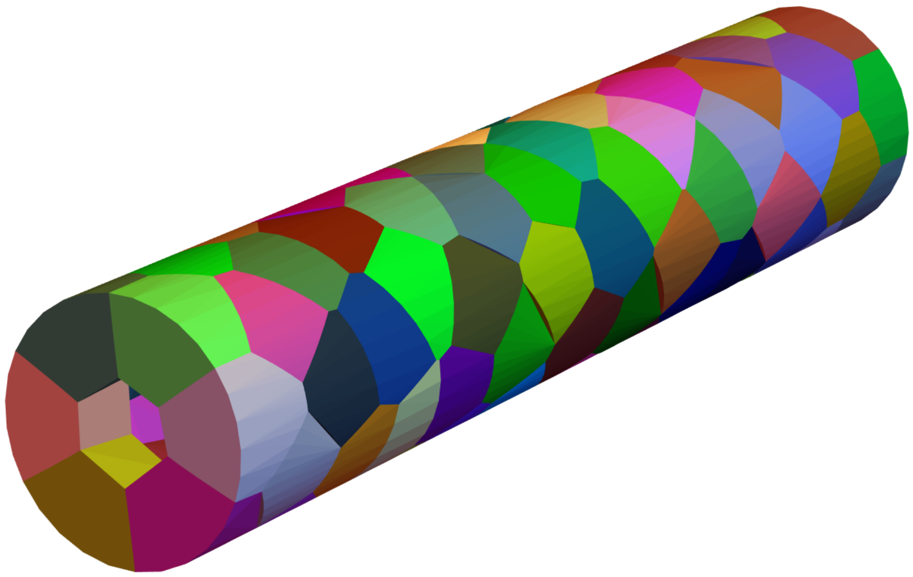
3DTubularVoronoi – User Manual V1

Léna Guitou, Eloy Tomás Serrano Andrés, Alberto Conejero and Javier Buceta

February 3, 2025



Abstract

Computational model for 3D tissue simulation modeling tubuluar epithelial tissues. Using Voronoi tessellations, a computational geometry tool that divides the plane into polygonal disjoint convex cells. We also take into account different cellular dynamics with the Metropolis algorithm. Through this approach, we have achieved a realistic model consistent with the most recent research in which a new shape, very famous in the recent literature, emerges naturally: the scutoid.

Contents

1 Model

We model tubular epithelia as a 3D solid cylinder with an inner cylindrical cavity which represents the lumen. Our approach consists in discretizing this solid into intermediate cylinders, the intern cylinder representing the apical layer and the external cylinder representing the basal layer (fig 1).

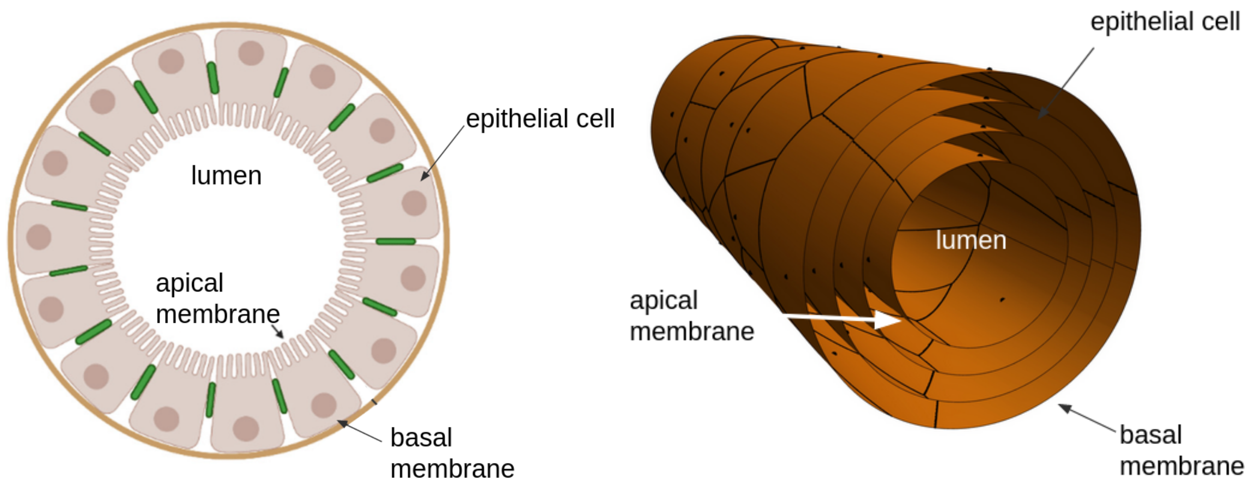


Figure 1: Modeling tubular epithelia. Left: Scheme of the front view of a tubular epithelia. Right: 3D representation of a epithelia tube discretized into layers. Voronoi tesselations are defined by the orthogonal projections (black points).

1.1 Voronoi tessellations on cylindrical surfaces

Let's consider a tubular epithelia of length L composed of N cylinders. We want to build Voronoi tesselations on each layer of the tube. However, computing the geodesic distance in 3D can be very time consuming and can suffer from a loss of precision. That is why we transfered the cylinder to the plane. The most intuitive way to treat the cylinder as a plane would be to "unroll" it, resulting in a flat rectangle (figure 3 A). The cylinder and the plane are topologically distinct, so there is no global isometry between them, however they present local isometry. This property allows us to translate each point of the cylinder to the plane, preserving the distances between them since the geodesic distance is maintained in the environment.

1.1.1 Parametrization

We can define the parametrization f_0 of the local isometry which translates a plane P of equation $z = 0$ to a cylinder C of length L and equation $x^2 + y^2 = 1$.

$$\begin{aligned} f_0 : P &\longrightarrow C \\ (u, v) &\mapsto (v, \cos(u), \sin(u)), \qquad (u, v) \in R \times R \end{aligned}$$

First, we consider the apical cylinder with a radius R_a parametrization:

$$\begin{aligned} f_1 : P_{\text{rect}} &\longrightarrow C_{R_a} \\ (u, v) &\longmapsto \left(v, R_a \cos\left(\frac{2\pi}{R_a} u\right), R_a \sin\left(\frac{2\pi}{R_a} u\right)\right), \quad (u, v) \in [0, R_a] \times [0, L] \end{aligned}$$

Then, the other cylinders have a parametrization such as:

$$\begin{aligned} f_k : P_{\text{rect}} &\longrightarrow C_{R_k} \\ (u, v) &\longmapsto \left(v, R_k \cos\left(\frac{2\pi}{R_k} u\right), R_2 \sin\left(\frac{2\pi}{R_k} u\right)\right), \quad (u, v) \in [0, R_k] \times [0, L], k = 2, .., N \end{aligned}$$

1.1.2 Boundary conditions

In the process of unrolling, there are inherent challenges because a plane and a cylinder are not topologically equivalent. Consequently, while we can locally transport points and paths, we can't do so globally. This distinction becomes evident when considering movement on the surface: on a cylinder, there's a direction where one can move and return to the starting point, which isn't possible on a bounded rectangle without specific boundary conditions. These conditions are crucial because in calculating Voronoi tessellations within a rectangle, points near opposite edges (which are close on the cylinder) become far apart, complicating the accurate calculation of the diagram's cells. This is why, we decided to use two replicas of the rectangle in order to overcome the issue of lateral border points (figure 2)

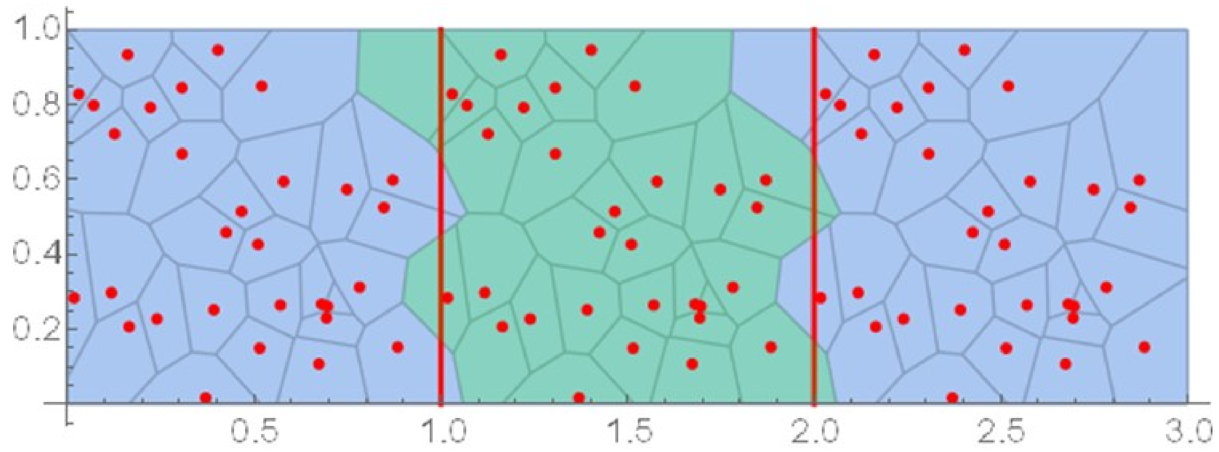


Figure 2: Replication of the cylinder unrolling, solving the topological issues. Image generated with Mathematica code. Red points represent the generating seeds, red lines correspond to the unrolling axis, and green Voronoi cells are the actual geodesic Voronoi diagram in the cylinder after the unrolling.

1.1.3 Building layers

First of call, we consider the apical layer and convert it into a plane so that is a rectangle of length L and width $2\pi R_a$ (fig. 3A). We set M random centroid points $C_i^a, i \in \{1, \dots, M\}$ of coordinates (u, v) and define the apical tissue using the geodesic Voronoi tessellations. Next, we build the layers by computing the orthogonal projection of each apical centroid, with a projection length $\Delta R = (R_b - R_a)/(N-1)$ (fig. 3B). Thus, the cylinders have a radius $R_j = R_a + j\Delta R,\ j \in \{0, \dots, N-1\}$ with centroids defined such as $C_i^j(u, v) = C_i^a(u, v) + j\tau(u, v)$, with $\tau(u, v)$ the normal vector to the tangent plane of the apical surface at the point $C_i^a(u, v)$. Then, we compute the geodesic Voronoi tessellations on each surface so that each cell C_i is defined by the projections of all the surfaces i.e the cell C_i is defined by $(C_i^a, C_i^2, ..., C_i^N)$

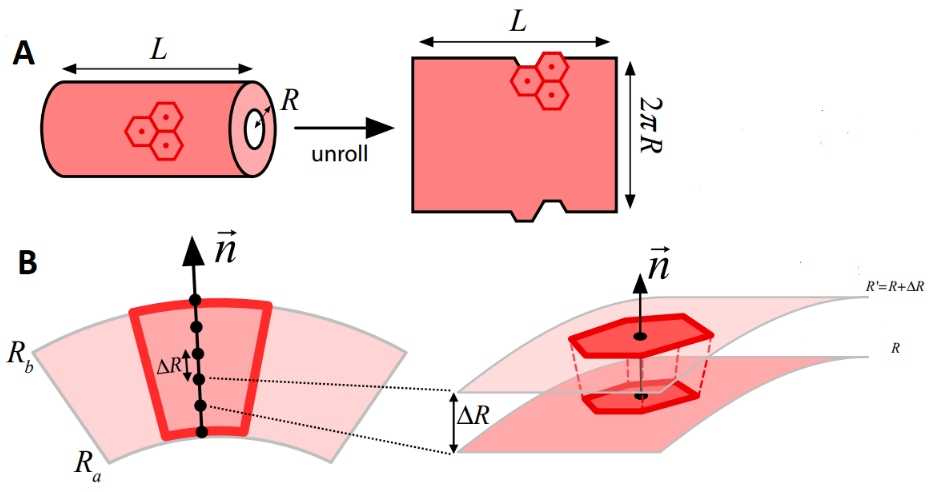


Figure 3: Overview of the model and algorithm. **A)** Unrolling of the exterior surface of a tube of length L and radius R. The hexagons are given by the Voronoi cells and represent tissue cells. After the unrolling, we treat the cylindrical surface as a plane, and so we need boundary conditions since the cylinder and the plane are not topologically equal. **B)** Structure of a single cell in the model. We discretize the solid tube into different layers. If the tube has an interior radius R_a (apical) and an exterior radius R_b (basal), discretizing into N layers we get $\Delta R = (R_b - R_a)/(N-1)$, and N cylinders with radius $R_j = R_a + j\Delta R,\ j \in \{0, \dots, N-1\}$. To each tissue cell, we assign a generator seed in the interior layer of the tube (at radius R_a) and project it orthogonally into superior layers. Each cylindrical layer is unrolled as in A), and the seeds generate a Voronoi cell in each layer.

1.2 Energy relaxation

1.2.1 Total energy

To match experimental observations, the system has to reach an energerical steady state i.e state in which the tissue geometry is realistic. To do so, we consider the mechanical properties of each cell and compute the total energy of the system. The energy E_i^j of the Voronoi centroid i in the layer j is defined as

$$E_i^j = \frac{K}{2}(A_i^j - A_0)^2 + \frac{\Gamma_j}{2}(P_i^j)^2 + \Lambda P_i^j \tag{1}$$

with K the elastic constant, A the cell area, A_0 the prefered cell area, Γ the contractility constant, P the cell perimeter and Λ the adhesion constant. We assume the elastic and adhesion constant homogenous along the radius and we assume that the contractility constant depends on the radius such as the contractility forces are higher on the apical layer. Thus, we define the ratio $s_j = \frac{R_j}{R_a}$ to have

$$\Gamma_j = \Gamma_0 e^{1-\frac{s_j}{s_0}}$$

with Γ_0 and s_0 constants. Then, the total energy of the tube is defined such as

$$E_{\text{total}} = \sum_{i=1}^{M}\sum_{j=1}^{N} E_i^j$$

1.2.2 Dimensionless energy

We adimensionalize the system using the characteristic mass m_c, length l_c, and time t_c to define the dimensionless area $\tilde{A}_i = A_i/l_c^2$, the prefered cell area $\tilde{A}_0 = A_0/l_c^2$ and the cylinder length $\tilde{L}_i = L_i/l_c$. We assume a characteristic energy e_c such as $E_i = \tilde{E}_i e_c$ so that $\tilde{E}$ is dimensionless. In addition, let's set $l_c = \sqrt{A_0}$ so that $\tilde{A}_0 = 1$. Then, we define $\tilde{k} = kA_0^2/e_c$ and $e_c = KA_0^2$ so that $\tilde{k} = 1$. Finally, we set $\tilde{\Gamma} = \Gamma/kA_0$ and $\tilde{\Lambda} = \Lambda/kA^{\frac{3}{2}}$. The dimensionless system is then defined such as

$$\tilde{E}_i = \frac{1}{2}(\tilde{A}_i - 1)^2 + \frac{1}{2}\tilde{\Gamma}\tilde{L}_i^2 + \tilde{\Lambda}\tilde{L}_i \tag{2}$$

1.2.3 Preferred cell area

We want to evaluate the constant A_0 which represents the cell area in a flat condition i.e without tissue curvature. In our cylindrical model, cells tend to have a larger area at the basal layer due to the basal surface's larger area compared to the apical surface. This discrepancy influences adhesion energy calculations, which rely on the assumption of cell volume conservation, given the constant liquid content within cells. In average, the cell area in layer j is

$$\langle A_j \rangle = \frac{2\pi R_j}{M} = \frac{2\pi R_a}{M} s_j = \langle A_a \rangle s_j$$

Then, the average energy E_j of the layer j of radius R_j is given by

$$\langle E_j \rangle = \frac{\tilde{K}}{2} \langle (A_a s_j - A_0)^2 \rangle$$

and reaches a minimum at $s^* = \frac{A_0}{\langle A_a \rangle}$. This means that the adhesion energy reached its minimum on the layer k such as $s^* = R_k/R_a$. To find A_0, we apply the volume conservation rule. On one hand, the average volume of a cell on a flat tissue is $\langle V_{flat} \rangle = A_0(R_b - R_a)$. On other hand, the average volume of the cells on a curved tissue is $\langle V_{curv} \rangle = \frac{\pi L(R_b^2 - R_a^2)}{M}$. Therefore,

$$A_0 = \frac{\pi L(R_b + R_a)}{M}$$

so

$$s^* = \frac{R_b + R_a}{2R_a}$$

i.e the layer in which the adhesion energy is the lowest is in the middle of the tissue.

1.2.4 Establishing Real-Time from Algorithm Iterations

To connect the algorithm's iterations to real-time, we base our approach on the method presented by Farhadifar *et al.* [10.1016/j.cub.2007.11.049]. In their work, vertex displacement over time was modeled using the equation:

$$y(t) = d_0 + d_1 \left(1 - e^{-\frac{t}{\tau}}\right), \tag{3}$$

where τ represents the relaxation time for the system to reach equilibrium, ranging between $13\,\mathrm{s}$ and $94\,\mathrm{s}$. Similarly, we model energy relaxation using the equation:

$$y(x) = a + b\left(1 - e^{-\frac{x}{n_0}}\right), \tag{4}$$

where n_0 is a parameter linking iterations (x) to real-time.

To fit this model, we performed nonlinear regression using the *nls.multstart* package in R, which applies the Levenberg-Marquardt algorithm for least squares fitting. For 100 simulations, we fit the function to the energy relaxation curve for each simulation, averaged the results, and obtained the following parameters:

- **Single simulation:** $a = 277.62346$, $b = -86.38992$, $n_0 = 14.00112$,
- **Average across 100 simulations:** $a = 289.90758$, $b = -97.33677$, $n_0 = 15.80831$.

Using these parameters, we establish the relationship between iterations and real-time as follows:

$$\frac{t}{\tau} = \frac{x}{n_0} \quad \Rightarrow \quad t = \frac{x \cdot \tau}{n_0}, \tag{5}$$

where τ is the relaxation time in seconds, x is the iteration, and n_0 is the scaling parameter.

Real time per iteration:
For $\tau = 16\,\mathrm{s}$, $x = 1$, and $n_0 \approx 15.81$:

$$t = \frac{1 \cdot 16}{15.81} \approx 1.01\,\mathrm{s}. \tag{6}$$

For $\tau = 93\,\mathrm{s}$, $x = 1$, and $n_0 \approx 15.81$:

$$t = \frac{1 \cdot 93}{15.81} \approx 5.88\,\mathrm{s}. \tag{7}$$

The algorithm simulates relaxation times between 1 and 6 seconds per iteration, depending on the system's relaxation time (τ).

1.2.5 Algorithm

In order to reach energy relaxation, we use statistical mechanics and the algorithm described in figure 3. In short, successively cell centroids of a same layer are selected and moved following a 2-dimensional Gaussian distribution. The evolution of the movement is given by the Markov chain. At each movement, Voronoi tesselations are computed and the total energy is calculated. If the energy decreases then the algorithm keeps going with the next cell. The movement is "accepted". If not, the Metropolis algorith gives the approval of motion depending on the Boltzmann distribution (probability $e^{-\beta(E_y - E_x)}$ with β the product of the Boltzmann constant and thermodynamic temperature). Once the M cells of the apical layer moved, it does the same to the following layers. We can repeat this process as much as wanted. The number of steps ($nSteps$) has to be enough larger to reach the thermal equilibrium.

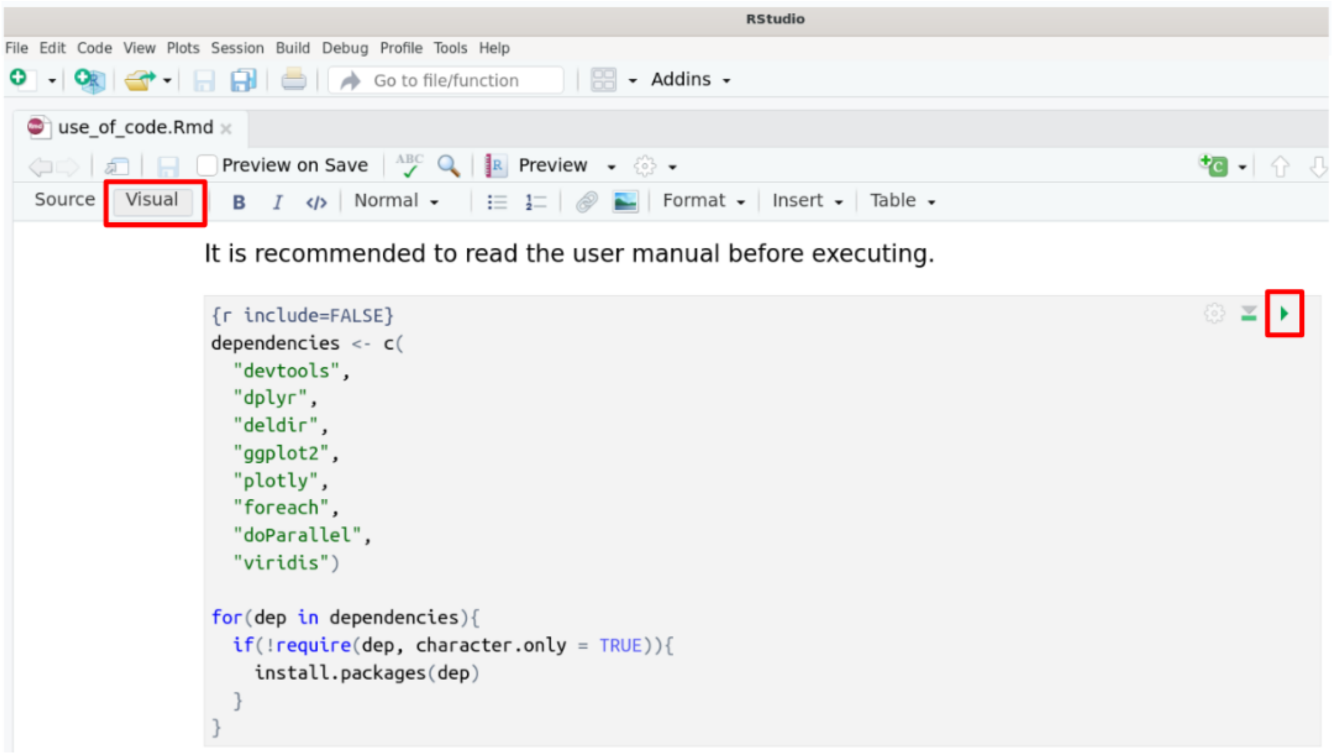
2 Installation and Execution

2.1 Installation

3DTubularVoronoi is open and available at `https://github.com/lenaguitou/3DTubularVoronoi.git`. Download the whole repository using **Code/Download ZIP**. The folder **tubular_simulations_source** contains all the source codes. You need to install R and RStudio to execute the code and Mathematica for tubes visualization (if you have a license).

2.2 Execution

To start open the file *use_of_code.Rmd* with RStudio. Within this file you can see several sections delimited by gray boxes called chunks. Each chunk can be executed individually with the green triangle on the right top. When a chunk is running, the green triangle is replaced by a red square until the execution is over. If you are not familiar with R and Studio, it is recommended to use the "Visual" mode, accesible on the left top next to "Source". It is recommended to read the comments within the code and not execute all the code at once. The necessary packages (devtools, dplyr, deldir, ggplot2, plotly, foreach, doParallel and viridis) can be installed executing the first chunk. Assure that all the packages have been well installed by looking at the terminal for eventual errors.



It is recommended to read the user manual before executing.

```
{r include=FALSE}
dependencies <- c(
  "devtools",
  "dplyr",
  "deldir",
  "ggplot2",
  "plotly",
  "foreach",
  "doParallel",
  "viridis")

for(dep in dependencies){
  if(!require(dep, character.only = TRUE)){
    install.packages(dep)
  }
}
```

Figure 4: All the necesary packages can be installed executing the first chunk. A chunk can be executed with the green triangle. Visual mode is recommended for users who are not familiar with R.

2.3 Single simulation

To launch one single simulation, refer to the section **1/ Single Simulation**. Here is the description of each chunk:

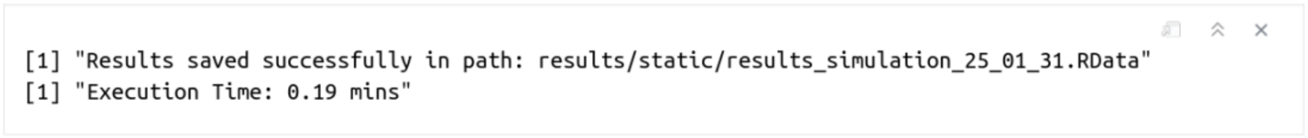
2.3.1 Parameters

Parameters for a single simulation can be edited in the list called *params* defined such as follows:

1. *seed*: generating pseudo-random numbers
2. n_steps: simulation steps
3. n_cells: number of cells (M)
4. n_layers: numbers of layers (N)
5. $apical\_rad$: apical radius (R_a)
6. $ratio\_rad$: ratio between apical and basal radius $\frac{R_a}{R_b}$
7. cyl_length: cylinder length (L)
8. *kappa* κ: cell membrane elasticity (always 1 because of adimentionality of the model)
9. *gamma* Γ: cell membrane contractility (Γ_0)
10. *lambda* Λ: cell-cell adhesion constant
11. $beta$: constant involved in jumps because of perturbations [1]
12. $s0\_ratio$: characteristic ratio between the actual radius and apical radius

2.3.2 Execution

Execute the chunk without editing anything. Once it is over, a message should display about the right data storage in the **results** folder and the execution time of the code.



```
[1] "Results saved successfully in path: results/static/results_simulation_25_01_31.RData"
[1] "Execution Time: 0.19 mins"
```

Figure 5: Message displayed that validates that the algorithm worked

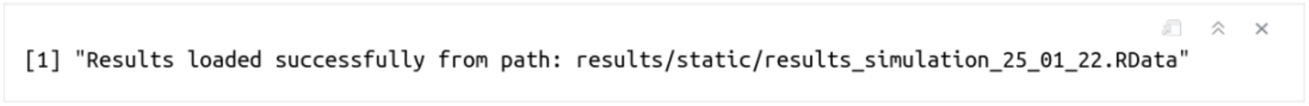
This chunk returns a data frame called *simulation* containing 3 elements: a data frame called *result_alg*, a list called *parameters* and a string called *algorithm*. The data frame *result_alg* contains the history of the points coordinates (*points_evolution*) and the history of the total energy (*energy_evolution*)

- *points_evolution* contains $(n\_steps + 1) \times n\_cells$ rows. Each row represents a point, the columns "x" and "y" represent the x and y coordinates of the point, and the column "Iteration" represents the step of the algorithm.
- *energy_evolution* contains $(n\_steps + 1)$ rows and 2 columns, one indicating the iteration and the other one indicating the corresponding total energy of the tube.

For simplicity we store these 2 lists in the variables (*hist_points* and *hist_energy*). The list *parameters* allows to access any parameter value using the symbol "$". For instance, the number of layers of the simulation can be called by "parameters$n_layers".

2.3.3 Loading results

To access data from previous simulations, indicate the file you want to import in *fileName*. Results are stored in the same variable names.

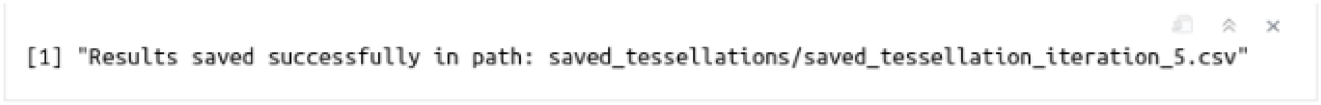


```
[1] "Results loaded successfully from path: results/static/results_simulation_25_01_22.RData"
```

Figure 6: Caption

2.4 Tube visualization

To visualize the tube, you first need to save the data from the algorithm. Before executing the chunk, select the frame to visualize by editing the variable *saveIteration*. The chunk generates a csv file that is saved in the folder "saved_tesselation" and the file is named "saved_tesselation_iteration_i" where i is the selected iteration. A message should display to indicate that the file had been properly saved.



```
[1] "Results saved successfully in path: saved_tessellations/saved_tessellation_iteration_5.csv"
```

Figure 7: Caption

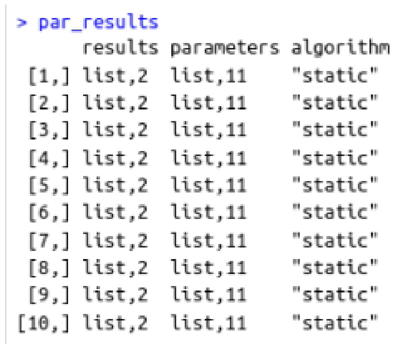
This file contains the cell id (*id_cell*), the number of layers (*Layer*), the cell radius (*Radius*), the coordinates of the cell centroids (*Centroidx* and *Centroidy*), the number of vertices of the cell (*n_vertices*), the coordinates of the vertices i with 11 maximum (*vertix,vertiy*). Thus, it follows this format:

id_cell,layer,radius,centroidx,centroidy,n_vertices,vert1x,vert2x,....vert11x, verty1,verty2,.....vert11y

Then, you can open the Mathematica script called "TubularGraphicsMain.nb" and edit the path. Once you entered the right path and right name of the csv file, you can execute the whole script. To do this, you can simply select "Evaluation" and then "Evaluate notebook". The function **cellreprlayerscover** allows to plot a single cell and **Graphics3D** allows to plot the whole tube.

2.5 Parallelization

To launch several simulations simultaneously with different seeds (random initial conditions), refer to the section **2/ Parallelization**. Choose the number of simulations using the variable *N_SIM* and edit the list of parameters. The dataframe generated stored in the variable *par_results* contains *N_SIM* dataframes, each one composed as described in 2.3.2.



```
> par_results
      results parameters algorithm
 [1,] list,2  list,11    "static"
 [2,] list,2  list,11    "static"
 [3,] list,2  list,11    "static"
 [4,] list,2  list,11    "static"
 [5,] list,2  list,11    "static"
 [6,] list,2  list,11    "static"
 [7,] list,2  list,11    "static"
 [8,] list,2  list,11    "static"
 [9,] list,2  list,11    "static"
[10,] list,2  list,11    "static"
```

Figure 8: Example of the data frame *par_result* with *N_SIM = 10*

As for the single somulation, you can load previous results choosing the *fileName* and the number of simulations *N_SIM* you want to import. You also can save the tesselation data of a selected iteration of a selected simulation.