End of DRUMS Overview– Cell Packing in Epithelial Tissue

Authors: Jeremiah Burden, Taylor L Cobb, Nina De La Torre, Nicole M Lacey
Graduate Mentor: Lord Eric W Schoen
Faculty Mentor: Dr. Sharon R Lubkin
July 28, 2022

I. Introduction

How organisms take shape over time define the anatomical structure and physiological processes of the organism. This process of taking shape from fertilization, known as morphogenesis, is influenced immensely by tissue shape. Cells, the building blocks of tissues, have their movement governed by both physical laws and biology. How cells pack inside tissue define tissue shape and therefore studying cell packing can give biologists insight into morphogenetic processes. In this project, we study how cells pack in the notochord of the zebrafish based on the surface tensions, volumes, and pressures of the cells. Our end goal is to create a robust model of epithelial tissue with explanatory power.

Before we model the notochord, we look at alternate methods of how cell packing can be modeled. The simplest, a particle model, considers the centroids of the cells as particles and treats the cells as points. Each particle applies a force to all other particles as a function of how far apart they are and other parameters. This model has value because it is simple and easy to see, but its simplicity limits this method's use. Continuing on to more advanced models, a popular method to model cells is by using a vertex model. Vertex models can be implemented in 2D or 3D, with the cells being represented as polygons or polyhedra, respectively. This model still uses points, but the points are no longer representative of the cells; they represent the vertices that bound the edges which bound the faces of the polygons and the faces bound the bodies of the polyhedra in 3D. These models have some limitations, such as treating cells as living in 2D (2D model) and assuming every interface between cells is a straight line (2D and 3D model). Thus, one improvement on the vertex model is the foam model which treats the cells as a foam, similar to the froth on a cold beverage. The foam model can be 2D or 3D, with the cells being bounded by curves or surfaces, respectively. This increases the accuracy of the vertex model and also increases the geometric complexity. These three models– particle, vertex, and foam– make up our tool kit for this project.

II. Particle Model

a. Methods

The particle model is a great stepping stone into the other models because this method is simple. As aforementioned, it treats each cell as a particle, the cell's centroid. The goal of all of these models is to display the minimal energy state, or the configuration that minimizes the total energy. In the particle model, an "equilibrium distance" is defined, which is the "preferred" distance that cells wish to be apart from one another. In nature, this value would be dependent on the volume and different radii of the cells. Once an equilibrium distance is defined, cells will converge to that distance. If the cells are outside the equilibrium distance, i.e. too far, a negative force will be applied, bringing them towards the equilibrium distance. If the cells are inside the equilibrium distance, i.e. too close, a positive force will be applied, spreading them out to approach said distance. For the 1D particle model, we set the equilibrium distance to be an arbitrary constant. For the 2D particle model, each particle is assigned a radius. The equilibrium distance between two cells is then calculated as the sum of their radii. This makes sense, because in nature cells would not be able to overlap. Now that the equilibrium distance is defined, the forces on each particle can be defined.

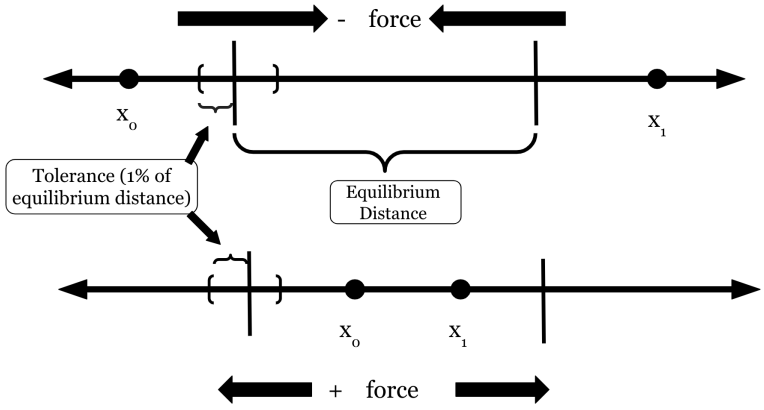


Figure 2.1
1D particle model showing directional forces acting on
the particles seeking their equilibrium distance

We assume each particle applies a force on every other particle. For 1D this force is a linear function

$$F = -kr_0$$

where F is the force applied to the particle, k is the damping constant (a parameter for the system) and r_0 is the distance between the particle and the equilibrium distance. Therefore, as the particle approaches the equilibrium distance, the force applied to the particle approaches zero.

The minimal energy state is found when each particle is sufficiently close to their preferred distance. In our model, we defined "sufficiently close" to being within 1% of its preferred distance. In 2D, we use the Lennard-Jones Potential function which are equations that govern particle motion. If the particle is outside the equilibrium distance, the force acting on that particle is

$$F = 4a[(r_0/r)^{12} - (r_0/r)^6]$$

where a is the maximum force of attraction, r_0 is the distance between the particle and its equilibrium position, and r is the distance between the particles. If the particle is inside the equilibrium distance, the force acting on that particle is

$$F = -m(r^2/r_0^2) + m$$

where r_0 and r are the same as before and m is the maximum force of repulsion. So, the net force on the particle is the sum of all of the forces each other particle is putting onto it. Over time, the net force acting on each particle approaches zero. We define a tolerance force relatively close to zero. When the net force on each particle is within that tolerance, it causes the program to terminate because it is in its minimal energy state. Thus, for the 2D model, the minimal energy state is defined to be when the net force on each particle is relatively close to zero.

b. Examples in 1D

The reason we spend time making the 1D model is so that we can better understand the principles that governs cell packing. To start our model, we declare the number of cells (points) and randomize each of their locations on our 1D line. The program then runs until each particle is within 1% of its preferred distance. Since our force is linear and dependent on the distance away from the equilibrium distance, as the cell approaches the distance, the magnitude of the force will grow weaker and therefore so will the magnitude of the velocity.

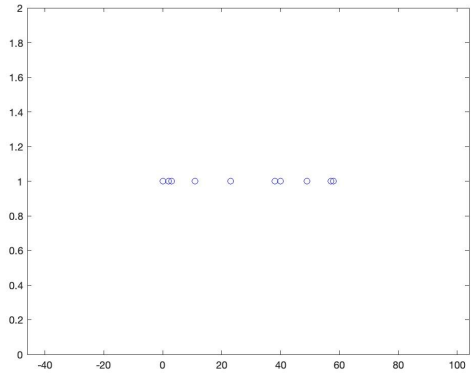


Figure 2.2 a
10 particles randomly positioned on a line before packing

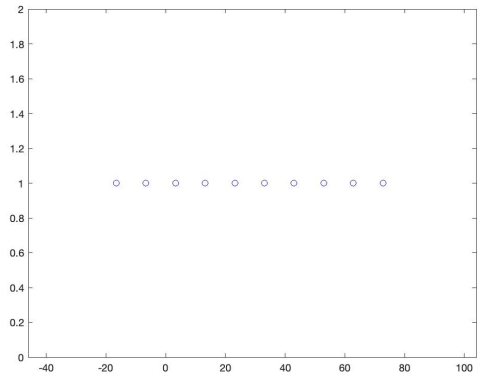


Figure 2.2 b
10 particles after packing

c. Examples in 2D

The 2D particle model is still a simple model but is more accurate than the 1D model and can therefore be expanded to have uses beyond just cell movement, like cell growth and cell division. Once again, to start this model we declare the number of particles and randomize their positions. Since it is a 2 dimensional model, each particle will have an x and y position. The program is run until each particle is within the tolerated net force.

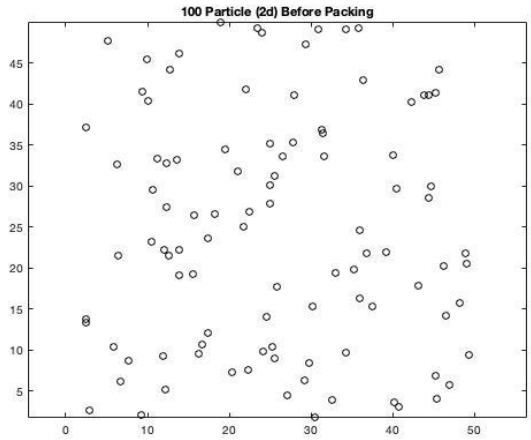


Figure 2.3 a
100 particles randomly located
on a plane before packing

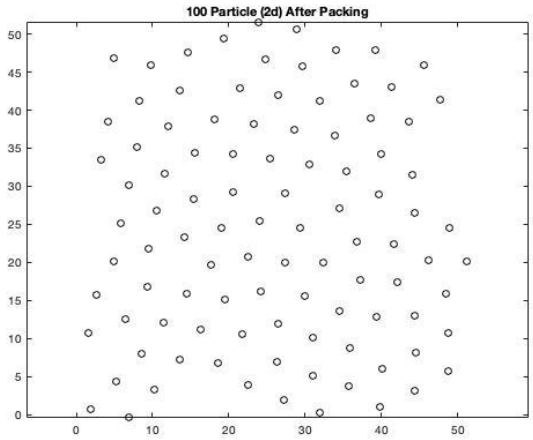


Figure 2.3 b
100 particles on a
plane after packing

In this configuration it is easy to see the even spacing and hexagonal packing of the cells. We then expand this model to incorporate cell growth and cell division. Each cell is given a random starting radius as well as a random value for cell growth. Once a cell reaches a specified area, it will divide into two cells of equal size so that their sum is the original area .

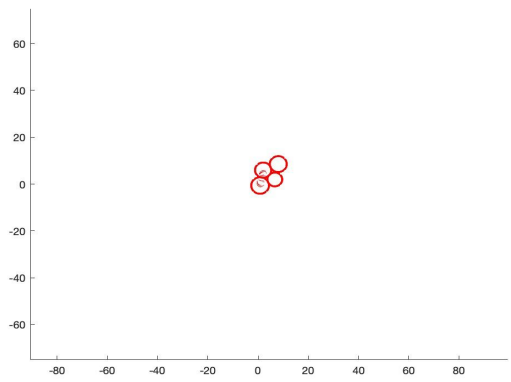


Figure 2.4 a
4 cells with randomized sizes and
growth rates before cell division

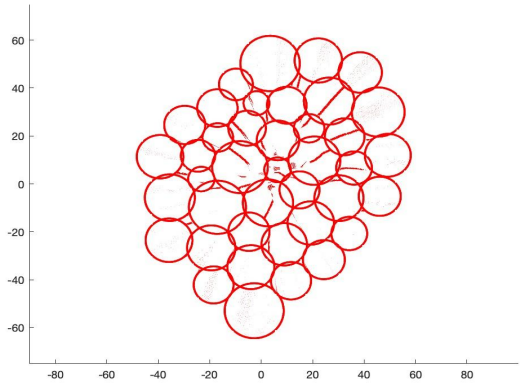


Figure 2.4 b
Cells after cell division and growth

In the figure on the right we see circles packed in tightly so that they are overlapping; this overlapping is because the circles can not flatten their edges to be perfect hexagons. Another visualization that makes the 2D model more meaningful is using a Voronoi tessellation to visualize one way cells might partition the plane. When we consider each colored polygon as a cell, we notice that they often create hexagonal shapes, but we also see pentagons and even a few heptagons.

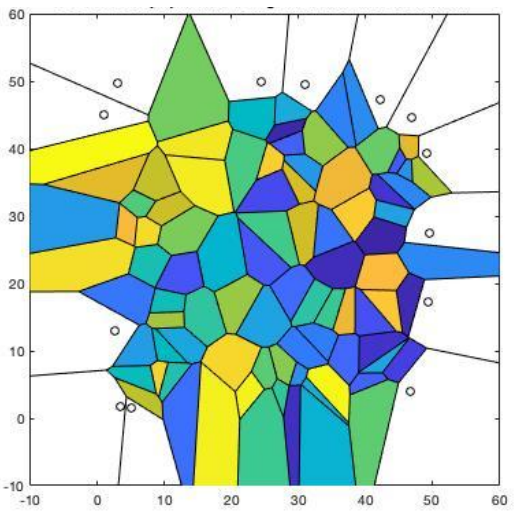


Figure 2.5 a
2D particle model with voronoi tessellations before packing

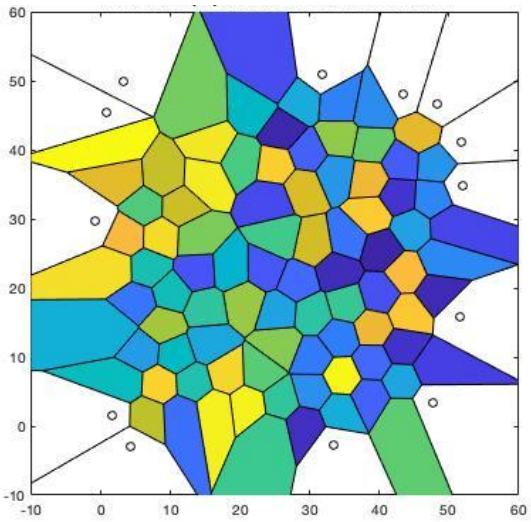


Figure 2.5 b
2D particle model with voronoi tessellations after packing

d. Conclusion

Overall, the particle model is simple and easy but not accurate or realistic. Because of the simplicity we can see easy results that may give us insight into the construction of other models, but particle models alone are not enough. Another benefit of the particle model is the simplicity at which we can model cell growth and cell division, which is not always the case.

III. Foam Model

a. Methods

This next type of model uses vertices, curved edges, and curved faces to represent cells. This is different from the aforementioned particle models in that they use volume, pressure, and surface tension to create changes in cell structure. We use this model because it is a more robust

representation of how cells exist in nature. For example, cells and foam are both governed by surface tension and we use cell-to-cell and cell-to-medium tensions in many of these models. In 2D foam models, pressure, curvature, and line tension can be set, and we use reasonable constraints to make our models. In 3D, however, the Laplace-Young condition governs how pressures, mean curvatures, and surface tension relate to each other. Also in these 3D models, the smooth surfaces are created by a mesh, made of equilateral triangles. This is then iterated and refined to seek the minimal energy configuration for the shape started with.

We use The Surface Evolver to create these models. This program works by first setting the desired topology. For example, if we want to create a sphere, we would initialize it as a cube (Figure 3.1a). Every iteration adds triangles or forms the mesh to minimize energy. Eventually, it becomes a sphere (Figure 3.1b).

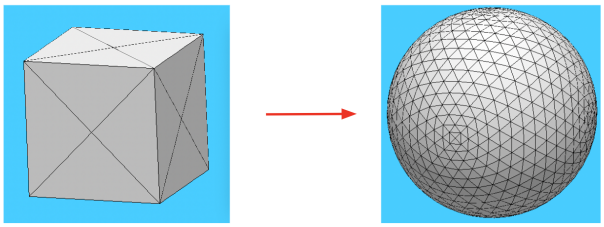


Figure 3.1a

Figure 3.1b

We consider Plateau's Laws, which define the structure of soap films. Following these laws means that all surfaces must be smooth, there are only 3–way junctions between vertices, and vertices must meet at tetrahedral angles (109.47°). In addition to this, we ensure mesh quality by checking for equilateral triangles, symmetry along the shape, and smoothness instead of scrunching. In Figure 3.2, we see many four way junctions and triangles that are not equilateral. This means that the mesh must be fixed with 'vertex popping'. This applies a T1 transformation to the affected vertices in Figure 3.3.

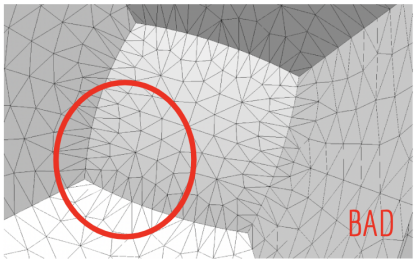


Figure 3.2

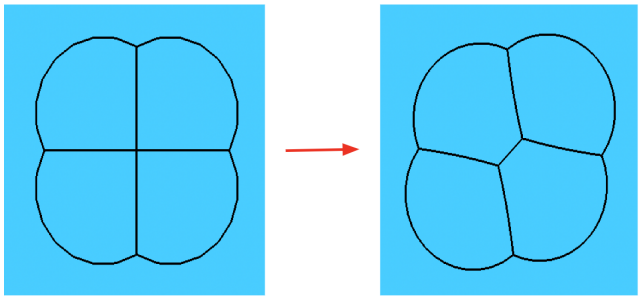


Figure 3.3
T1 cell transformation

Additionally, we must check for scrunching. We see this in Figure 3.4c, which shows a torus that was ineffectively developed and now the inside face folds over itself. This is fixed with vertex averaging. The bad torus and the smooth torus (Figure 3.4b) are both developed from the

same starting configuration (Figure 3.4a), but the smooth one uses vertex averaging. It is a more reasonable model.

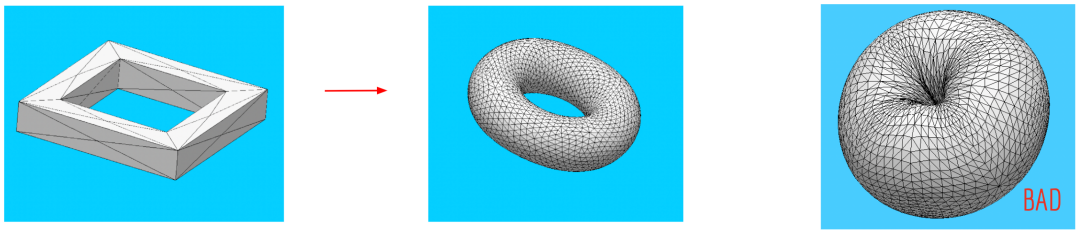


Figure 3.4 a
Starting configuration for a torus

3.4 b
Properly developed torus

3.4 c
A model developed without vertex popping

b. Examples in 3D

Next, we aim to model biological processes as a foam. This will give us a more accurate representation of how cells work because the cells will be portrayed as bodies instead of just faces. We choose to do the frog egg epiboly as it is intuitive to implement. This process is where epithelial cells spread to cover the yolk of a frog egg, and is an early part of development for the frog (Figure 3.5) (Swift).

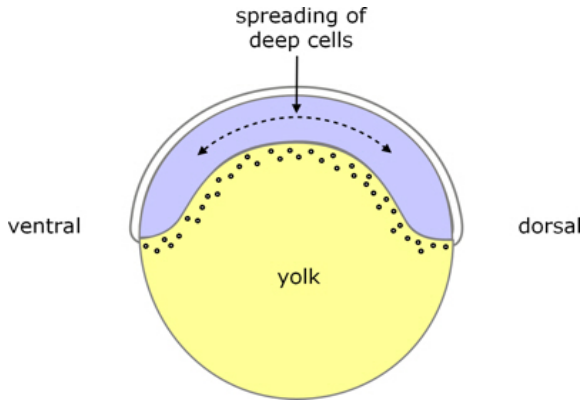
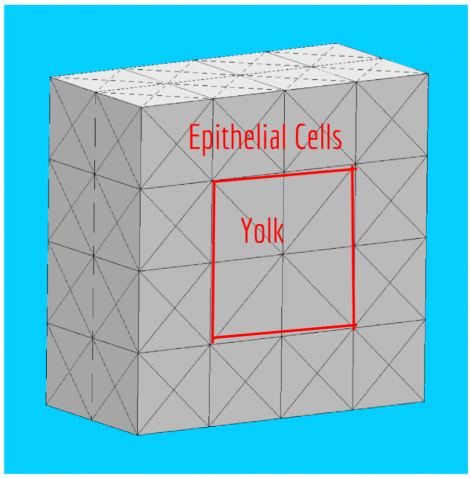


Figure 3.5

To make this model, we start with a ring formation. This means there there is a large cell in the middle, which is the yolk, surrounded by a ring of smaller cells, which are the epithelial cells (Figure 3.6a). There are some inconsistencies with our model and real life. First, the actual epiboly creates a cap of cells that will spread over the yolk, instead of starting already dispersed. Additionally, these cells are too large compared to the yolk size– where there should be thousands of cells, we only have twenty four surrounding the yolk. Keeping these in mind, the resulting image is in Figure 3.6 b.



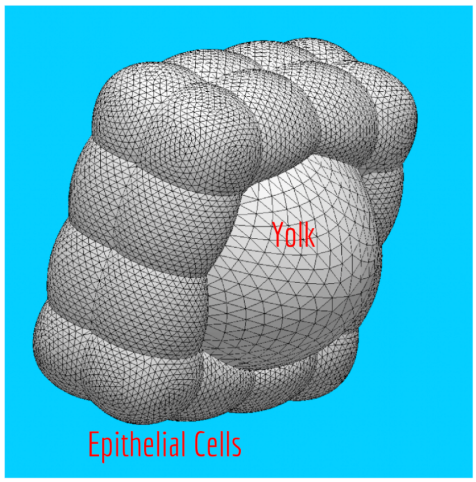
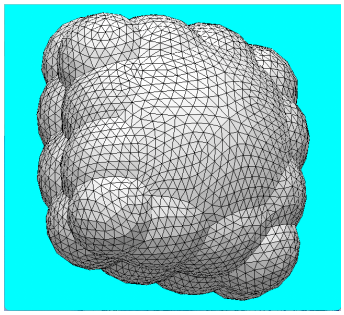
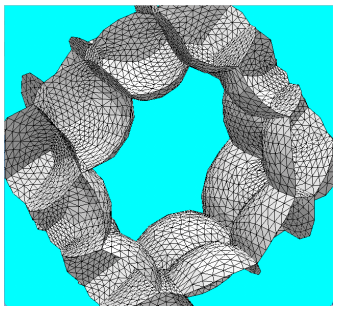


Figure 3.6a

Figure 3.6b

This is the first result. However there are some more unrealistic features. First, the ring of cells on the outside meet at four way junctions all the way around. This is not something that happens in nature and so it will need to be fixed with vertex popping. We also note that the tension between the yolk and epithelial cells is too high. We adjusted this to create a smoother transition at the yolk and epithelial cell junction. Below is the final product (Figure 3.7a), as well as a view of the internal faces (3.7b).





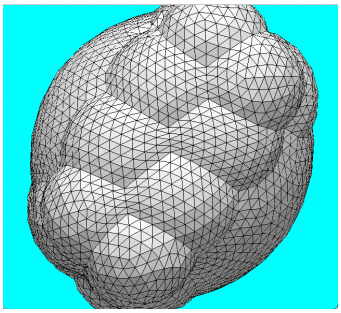


Figure 3.7a

Figure 3.7b

Figure 3.7c

Figure 3.7c is what happens to the epithelial cells when vertex popping is used. Instead of symmetric four way junctions, it creates a zig-zag down the sides. This is exactly what happens in nature, so this part of our model is accurate. Another side effect of the vertex popping means that cells will be in randomized positions– this final shape is not uniform. This is also ideal because no frog egg epiboly in nature is perfectly uniform.

c. Conclusions

Using foam for modeling cells has many strengths, and some limitations. One of the main conveniences is that we only need simple starting conditions. For example, we set vertices, edges, faces, and bodies, and from there The Surface Evolver will refine and shape the mesh with basic commands. Another advantage is that we can see the outside and inside geometry of the models. Other benefits include using curved edges, which are more accurate biologically, and being able to set the tensions on these boundaries.

The few limitations that The Surface Evolver has is that it struggles to model cell movement. This includes the inability to do cell splitting and the actual process of epiboly where cells move around the outside of the yolk. So, we are limited in that we can only model how cells change shape, not how they move around each other.

IV. Detailed Study: Zebrafish notochords in 2D

a. Introduction

In order to apply the principles of how cells pack into epithelial tissue, we choose to examine the notochord sheath of the zebrafish. These notochords offer valuable insight into morphogenesis because it is a simple system to analyze cell behavior. For this reason, substantial research has already been conducted on the zebrafish. The notochord prepatterns the spinal cord of the zebrafish and is prevalent in the early embryonic stages of development. Although there are several structures that make up the notochord, we specifically focus on the interior and sheath cells. When examining its cross section, the interior is the large vacuolated cell that is surrounded by smaller epithelial, or sheath, cells. The interior has different stacking configurations, mainly either bamboo or staircase. In the bamboo packing, there is one interior cell in each cross section. However, in staircase packing, there are two interior cells.

Other notochord models provide guidance in our examination. More specifically, Sorrell models an infinitely long sheath to show its eccentricity, how much a conic section deviates from a perfect circle (2022). From her work, we see that a bamboo-packed sheath maintains a perfect circle in its cross section, with an eccentricity of zero, while a staircase packed sheath results in an eccentricity between zero and one. Others have also noted what tension values can visualize reasonable notochord models (Brezavšček 2012).

In our research, we aim to build on what work has been done by observing ranges of tensions that yield reasonable model outputs. We also seek to observe the difference in constraining the volume of the cells compared to constraining their pressure as we examine the optimal packing state of the notochord's cross section. Through this, we can model a simple system that contributes to the overall understanding of morphogenesis.

b. Methods

To model this system, we choose a 2D string model because of its simplicity to implement and its comparability to previous work. We use The Surface Evolver to visualize the

boundaries between cells as curved edges all in one plane. Because we use a string model, it must abide by Plateau's Laws and the Young-Dupré condition. Plateau's Laws describe how soap films must behave. In two dimensions, they state that soap films are made of smooth edges, their mean curvature is constant on any point of the same soap film, there are no four way junctions of edges converging to a single vertex, and that the angles meet at 120° angles if the tensions are equal. These laws govern how our model must behave in order to compare to real biological systems. The Young-Dupré condition relates the contact angle of a liquid on a solid surface with the interfacial tensions acting on that angle. The Young-Dupré equation is

$$cos\theta_Y = (\gamma_{sv} - \gamma_{sl})/\gamma_{lv}$$

where γ_{lv} is the liquid-vapor interfacial tension, γ_{sl} is the liquid-solid interfacial tension, γ_{sv} is the solid-vapor interfacial tension, and θ_Y is the equilibrium contact angle. This equation also bounds what are realistic tensions for our model. Through these constraints, Surface Evolver minimizes the energy of our string model by moving the vertices in the direction that minimizes the energy based on the gradients at each vertex.

c. Model Assumptions

To create our model in the Evolver, we import a text file that defines the vertices, edges, faces, and bodies. For our specific program, we create two concentric nonagons. This initial geometry establishes nine sheath cells surrounding either one or two interior cells. We make this assumption because we approximated nine sheath cells from the microscopy given by Bevilacqua (2019).

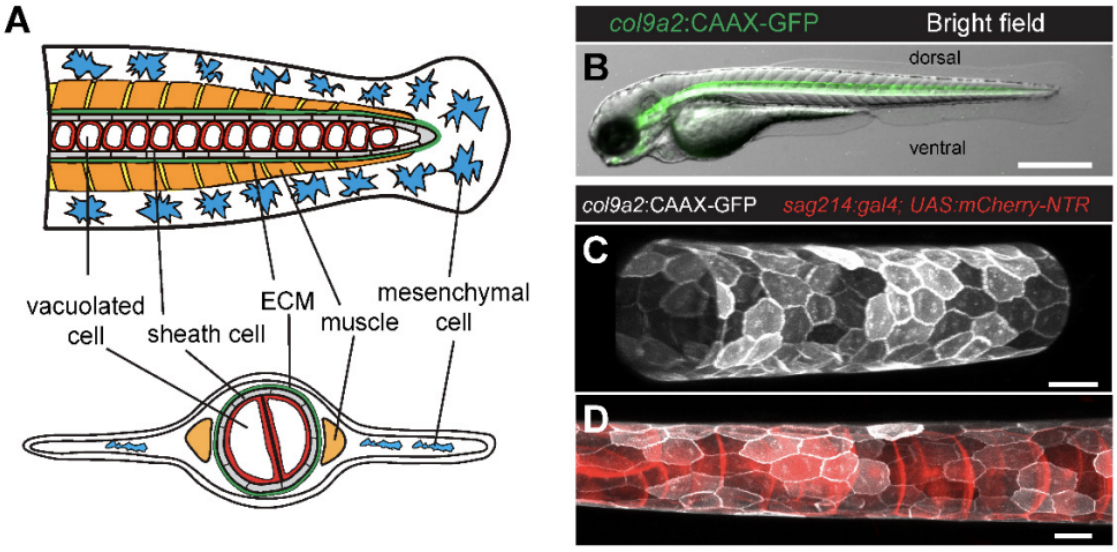


Figure 4.1a shows a diagram of the notochord sheath. Figure 4.1bcd show microscopy images from Bevilacqua 2019. We see nine surrounding cells in *b* and we seek to examine a cross section of the long tube.

We also initialize the key parameters that we examine in our study. These include five tension values and the pressure or volume constraints. I stands for the interior cell, S for the sheath cell, and M for the medium in the space surrounding the notochord. We name the line tension, energy per unit length, using the first initial of the interface type. Because of this, γ_{SM} is the edge tension between sheath and medium, γ_{Si} is the edge tension between sheath and interior, γ_{SS} is the edge tension between sheath cells, and γ_{ii} is the edge tension between interior cells (not pictured). We initialize these tensions and what is being constrained– either pressure or volume– for each type of cell.

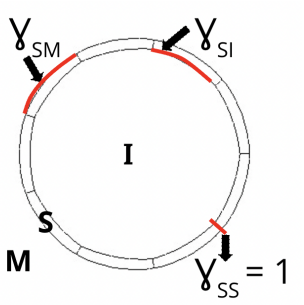


Figure 4.2 is a labelled figure of our model with the tensions and type of area clearly defined.

Because there are four tensions we intend to investigate, we utilize three tension ratios constructed by the already defined values. We initialize **α**, **β**, and **Γ** as

$$\boldsymbol{\alpha} = \gamma_{SM} / \gamma_{SS}$$
$$\boldsymbol{\beta} = \gamma_{Si} / \gamma_{SS}$$
$$\boldsymbol{\Gamma} = (\gamma_{SM}+\gamma_{Si}) / \gamma_{ii}.$$

This allows us to examine three tensions in 2D bifurcation diagrams which makes the visualization of our results simple. We choose small values of the tensions jumping multiplicatively so that we can see a large parameter space.

In our study, we fix both the sheath and interior cell pressure or volume independently of each other so that we can examine the influence that each constraint has on the resulting configuration. This results in four different cases to examine. Both the interior and sheath cells have two different constraints, either pressure or volume, so we see there are 4 combinations that can be made (interior fixed volume and sheath fixed volume, interior fixed pressure and sheath fixed volume, etc.). We assume that the interior cell has a volume of π, with a radius of one, and the sheath cell width is 10% of the interior cell radius. Note, we are modeling a 3D system in 2D, so we are constraining volume, which appears to be the area in our model outputs. Therefore, we calculate the volume of each sheath cell to be constrained to 0.0723 when the sheath cell volume is fixed. From this initial fixed volume case, we record the pressure values the model outputs and fix the pressure to these values in the constrained pressure cases.

When there are two interior cells, this adds another significant edge tension, the tension between the two interior cells. Because of this, the parameter space grows larger and more complicated, so certain simplifications to our system are made. Firstly, we note that γ_{SM}, γ_{Si}, and γ_{ii} should all be greater than 2 for a proper converged image (no straight edges). Second, we set γ_{SS} to be two-thirds of γ_{SM} to ensure the sheath cells that connect to the interior-interior interface are well behaved and do not cross over other edges. Since γ_{SM} and γ_{Si} both do similar things to the system (squeezing the body as a whole) to make things simpler, we assume γ_{SM} is half of γ_{Si}. These assumptions make our system more manageable to work with but still thorough.

d. Results

i. One Interior Cell

To begin our analysis, we investigate the relationship between pressure and tension with one interior cell. We only vary **α** and **β** since **Γ** is nonsensical without a γ_{ii} value.

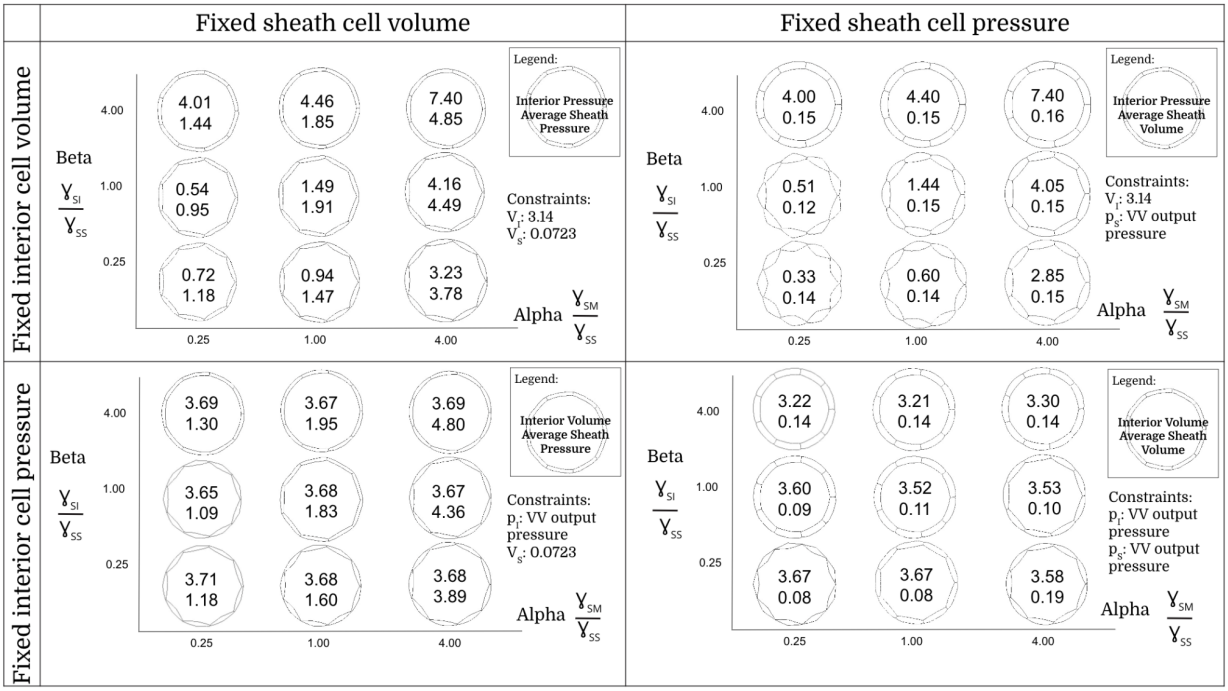


Figure 4.3 shows our results with one interior cell with varying constraints and tension parameters. The columns correspond to the sheath cell constraint and the rows correspond with the interior cell constraint. For each case, Alpha is on the x axis and Beta is on the y axis with 3 different values for each. The values that the model outputs are located in the model. They correspond to the value that was not being constrained. As Alpha increases, the configurations are similar but the volume/pressure values vary, while Beta more strongly influences the shape as it is increased.

ii. Two Interior Cells

As we move our focus into two interior cells, we focus on the fixed sheath and interior cell volume scenario to narrow our attention. More specifically, we seek to observe the possible tension ranges that create reasonable outputs. Through our preliminary simulations, we discover that tensions surrounding **Γ** = 2 provide the most insight because the interior to interior interface changes the shape of the intersecting sheath cells.

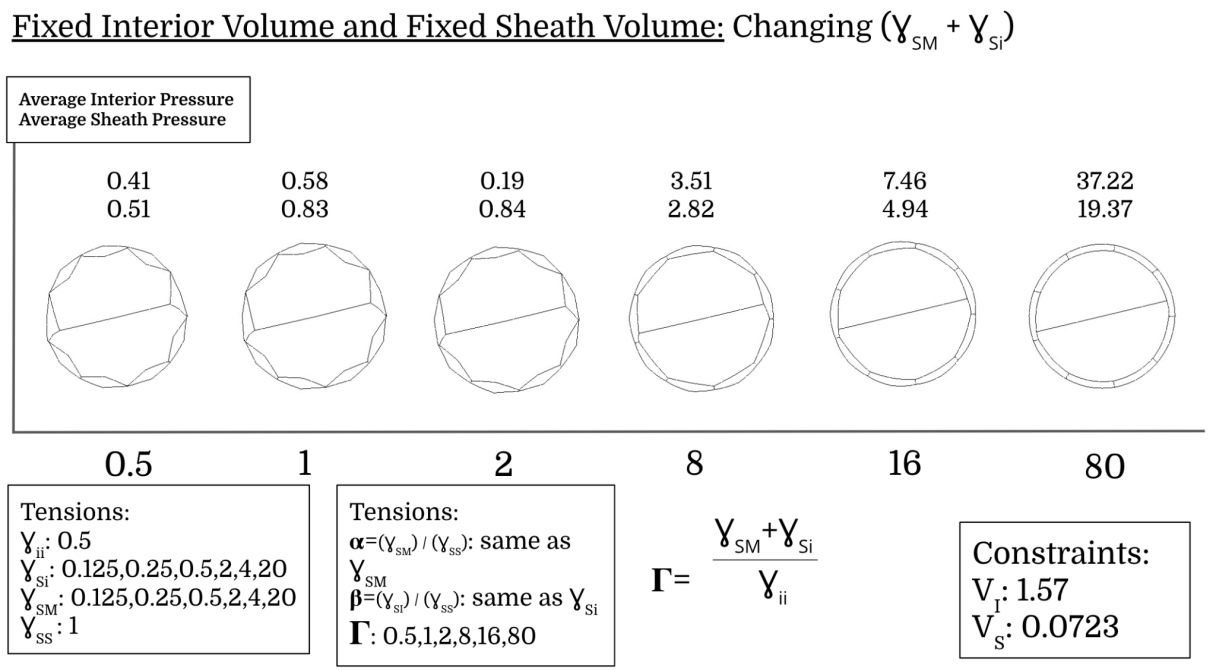


Figure 4.4 shows our initial trials with two interior cells. This figure varies the numerator of Γ while setting the interior to interior tension to 0.5. We observe poor configurations with pointed edges at low values of Γ.

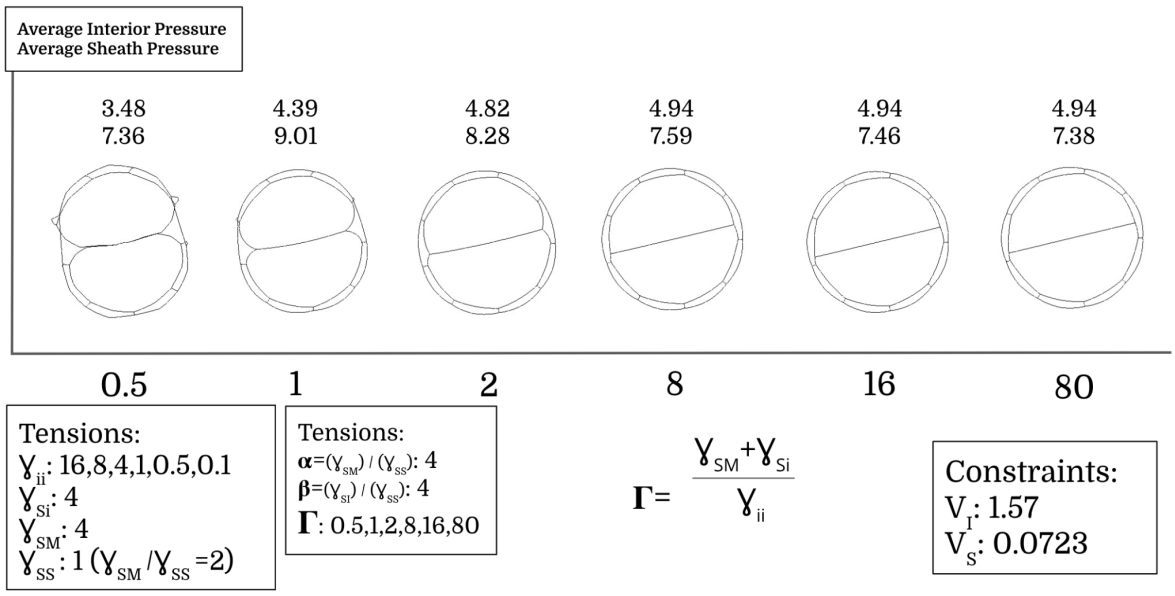


Figure 4.5 shows two interior cells with a varying denominator of Γ. We see the crossing of edges for low values of Γ, which suggests it is not a realistic tension space.

Because of our initial observations, we look more closely at smaller Γ values while fixing the numerator or denominator to examine the notochord shape's relationship to each of the tensions.

Fixed Interior Volume and Fixed Sheath Volume: Changing γ_{si}, γ_{sm} with fixed γ_{ii}

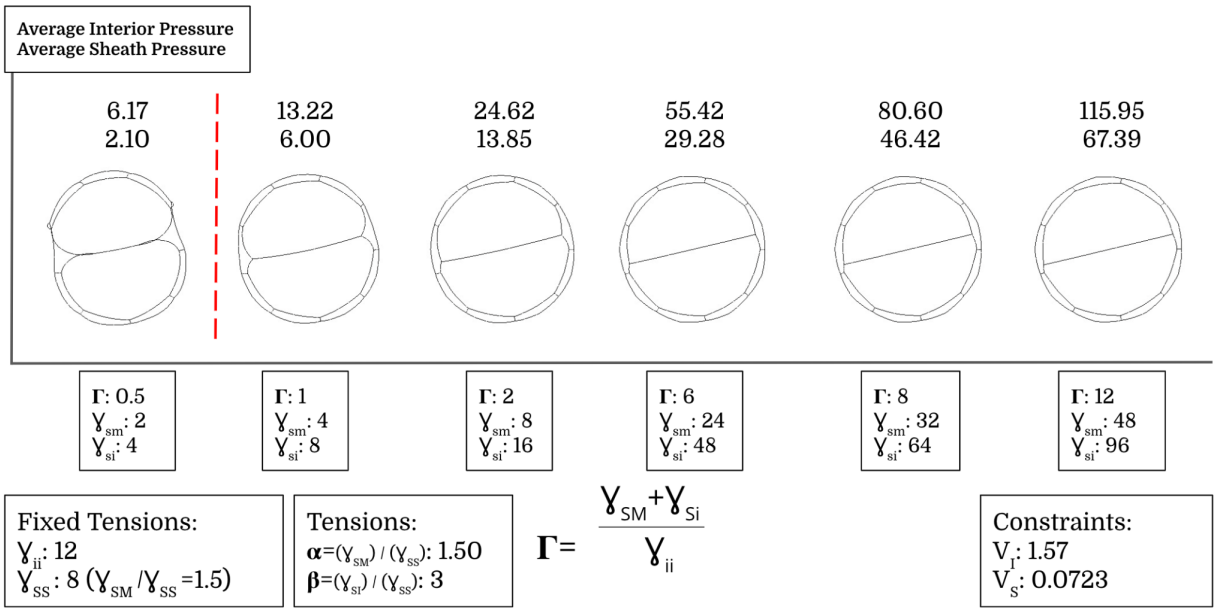


Figure 4.6 shows our follow up simulations with varying the numerator of Γ. We fix the tensions according to what we observe as viable tensions and see better configurations. The average interior and sheath cell pressures are also listed above the model figures.

Fixed Interior Volume and Fixed Sheath Volume: Changing γ_{ii} with fixed γ_{si}, γ_{sm}

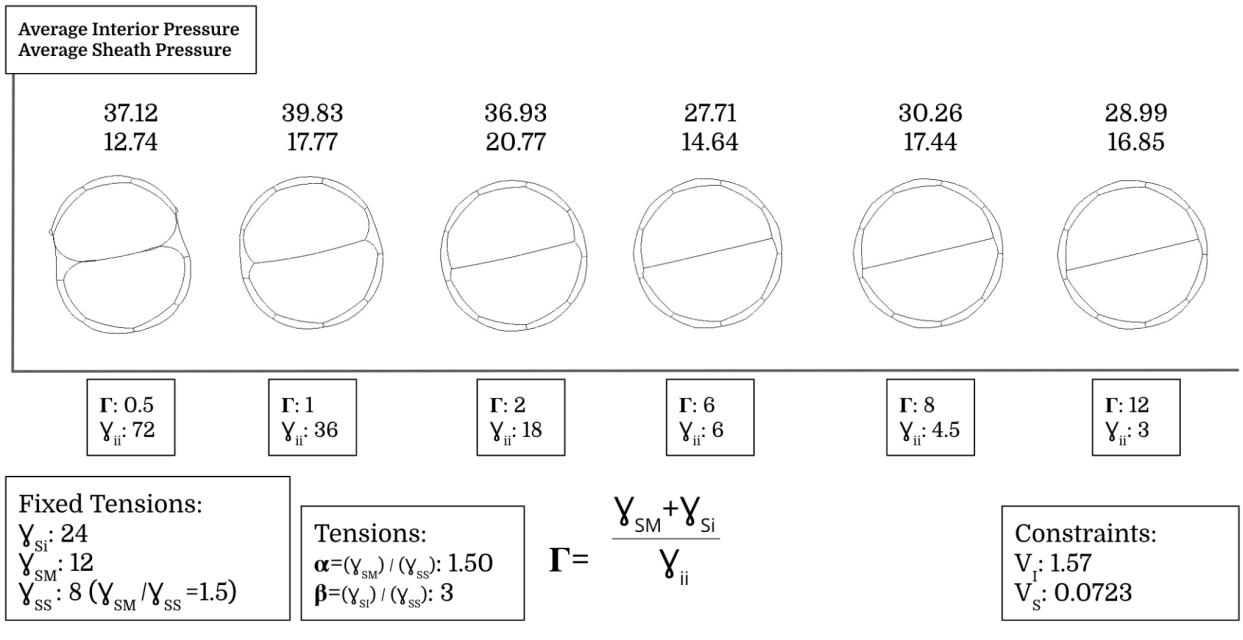


Figure 4.7 shows our follow up simulations with varying the denominator of Γ. We fix the tensions according to the same observations as before and see similar results.

Red: Interior Cells
Blue: Sheath cells
Yellow: Sheath cells directly adjacent to where interior cells touch

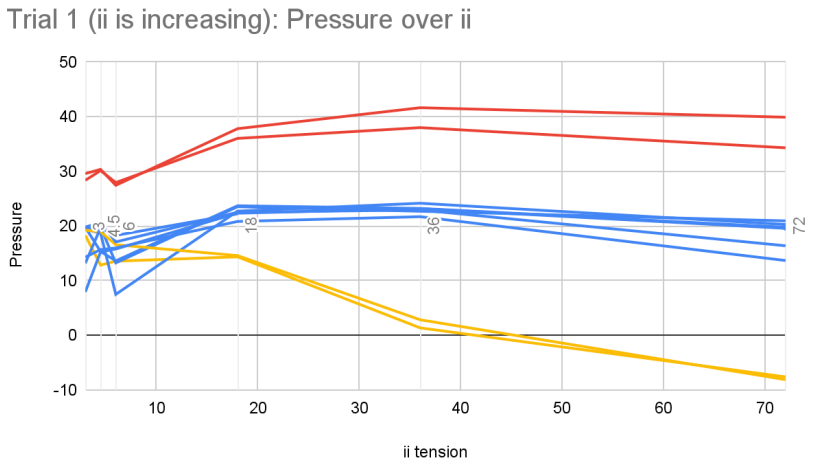


Figure 4.8
This shows trial 2, where the x-axis is the tensions between the interior cell to interior cell, and the y axis is the pressure of all the cells.

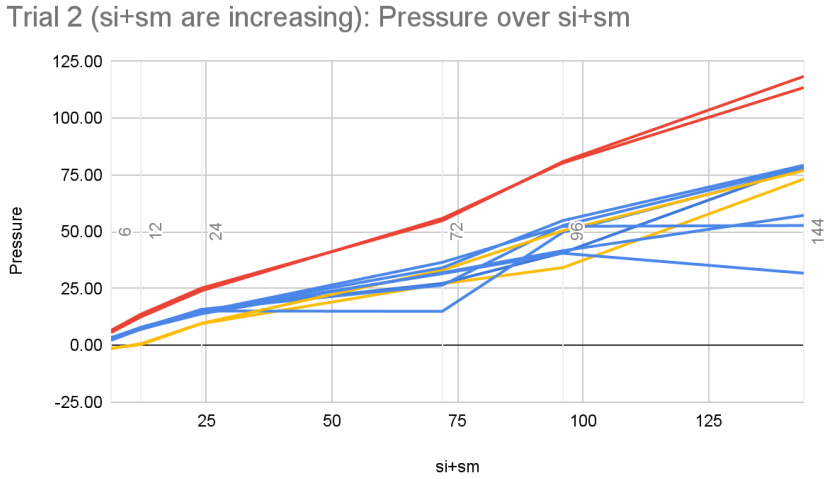


Figure 4.9
This shows trial 2, where the x-axis is the tensions between the sheath to interior plus sheath to medium, and the y axis is the pressure of all the cells.

e. Conclusions

i. Scientific Conclusions

From the one interior cell outputs, we see that **β** must be greater than one. We draw this conclusion because when **β** = 1 or **β** = 0.25 in our study, we see that the interior to sheath boundary is not convex, which we expect based on the microscopy images. We anticipate that a smooth interior and exterior shape is more accurate than protruding sheath edges. Within this relationship where **β** > 1, we see that to have a convex interior, the interior pressure must be greater than the sheath pressure, which must be greater than the medium pressure ($p_i > p_S > p_M$).

In these single vacuolated cell investigations, we observe little notable differences in the output images between constraining pressure and constraining volume. However, we do note that there are discrepancies in the output volume values compared to the values we expect from our fixed interior and sheath volume case. This could suggest that there may be multiple minimal energy configurations, though more research is needed to draw any clear conclusions.

In our two interior cell studies, our research is ongoing so deductions are not concrete. That being said, in our initial trials, we observe that γ_{SM}, γ_{Si}, and γ_{ii} should all be greater than 2 so that there are not any straight edges. This result guides our future study as it provides additional assumptions and a more focused parameter space.

From our later trials, we observe that as γ_{Si} increases, p_s tends to decrease while p_i tends to increase. In addition, we see that as γ_{SM} increases, p_s tends to increase while p_i tends to increase less significantly. These results, though, are preliminary. In our graphs, we can see that the pressure of the interior cells directly adjacent to the interior to interior boundary is starkly smaller than the other sheath cells in the case where γ_{ii} varies.

ii. Strengths

There are many strengths to this model construction such as the variables we are able to manipulate and the connections to more accurate representations. As a foam model, our examination of the notochord sheath can manipulate many different parameters, like the tensions, pressures, and volumes, with relative ease. Because we are able to fix either the pressure or the volume, we can analyze those constraints as it relates to the model configuration. In addition, our 2D model can utilize curved boundaries with tensions acting on those edges, which is more accurate to actual natural occurrences.

iii. Limitations

Although we are able to learn a lot from our model, it has limitations. We are examining figures in 2D, but the notochord sheath is a 3D system. We also have not found a way to add other interactions with cells, such as cell splitting, which we could do in other more simple models.

f. Remaining Tasks

Going forward, we intend to analyze the relationship of eccentricity and the tension values for our two interior cell models. Eccentricity is a topic of interest because the shape of the model is no longer perfectly circular when we introduce an additional vacuolated cell. We intend to examine eccentricity as a function of the previously mentioned tensions. We also plan to observe if we can create a biological phenomenon where a sheath cell migrates into the center to become an interior cell when an interior cell is ablated.

References

Bevilacqua et. al, 2019 Bevilacqua, Carlo, et al. "Imaging mechanical properties of sub-micron ECM in live zebrafish using Brillouin microscopy." *Biomedical Optics Express* 10.3 (2019): 1420-1431.

Hočevar Brezavšček A, Rauzi M, Leptin M, Ziherl P. A model of epithelial invagination driven by collective mechanics of identical cells. Biophys J. 2012 Sep 5;103(5):1069-77. doi: 10.1016/j.bpj.2012.07.018. PMID: 23009857; PMCID: PMC3433605.

Sorrell EL, Lubkin SR. Bubble packing, eccentricity, and notochord development. Cells Dev. 2022 Mar;169:203753. doi: 10.1016/j.cdev.2021.203753. Epub 2021 Oct 30. PMID: 34728430.

Swift, David. “Evolution under the Microscope.” Embryonic Development of Teleost Fish, https://www.evolutionunderthemicroscope.com/embryo04.html.

Acknowledgements

We would like to thank Evan J Curcio for help with The Surface Evolver and providing us code to make text files for our notochord sheath model.

Symbols

Symbol	Definition
k	An arbitrary damping constant
F	A force applied to a particle
r_0	The distance between a particle and its equilibrium distance
r	The distance between particles

a	The maximum force of attraction between particles
m	The maximum force of repulsion between particles
θ_Y	The angle a liquid contacts a surface
γ_{sv}	The solid-vapor interfacial tension
γ_{lv}	The liquid-vapor interfacial tension
γ_{sl}	The liquid-solid interfacial tension
γ_{SM}	The line tension of the sheath to medium interface
γ_{Si}	The line tension of the sheath to interior interface
γ_{ii}	The line tension of the interior to interior interface
γ_{SS}	The line tension of the sheath to sheath interface
α	The dimensionless ratio $\gamma_{SM} / \gamma_{SS}$
β	The dimensionless ratio $\gamma_{Si} / \gamma_{SS}$
Γ	The dimensionless ratio $(\gamma_{SM}+\gamma_{Si}) / \gamma_{ii}$
V_I	The volume of the interior cell(s)
V_S	The volume of the sheath cells

Appendices

2D Particle Model (with cell growth and division) Matlab code example:

```
%% n Particle in 2D with cell splitting
clc
cla
clear


%% Parameters
ncount = 4;
```

```
% How many particles are in the system. (parameter)

maxn = 10*ncount;
% Maximum amount of cells in the model.

deltat = .01;
% The change in time after each iteration, it will be used to find the
distance that the particles move together. (parameter) (T)

ma = 1;
% The maximum attraction between the particles. Arbitrary constant.
(parameter) (nanonewtons)

mr = 80;
% The maximum repulsion between the particles. Arbitrary constant.
(parameter) (nanonewtons)
% Higher numbers here prevent cells from stacking on each other.

% r = (b-a).*rand(1000,1) + a;
splitsize = 4;

%% Initializing arrays

position = zeros(maxn,2);
% This is the randomized initial positions where each row is one particle
and the first column is the x position and the second column is the y
position.
% We change the range here to make the particle either go closer together
% or further apart.

eq_dis = zeros(maxn,maxn);
% This is the ideal distance that each particle is converging to.
(parameter) Equilibrium Distance. (microns)

xdistance = zeros(maxn,maxn);
ydistance = zeros(maxn,maxn);
% This initializes the x and y distance arrays between particles.

distance = zeros(maxn,maxn);
% This initializes the array to store the distances between particles (not
% component-wise).

ljscale = zeros(maxn,maxn);
% This initializes the array that holds the magnitude scalar of the
forces.

xforce = zeros(maxn,maxn);
```

```
yforce = zeros(maxn,maxn);
% This initalizes the force arrays for x and y.

radius = zeros(maxn, 1);

drdt = zeros(maxn,1);


%% Populating arrays

for i = 1:ncount
    for j=1:ncount
        if maxn > ncount
            position(i,2) = 1+ (10-1).*rand();
            position(i,1) = 1+ (10-1).*rand();
            radius(i,1) = 1+ (5-1).*rand();
            drdt(i,1) = 1 + (4-1).*rand();
        end

        % This initializes the distance arrays.
        if i~=j % The distance between a particle and itself is 0. The
array is initially zeros.
            xdistance(i,j) = position(j,1)-position(i,1); % This finds the
x distance between each particle and stores them in an array.
            ydistance(i,j) = position(j,2)-position(i,2); % This finds the
y distance between each particle and stores them in an array
            eq_dis(i,j) = radius(i) + radius(j);
        end

        % This populates the distance array using the pythagorean theorem.
        distance(i,j) =
sqrt((position(i,1)-position(j,1))^2+(position(i,2)-position(j,2))^2);



        % This populates the magnitude scalar array using the Lennard
Jones
        % potential.
        if i==j
            ljscale(i,j) = 0;
        elseif distance(i,j) > eq_dis(i,j)
            ljscale(i,j) = (4*ma*(((eq_dis(i,j)/distance(i,j))^12) -
((eq_dis(i,j)/distance(i,j))^6)))/distance(i,j);
        elseif distance(i,j) < eq_dis(i,j)
            ljscale(i,j) = (-(mr/(eq_dis(i,j)^2))*((distance(i,j))^2)
+mr)/distance(i,j);
        end
```

```
        % This populates the force arrays. Distance x Scale.
        xforce(i,j) = -xdistance(i,j)*ljscale(i,j);
        yforce(i,j) = -ydistance(i,j)*ljscale(i,j);

    end
end


displacement = [deltat*sum(xforce,2) deltat*sum(yforce,2)];
% The force of each component affecting each particle. (nN)


%% Iterative loop


flag =0;
while maxn > ncount && flag == 0
    % This loop iterates as long as a single boolean value is greater than
zero. That is, at least one particle is not within its force tolerance.
    figure(1)
    % plot(position(:,1), position(:,2),'ko');
    viscircles([position(:,1), position(:,2)],radius);
    % This graph the voronoi tessellation. Can change to 'plot'.

    %Can uncomment this for voronoi coloring.
    %{
     voronoi(position(:,1), position(:,2), 'ko');
     x=[position(:,1), position(:,2)];
     [v,c]=(voronoin(x));
     for i=1:length(c)
         if all(c{i}~=1)
             patch(v(c{i},1),v(c{i},2),i);
         end
     end
    %}
    xlim([-75,75]);
    ylim([-75,75]);


    axis equal
    % This centers the axes.

    % CELL SPLITTING
    for i = 1:ncount
        if splitsize <= radius(i)
            if ncount >= maxn
```

```
                flag =1;
                break
            end
            position(ncount+1,:) = position(i,:) + [0.0001 0];
            radius(i) = radius(i)/sqrt(2);
            radius(ncount+1) = radius(i);

            drdt(ncount+1) = 1 + (4-1).*rand();
            ncount=ncount+1;
        end
    end



    displacement = [deltat*sum(xforce,2) deltat*sum(yforce,2)];
    % The net force for each particle. This includes the force from all
    % other particles in the graph.

    position = position + displacement;
    % This updates the position array based on the net force.
    for i=1:ncount
           radius(i) = radius(i) + deltat*drdt(i);
    end

    for i = 1:ncount
        for j=1:ncount

            % This initializes the distance arrays.
            if i~=j % The distance between a particle and itself is 0. The
array is initially zeros.
                xdistance(i,j) = position(j,1)-position(i,1); % This finds
the x distance between each particle and stores them in an array.
                ydistance(i,j) = position(j,2)-position(i,2); % This finds
the y distance between each particle and stores them in an array
                eq_dis(i,j) = radius(i) + radius(j);
            end

            % This populates the distance array using the pythagorean
theorem.
            distance(i,j) =
sqrt((position(i,1)-position(j,1))^2+(position(i,2)-position(j,2))^2);



            % This populates the magnitude scalar array using the Lennard
Jones
            % potential.
```

```
            if i==j
                ljscale(i,j) = 0;
            elseif distance(i,j) > eq_dis(i,j)
                ljscale(i,j) = (4*ma*(((eq_dis(i,j)/distance(i,j))^12) -
((eq_dis(i,j)/distance(i,j))^6)))/distance(i,j);
            elseif distance(i,j) < eq_dis(i,j)
                ljscale(i,j) = (-(mr/(eq_dis(i,j)^2))*((distance(i,j))^2)
+mr)/distance(i,j);
            end

            % This populates the force arrays. Distance x Scale.
            xforce(i,j) = -xdistance(i,j)*ljscale(i,j);
            yforce(i,j) = -ydistance(i,j)*ljscale(i,j);

        end
    end

    %pause(0.01)
    axis equal
    figure(1)
    % plot(position(:,1), position(:,2),'ko');
    viscircles([position(:,1), position(:,2)],radius);

end

%% Iterative Loop after cells done splitting
for k=1:200
    % This loop iterates as long as a single boolean value is greater than
zero. That is, at least one particle is not within its force tolerance.
    figure(1)
    % plot(position(:,1), position(:,2),'ko');
    viscircles([position(:,1), position(:,2)],radius);
    % This graph the voronoi tessellation. Can change to 'plot'.

    %Can uncomment this for voronoi coloring.
    %{
     voronoi(position(:,1), position(:,2), 'ko');
     x=[position(:,1), position(:,2)];
     [v,c]=(voronoin(x));
     for i=1:length(c)
         if all(c{i}~=1)
             patch(v(c{i},1),v(c{i},2),i);
         end
     end
    %}
    xlim([-75,75]);
    ylim([-75,75]);
```

```
    %axis equal
    % This centers the axes.



    displacement = [deltat*sum(xforce,2) deltat*sum(yforce,2)];
    % The net force for each particle. This includes the force from all
    % other particles in the graph.

    position = position + displacement;
    % This updates the position array based on the net force.
    for i=1:ncount
           radius(i) = radius(i) + deltat*drdt(i);
    end

    for i = 1:ncount
        for j=1:ncount

            % This initializes the distance arrays.
            if i~=j % The distance between a particle and itself is 0. The
array is initially zeros.
                xdistance(i,j) = position(j,1)-position(i,1); % This finds
the x distance between each particle and stores them in an array.
                ydistance(i,j) = position(j,2)-position(i,2); % This finds
the y distance between each particle and stores them in an array
                eq_dis(i,j) = radius(i) + radius(j);
            end

            % This populates the distance array using the pythagorean
theorem.
            distance(i,j) =
sqrt((position(i,1)-position(j,1))^2+(position(i,2)-position(j,2))^2);



            % This populates the magnitude scalar array using the Lennard
Jones
            % potential.
            if i==j
                ljscale(i,j) = 0;
            elseif distance(i,j) > eq_dis(i,j)
                ljscale(i,j) = (4*ma*(((eq_dis(i,j)/distance(i,j))^12) -
((eq_dis(i,j)/distance(i,j))^6)))/distance(i,j);
            elseif distance(i,j) < eq_dis(i,j)
```

```
                ljscale(i,j) = (-(mr/(eq_dis(i,j)^2))*((distance(i,j))^2)
+mr)/distance(i,j);
            end

            % This populates the force arrays. Distance x Scale.
            xforce(i,j) = -xdistance(i,j)*ljscale(i,j);
            yforce(i,j) = -ydistance(i,j)*ljscale(i,j);

        end
    end

    %pause(0.01)
    %cla
    axis equal
    figure(1)
    % plot(position(:,1), position(:,2),'ko');
    viscircles([position(:,1), position(:,2)],radius);
end
```

Example of notochord sheath text file for Surface Evolver :

```
STRING
SPACE_DIMENSION 2
PARAMETER sheath_volume = 0.0723
PARAMETER interior_volume = 1.57
PARAMETER sheath_pressure = 1
PARAMETER interior_pressure = 0.04
PARAMETER sheath_interior = 96
PARAMETER interior_interior = 12
PARAMETER sheath_medium = 48
PARAMETER interior_medium = 1
#define sheath_sheath sheath_medium/1.5

vertices
1 1.100000 0.000000
2 0.842649 0.707066
3 0.191013 1.083289
4 -0.550000 0.952628
5 -1.033662 0.376222
6 -1.033662 -0.376222
7 -0.550000 -0.952628
8 0.191013 -1.083289
9 0.842649 -0.707066
10 1.250000 0.000000
11 0.957556 0.803485
```

```
12 0.217060 1.231010
13 -0.625000 1.082532
14 -1.174616 0.427525
15 -1.174616 -0.427525
16 -0.625000 -1.082532
17 0.217060 -1.231010
18 0.957556 -0.803485
19 1.014216 0.235689
20 -1.033662 -0.250815
edges
1 1 19 tension sheath_interior // changed
2 2 3 tension sheath_interior
3 3 4 tension sheath_interior
4 4 5 tension sheath_interior
5 5 20 tension sheath_interior // changed
6 6 7 tension sheath_interior
7 7 8 tension sheath_interior
8 8 9 tension sheath_interior
9 9 1 tension sheath_interior
10 10 11 tension sheath_medium
11 11 12 tension sheath_medium
12 12 13 tension sheath_medium
13 13 14 tension sheath_medium
14 14 15 tension sheath_medium
15 15 16 tension sheath_medium
16 16 17 tension sheath_medium
17 17 18 tension sheath_medium
18 18 10 tension sheath_medium
19 1 10 tension sheath_sheath
20 2 11 tension sheath_sheath
21 3 12 tension sheath_sheath
22 4 13 tension sheath_sheath
23 5 14 tension sheath_sheath
24 6 15 tension sheath_sheath
25 7 16 tension sheath_sheath
26 8 17 tension sheath_sheath
27 9 18 tension sheath_sheath
28 19 20 tension interior_interior
29 19 2 tension sheath_interior // added
30 20 6 tension sheath_interior // added
faces
1 19 10 -20 -29 -1 // phase 2 // changed
2 20 11 -21 -2 // phase 2
3 21 12 -22 -3 // phase 2
4 22 13 -23 -4 // phase 2
5 23 14 -24 -30 -5 // phase 2 //changed
6 24 15 -25 -6 // phase 2
```

```
7 25 16 -26 -7 // phase 2
8 26 17 -27 -8 // phase 2
9 27 18 -19 -9 // phase 2
10 29 2 3 4 5 -28 // phase 1 // changed
11 30 6 7 8 9 1 28  // phase 1 // changed
bodies
1 1 volume sheath_volume
2 2 volume sheath_volume
3 3 volume sheath_volume
4 4 volume sheath_volume
5 5 volume sheath_volume
6 6 volume sheath_volume
7 7 volume sheath_volume
8 8 volume sheath_volume
9 9 volume sheath_volume
10 10 volume interior_volume
11 11 volume interior_volume
read
gogo := {g; r; V 20; g; r; g; V 20; r; g; V 20; r; g 5; }
gogo2 := {o; r; V 200; g; r; V 200; g 3; r; V 200; j .05; g 10; }
col := {set background white}
```
