssays* **30** 260-266. (2008)
- [23] K. Nakashima, T. Nagai, K. Kawasaki. Scaling Behavior of Two-Dimensional Domain Growth: Computer Simulation of Vertex Models. *Journal of Statistical Physics* **57** 759-787. (1989)
- [24] Ohlenbusch, H.M., Aste, T. Dubertret, B., Rivier, N. The Topological Structure of 2D Disordered Cellular Structures. arXiv.
- [25] S.Okuda et. al. Reversible Network Reconnection Model for Simulating Large Deformation in Dynamic Tissue Morphogenesis. *Biomech Model Mechanobiol* **12** 627-644 (2013)

- [26] <http://www.millerplace.k12.ny.us/webpages/lmiller/photos/636532/Epithelial%20TissueTypes.bmp>
- [27] Potential Energy Graph <http://images.tutorvista.com/content/work-energy-power/potential-energy-variation.gif>
- [28] K. Ragkousi and M. Gibson. Cell Division and the Maintenance of Epithelial Order. *Journal of Cell Biology* **207** 181-188
- [29] Restrepo, S., Zartman, J. , and Basler, K. *Coordination of Patterning and Growth by the Morphogen DPP* Current Biology 24, 245- 255
- [30] R. Thom. Topological Models in Biology. *Topology* **8** 313- 315 (1969)
- [31] Weaire, D. and Rivier, N. *Soap, Cells, and Statistics*
- [32] H. Yamanaka and H. Honda. A checkerboard pattern manifested by the oviduct epithelium of the Japanese Quail. *Int. J. Dev. Biol.* **34** 377-383.

Appendix A

Getting, Running, and Modifying the Code

A.1 GitHub

When you are working on a large project such as this one, it is a good idea to have some sort of version control system which tracks the changes you have made to your code, and to return to an earlier working version in case something gets terribly broken. Many people who do not know about version control will do just this, except they will save as' every couple of days. Unfortunately, this method is very space inefficient, as each time you 'save as', you save your entire project. Roughly speaking, git saves only the small changes you have made between versions. Git is a popular version control tool, and github is a popular place to store you files online.

The following instructions show you how to create and clone git repositories. The repository for *GrowFlesh* can be found at <https://github.com/JulianCienfuegos/NAGAIHONDAMODEL>.

Get Your Files on GitHub

- Go to github.com and sign up for an account
- Make a new repo on github
- cd to the directory of your project on your machine.
- `git init`
- `git add .`
- `git commit -m "some message"`

- git remote add origin YOUR URL HERE (This url is given to you from github when you make the repo.)
- git pull origin master
- git push origin master

Access And Modify These Files Somewhere Else

- Simply type git clone `url of repository here` into a terminal
- Work on project
- git push origin master, when done working.