Computer Modeling of Thymic Epithelial Cell Morphology

Masulah, Bidayatul

B. Masulah

Computer Modeling of Thymic Epithelial Cell Morphology

Master thesis

February 13, 2025

Thesis supervisors: **Dr. E. Tsingos**
Prof. dr. R.M.H. Merks

Leiden University
Mathematical Institute

Contents

Abstract

Many types of cells in the body exhibit intricate shapes, often characterized by elongated projections that are essential for their function. Nerve cells are a prime example, but even immune cells such as thymic epithelial cells (TECs) in the thymus rely on these dynamic structures. TECs are large cells that play a crucial role in the maturation of thymocytes and the development of the immune system. Interestingly, studies in zebrafish (*Danio rerio*) embryos have shown that TECs transform from a rounded shape to one with long projections after engaging with thymocytes, highlighting the dynamic nature of their shape.

Despite the importance of these protrusions, previous computational models of TECs have often assumed that they have a fixed morphology, overlooking their dynamic nature. This study addresses this shortcoming by developing a cutting-edge, particle-based dynamic model using Tissue Forge software. The model I developed simulates the formation and retraction of TEC protrusions, incorporating structural constraints and cell division dynamics. In the future, this model can be expanded to include how the morphology of TECs adapts in response to interactions with thymocytes and external forces. This model provides a dynamic framework for understanding TEC behavior, enabling future studies of TEC-thymocyte interactions and the role of TEC morphology in the development of the immune system.

1 Introduction

Cells in our body often have complex shapes, including long, thin projections. In addition to the well-known nerve cells, other important examples are supportive cells in the immune system, such as those found in lymph nodes and the thymus. The thymus, which is essential for the development of the immune system in vertebrates, is composed of two main cell types: cells from the lymphoid lineage, collectively called 'thymocytes', and thymic epithelial cells (TECs), a non-hematopoietic cell type [1]. Thymocytes, which encompass immature and developing T cells, are small and round, while TECs are larger and have many protrusions.

In previous computational work, TECs have been modeled as fixed structures [2, 3], limiting our understanding of their dynamic roles in the thymic environment. The static models fail to account for alterations in TEC morphology over time, which are crucial for thymocyte maturation and molecular diffusion. In this work, I will address this limitation by developing a subcellular-element model of TECs. Before delving into the modeling work, I will give an overview of the anatomical structure and cellular dynamics of the thymus, with a focus on TEC morphology and interactions with thymocytes. While the biological processes involved are still an active area of research, only key concepts and mechanisms necessary to comprehend the model framework are discussed.

1.1 Structure and function of the thymus

This section provides an overview of the anatomical and physiological characteristics of the thymus microenvironment, emphasizing the role of TECs in T cell development.

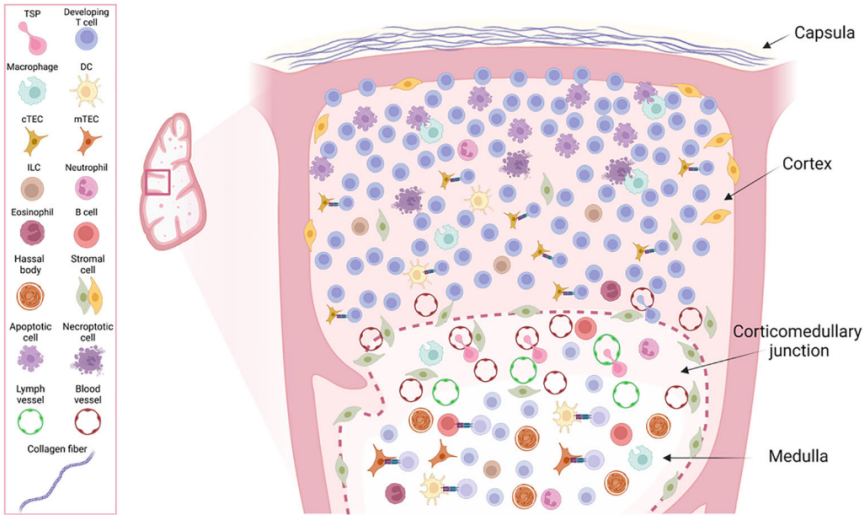


Figure 1: The structural layers and cellular makeup of the thymus lobe. The figure provides a detailed view of the four morphological layers, progressing from the outer to the inner regions: the capsule, cortex, corticomedullary junction (CMJ), and medulla. It highlights the variety of cell types within these layers and their documented anatomical locations within the thymus. The illustration does not reflect the relative abundance of these cells in the thymus tissue. Adapted from [4].

The thymus is a specialized primary lymphoid organ crucial for the development of the immune system, specifically for the maturation of T cells [5, 6]. It is where T lymphocytes, also known as thymocytes, undergo a series of developmental steps that are necessary for their ability to recognize and respond to pathogens while tolerating self-antigens. This process is critical for adaptive immunity and overall immune competence [5, 6].

The thymus is a bilobulated organ, separated by an interlobular septum and linked by connective tissue [4]. In humans, it is located in the anterior thoracic cavity, just above the heart. The organ is encased in a connective tissue layer known as the capsule, which envelops both lobes [5, 4]. The thymus is further divided into distinct regions (Fig. 1):

- Cortex: The outer region of the thymus characterized by a high density of immature lymphocytes or thymocytes [4, 6]. The cortex is the site of positive selection of developing T cells [5]. In fish, the entry and exit of thymocytes occurs in the cortex [2].

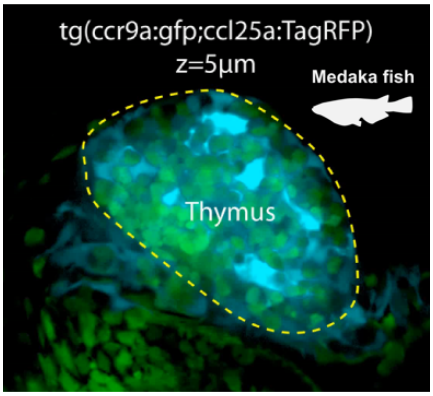


Figure 2: The thymic niche in Medaka fish (*Oryzias latipes*). The fluorescent label shows thymocytes in green and TECs in cyan. Adapted from [2].

- Medulla: The inner layer of the thymus, which is less dense than the cortex, containing mature T cells [5, 4]. The medulla is where negative selection occurs, which removes self-reactive T cells. The medulla also includes medullary thymic epithelial cells (mTECs), Hassall's corpuscles, macrophages, dendritic cells, B cells, and other myeloid cells.
- Corticomedullary Junction (CMJ): In mammals, the area that borders the inner cortex and the medulla, characterized by a high density of blood and lymph vessels [6]. In mammals, the CMJ also serves as a hub for both incoming immature lymphocytes and mature lymphocytes preparing to exit the thymus [4].

At the micro-anatomical level, the thymus consists of various immune cell types, connective tissue, and extracellular matrix components [4]. The most important cellular components are T cells and the specialized TECs, which are the main protagonists in thymopoiesis [6]. TECs can be broadly classified into several subtypes:

- Cortical TECs (cTECs): Located in the cortex, cTECs are crucial for positive selection by displaying small protein fragments to developing thymocytes. Their network resembles a growing vascular system, forming a mesh-like structure that is essential for thymocyte maturation. [6].
- Medullary TECs (mTECs): Found in the medulla, mTECs are responsible for the negative selection of T cells, removing those that react to self-antigens. mTECs also express the Autoimmune Regulator (AIRE) which is crucial for self-tolerance [5, 6].
- mcTECs (medullary-cortical TECs): Recent studies [4, 5] have identified a TEC subtype called mcTECs, which are bipotent progenitor cells located in the capsule or CMJ, capable of developing into either cTECs or mTECs.
- PVS-TECs (Perivascular Space TECs): TECs are also found in the perivascular spaces, around the blood vessels in the thymus, which are involved in immune cell migration [6, 5].

The TECs in the thymus play a similar role to that of fibroblastic reticular cells (FRCs) in lymph nodes by creating a supportive microenvironment that is crucial for T cell development and selection [7]. The FRC network is composed of interconnected cells with complex protrusions, forming a microenvironment that regulates immune cell movement, cell interactions, and molecule transport [7]. Like FRCs, TECs establish a structural and functional scaffold that is essential for the immune cells within the organ to perform their functions [6].

TECs are a key component of the thymic microenvironment (Fig. 2), orchestrating T cell development and maintaining thymic function. TECs are anatomically and functionally specialized, forming a three-dimensional (3D) network within the thymus. Their star-shaped morphology, characterized by protrusions that contact neighboring TECs, facilitates the creation of distinct thymic niches that influence T cell maturation [3].

Early T cell development begins with the migration of multipotent progenitors from the bone marrow to the thymus. In mammals, these progenitors enter the thymus at the corticomedullary junction (CMJ) [4]. In fish, progenitors enter the thymus from the cortex [2]. They are then guided through various

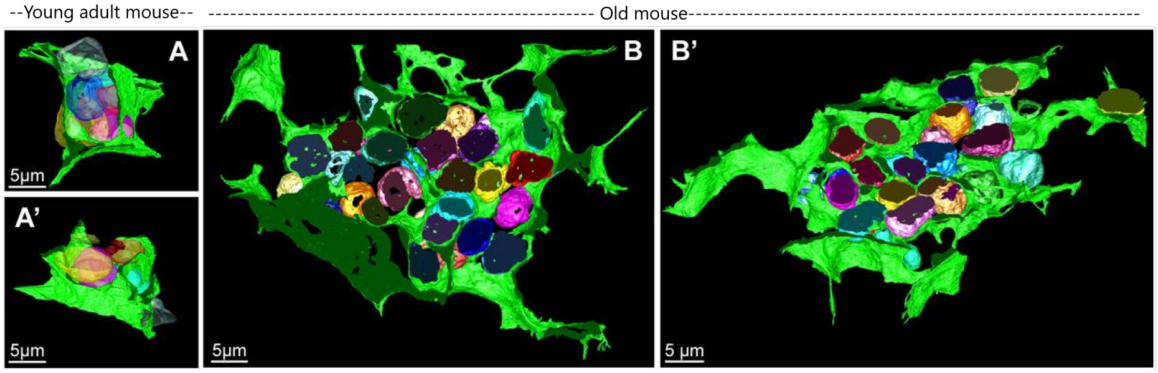


Figure 3: Three-dimensional reconstruction of the cTEC network (green) in the thymus of a mouse, interacting with thymocytes, represented by their translucent, multicolored nuclei. Panels A-A' illustrate the network structure in a young adult mouse, while panels B-B' show the same in an aged mouse, with alternative angles and representations provided in A' and B'. Adapted from [6].

compartments of the thymus, starting from the medulla or CMJ, then migrating to the cortex. This migration is regulated by chemokine receptors such as CCR7 and CCR9. Once inside the thymus, these progenitors, now called early thymic progenitors (ETPs), interact with the thymic stroma and are directed toward a T cell fate [4].

The thymic stroma is comprised of various cell types, including TECs, which are essential in guiding T cell development [4]. TECs form a complex 3D network with their protrusions, creating a microenvironment that is essential for thymocyte development. Fig. 3 and Fig. 4 show how TECs cradle T cells in spatially distinct niches that support the different stages of T cell development. TECs are not merely structural components but also actively participate in T cell selection [5]. The network of cortical TECs (cTECs) supports positive selection, allowing only thymocytes with working T cell receptors that can recognize specific protein markers on cell surfaces to survive [4, 7, 6]. cTECs display these protein fragments on their surface, and thymocytes that interact with them at just the right level continue developing [7, 6]. In contrast, thymocytes that fail to interact, or bind too strongly, are eliminated through cell death or redirected to become regulatory T cells [4].

Fish offer several advantages as a biological model for studying thymic biology, including their transparent embryos, rapid development, and well-characterized immune system, which closely parallels that of mammals [2]. Using medaka fish (*Oryzias latipes*) has provided critical insights into TEC organization and function within the thymus, establishing its value as a model system for studying thymic architecture and its role in immune development (Fig. 2). Experimental observations in zebrafish (*Danio rerio*) embryos have shown that TECs, which are initially round, begin to develop projections after first interacting with thymocytes during early development [8]. Furthermore, in medaka fish embryos, the number of TECs can adjust to thymocyte proliferation, increasing or decreasing as the number of thymocytes changes [2, 3].

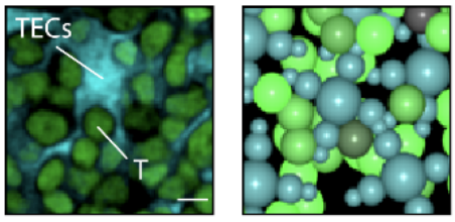


Figure 4: The spatial interaction between TECs (cyan) and thymocytes (T; green) in medaka fish. Left shows a detail from an experimental study, right shows a simulation. Adapted from [2].

The density, size, and number of protrusions in TECs are critical architectural parameters that directly influence thymocyte population size. In medaka fish, TECs occupy approximately 47% of the thymic volume [3]. Simulations conducted using virtual thymus models have demonstrated that increasing TEC density, size, and protrusion numbers results in a nearly two-fold increase in thymocyte numbers [3]. Conversely, reductions in these architectural features, particularly in TEC size, lead to significant decreases in the thymocyte population. This relationship underscores the role of TEC architecture in regulating

thymocyte proliferation, likely through its impact on the spatial distribution and availability of critical signaling molecules [3]. For instance, cTECs in the cortex provide cytokines such as interleukin-7 (IL-7), which are essential for early thymocyte development [2]. Meanwhile, mTECs in the medulla present self-antigens that facilitate the elimination of autoreactive T cells, ensuring the establishment of central tolerance [9]. All TECs express the Notch ligand DLL-4 on their surface, which is critical for thymocyte development [2, 3].

Overall, the morphology and 3D architecture of TECs are directly linked to their functional roles in T cell development. The long protrusions and the resulting network of TECs create a dynamic and structured thymic environment, enabling the segregation of developmental compartments and the regulation of signaling molecule availability. This structural organization ensures the efficient progression of thymocytes through the various stages of development and selection. Unraveling the intricate relationship between TEC morphology and thymocyte development is critical to understand thymic function and offers potential avenues for addressing thymus-related diseases and immune disorders.

1.2 Previous modeling work

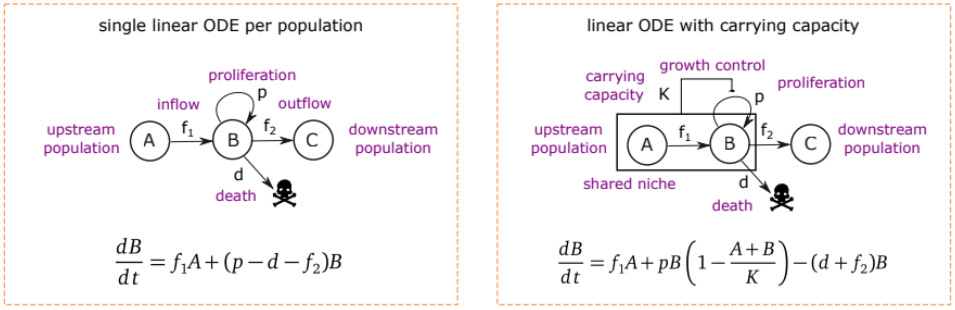


Figure 5: Mathematical models of thymic population dynamics often use different types of equations to simulate the progression of thymocyte populations through various stages of development. These models typically account for a source population (A) that generates progenitors fueling a subsequent population (B), which then differentiates into a later population (C). One approach (left) involves using simple linear ordinary differential equations (ODEs) to model proliferation (represented by circular arrows), cell death (depicted by skull icons), and differentiation (illustrated by flat arrows). Another approach (right) incorporates a regulated logistic growth model within the ODE framework, which accounts for a maximum carrying capacity (K) shared between populations (e.g., A and B). In these models, logistic growth control is applied by restricting cell proliferation rather than increasing cell death. This constraint reflects the competitive dynamics for shared niches within the thymus (shown as a large box). Adapted from [10].

The thymus, a central organ in the development of T cells, presents a fascinating yet complex system for mathematical modeling. The processes involved in T cell maturation, selection, and emigration, combined with the dynamic cellular interactions within the thymic microenvironment, necessitate diverse modeling approaches.

One of the key themes in thymic modeling is the use of different mathematical frameworks. Ordinary Differential Equations (ODEs) are a common approach, as seen in multiple studies [10]. These models often represent the thymus as a system of interconnected compartments (Fig. 5), each corresponding to a specific stage of T cell development, such as double negative, double positive, and single positive cells. ODEs are used to model the changes in the number of cells in each compartment, considering factors like cell influx, proliferation, differentiation, and death [10].

Another approach involves cellular automata (CA) models, which have been used to simulate thymocyte development. These models represent space as a grid, where each cell or location can exist in a specific state that is updated based on neighboring states [10]. The CA model developed by [11] simulates thymocyte migration and development within the thymus, where each site can be occupied by a cell or be empty (Fig. 6). This allows for a more direct simulation of the spatial aspects of thymocyte development, which are often overlooked by ODE-based models.

Furthermore, a paper by [12] utilizes the Cellular Potts Model (CPM) to simulate T cell movement in lymph nodes. As mentioned previously, the lymph nodes have a similar structure to the thymus, with the FRC network taking on the function of TECs [12]. Similar to the CA model, the space is divided into discrete units; however, in CPM, one cell can consist of multiple units and can also change shape and

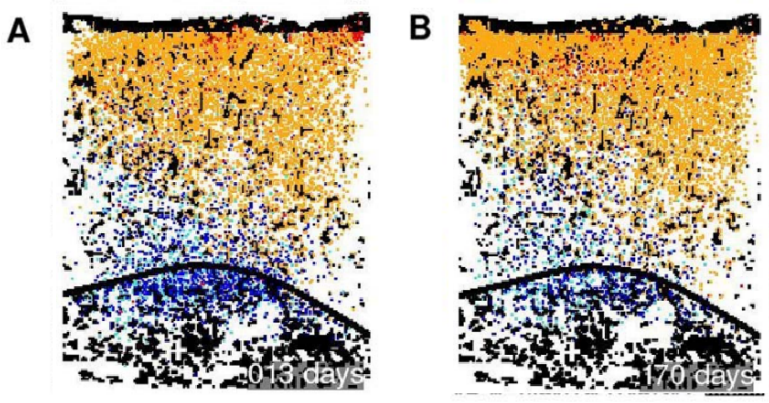


Figure 6: Cellular automata model of the thymus. The distribution, development, and temporal progression of thymocytes within the TEC network in mice are illustrated through their migration and maturation over time. The temporal dynamics are depicted in two frames: one showing the network at approximately 10 days (A) and the other at around 170 days (B). In these visualizations, different thymocyte populations are represented by distinct colors: black denotes the TEC network, red represents double-negative T cells, orange corresponds to double-positive T cells, blue indicates CD4 single-positive T cells, and cyan represents CD8 single-positive T cells. The black areas highlight locations where thymocytes interact with portions of the TEC network. Adapted from [11].

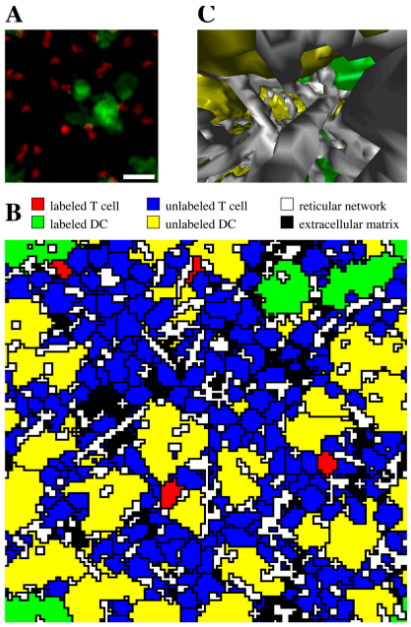


Figure 7: Simulations of T cells and dendritic cells (DCs) within a densely packed environment, based on a Cellular Potts Model (CPM), illustrate dynamic interactions and spatial organization over time. Panel (A) presents a top-view snapshot with compression along the z-axis at time 00:00, highlighting labeled T cells and DCs. Panel (B) displays a cross-sectional view at time 18:00, showcasing all cell types within the simulated space. Panel (C) provides a three-dimensional perspective at time 21:39, captured from the viewpoint of a T cell in motion. This view reveals the reticular network (gray), labeled DCs (green), and unlabeled DCs (yellow). Similar spatial patterns are observed in snapshots and cross-sectional views taken from different directions. The scale bar represents 20 μm. Adapted from [12].

interact with neighboring units based on surface energy. This model (Fig. 7) does not specifically address TECs, the thymus, or thymocytes but rather focuses on the behavior of T cells within lymph nodes. It demonstrates how interactions and morphological changes at the cellular level can shape T cell dynamics in complex environments, providing a foundation for exploring similar mechanisms in the thymus.

The virtual thymus model [2], illustrated in Fig. 8, offers a complementary perspective by focusing on T

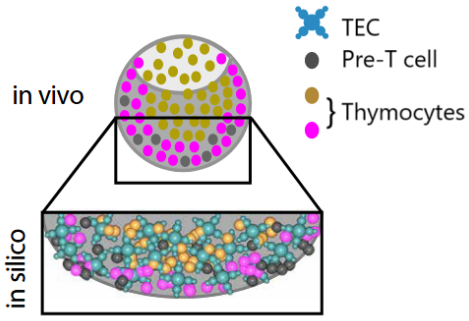


Figure 8: A spatial cell-based model for the cortical region of a medaka larval thymus. Adapted from [3].

cell development within a three-dimensional thymic microenvironment. Unlike the CPM approach, which emphasizes T cell behavior in lymph nodes, this agent-based model incorporates interactions between thymocytes and TECs in the thymus. While the model provides significant insights into the influence of TEC architecture on thymocyte behavior, it employs a static representation of TECs, limiting its capacity to capture dynamic cellular behaviors.

The model simulates thymocytes and TECs as individual agents within a three-dimensional thymic niche. The spatial distribution of TECs is represented by fixed parameters such as density, size, and the number of protrusions. This static approach, while computationally efficient, constrains the model's ability to simulate the dynamic adaptations of TECs. *In vivo*, TECs undergo continuous remodeling in response to interactions with thymocytes, external stimuli, or tissue-level changes, suggesting that a dynamic representation of TEC behavior is necessary for more accurate modeling [2].

However, the reliance on a static TEC network presents notable limitations. The model cannot account for morphological changes or adaptive responses of TECs, such as protrusion remodeling, proliferation, or migration, which are central to thymic development and the formation of functional thymic niches. This restricts the model's ability to explore dynamic feedback loops between TECs and thymocytes or simulate disease-driven alterations in thymic architecture. Incorporating dynamic behaviors into the virtual thymus model would address these limitations. This will allow for the exploration of key questions about TEC-thymocyte interactions, particularly their role in immune system development and the progression of thymus-related diseases such as T cell acute lymphoblastic leukemia.

2 Aim of this thesis

The aim of my Master thesis is to create a dynamic model of TECs by employing a particle-based modeling approach that could be used as an extension of the previous particle-based virtual thymus model [2, 3]. For this purpose, I will use the software Tissue Forge, an interactive platform that allows complex biological system modeling in much detail [13]. Using this tool, I can simulate changes in TEC morphology and number over time, which will in future allow to develop more complex models of the thymic environment. Central to this study is the question: **"How does the physical scaffold formed by TECs develop?"**, which will guide the exploration of TEC behavior and their role in thymic architecture.

3 Methods

3.1 Computational model

This study employs computer modeling to develop a spatially explicit representation of TECs and their dynamic morphology. As mentioned above, the previous model [2, 3] has limitations since software constraints did not permit to model TECs as multiple physically interconnected particles, and they were rendered as unchanging fixed objects in space. In this study, I focused on building a dynamic model of TECs and their cellular protrusions. I used a subcellular element model, a particle-based method where each cell can be represented by multiple particles, so as to simulate dynamic morphology [14]. For this purpose, I used the software Tissue Forge [13].

Tissue Forge, as described in [13], is an open-source platform for particle-based modeling and simulation, designed to address challenges in physics, chemistry, and biology. Users can construct simulations

Parameter	Value	Explanation
R_{main}	2.5	Initial radius of the main cell body. Units: micrometers
$R_{\text{protrusion}}$	0.5	Initial radius of a newly-formed protrusion segment. Units: micrometers
k	1	Stiffness constant of linear spring bonds. Units: Newtons per micrometer.
$n_{\text{max, p}}$	20	Maximum number of allowed protrusions. Unitless.
n_p	6	Number of protrusions in each main cell. Unitless.
$n_{\text{max, s}}$	4	Maximum number of allowed segments per protrusion. Unitless.
r_{growth}	0.01	Rate of growth or shrinking of protrusion segments. Units: Micrometers per time step.
r_{taper}	0.1	The parameter for subsequently decreasing the additional tip radius. Units: Micrometers.
d	-0.1	Morse potential depth. Negative sign indicates repulsion. Units: Joules
a	10	Morse potential width. Unitless.
$r_{0,\text{Morse}}$	0.0	Equilibrium distance in Morse potential. Defaults to 0.0. Units: micrometers.
f	1.5	Force magnitude applied at protrusion tips for active movement. Units: Newtons.
Pr[division]	0.1	Probability of cell division. Unitless.
l	3	Distance between daughter cells after cell division. Units: micrometers.

Table 1: Table of parameter values used in this study.

by utilizing built-in model components or defining custom models. These simulations can be executed interactively with real-time visualization or in high-performance computing environments, accommodating a broad range of scales from molecular to multicellular systems.

One of Tissue Forge's key features is its dynamic capabilities, allowing particles to be created, modified, or destroyed during a simulation through scripting or interactive commands. It supports modeling atoms, molecules, cells, and solid or fluid materials. The software also enables the incorporation of procedural code via user-defined functions, facilitating custom events and complex agent-based modeling. With user interfaces in C, C++, and Python, it supports development across multiple programming environments, including interactive platforms like IPython and Jupyter Notebooks. Tissue Forge is publicly available, well-documented with examples and guidelines, and maintained through a transparent and automated development cycle [13].

The codes written for this project have been included in the Appendix, and were uploaded to the following private GitHub repository: github.com/ComputationalAnimalDevelopment/dynamic_tec_protrusion

Parameter values used in this study are shown in Table 1.

3.2 Analysis methods

To analyze the spatial characteristics and organization of TECs during the simulation, I applied several computational techniques:

3.2.1 Voronoi Diagram

For analyzing the spatial distribution of particles in the simulation, I employed the Voronoi diagram. This method divides the simulation space into regions, where each region corresponds to the nearest particle. The diagram was generated using the `scipy.spatial.Voronoi` class, part of the `SciPy library`, which offers efficient algorithms for spatial partitioning [15]. The resulting diagram helps visualize how the TEC particles are arranged in space, revealing patterns of interaction between the main cells and protrusions. The visualization of the Voronoi diagram was created using `Matplotlib`, a plotting library that allows for detailed graphical representation of the regions and their boundaries [16]. This method helped to analyze the interactions between the main cells and protrusions within the TECs and their spatial arrangement.

3.2.2 2D Convex Hull Projection

I applied 2D Convex Hull Projection to analyze the shape and size of the TECs in the simulation. This method computes the convex boundary that encapsulates all the particles, providing a geometric representation of the TECs' structure. The convex hull was calculated using the `ConvexHull` class from the `SciPy` library. To ensure the boundary encompasses the entire particles, their positions and radii were considered using `NumPy`, a core library for numerical computing in Python that allows for efficient data manipulation and array handling [17]. The convex hull was then visualized using `Matplotlib`, which is used to plot the boundary along with the particles. This approach helped to characterize the overall geometry of the TECs by providing a minimal bounding boundary, useful for understanding their shape and compactness.

3.2.3 2D Density Analysis

The 2D Density Analysis method was employed to examine the spatial concentration of TEC particles. This method reveals areas of higher and lower density, offering insights into how particles are distributed throughout the simulation space. To perform this analysis, particle positions were plotted on a 2D grid, where the density of particles in each region was calculated. The resulting density plot was visualized using `Seaborn`, a Python visualization library that enhances the presentation of statistical data [18]. Seaborn's `kdeplot` function was utilized to generate kernel density estimates, providing a smooth representation of particle concentration, which helped identify patterns of clustering and dispersion across the TECs.

4 Model development

Earlier thymus models represented TECs as multiple particles with a fixed, static position in 3D space. This representation is inadequate for capturing the dynamic morphology of TECs. To address this limitation, I employ a subcellular element approach that divides TECs into two components: the main cell body and its protrusions.

Figure 9: The illustration of main cell body, which is represented by a single particle with a given radius.

The main cell body represents the central portion of the TEC (Fig. 9), serving as the origin for protrusions and as the reference structure during processes like cell division. Protrusions, in contrast, are slender, dynamic extensions that form, retract, move, and physically interact. The model incorporates mechanisms to conserve total cell volume during protrusion dynamics and ensures proper spatial organization through attractive and repulsive interactions between protrusions of the same and neighboring cells. In the following sections, I provide an in-depth breakdown of each component of the protrusion behavior mechanisms that I implemented.

4.1 Simulation setup and initial condition

The simulated domain was set up as a cubical domain with dimensions [50, 50, 50] micrometers. All forces in the model are solved with overdamped dynamics, reflecting the slow, viscous behavior of cellular components in a biological environment [19]. A cutoff distance of 20 micrometers is defined to manage interactions between particles, ensuring computational efficiency and stability.

As an initial condition, I placed a single main cell body at the geometric center of the cubical simulation domain. This placement ensures symmetry and allows protrusions to expand freely in all spatial directions. This main cell body serves as the origin for all subsequent dynamics, including protrusion formation, extension, and retraction. It represents a TEC in its simplest state, devoid of protrusions. Figure 10 depicts the initial environment.

The main cell is modeled as a sphere (Fig. 9) with a radius R_{main} of 2.5 micrometers, consistent with empirical observations of TEC size. Its volume (V_{main}) is calculated as:

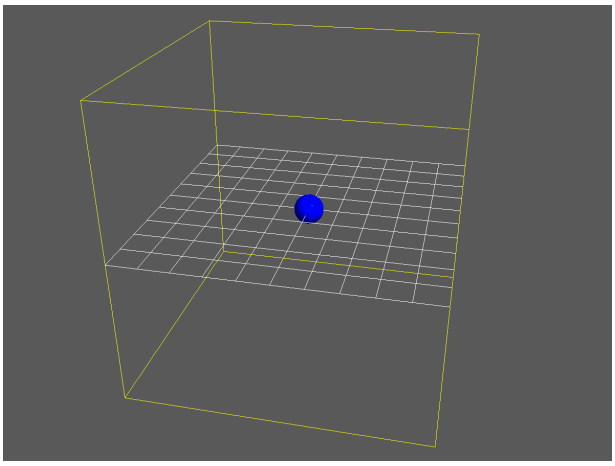


Figure 10: Initial condition of the simulation showing a single TEC cell without protrusions. The white lattice is purely for aesthetics to give a sense of depth.

$$V_{\text{main}} = \frac{4}{3}\pi R_{\text{main}}^3 \tag{1}$$

which provides a baseline reference for volume conservation during protrusion formation.

The main cell body acts as the origin for all protrusions, serving as both a physical anchor and a source of material. Each protrusion forms by extending outward from the surface of the main cell, with its position and direction determined relative to the cell's center. To preserve biological accuracy, the main cell also adjusts its radius and volume dynamically during protrusion formation, maintaining the total system volume as an inherent constraint.

Figure 11: The model conceptualizes a protrusion as a collection of segments, each represented by a particle. The segments are physically connected with bonds (green lines connecting particle centers).

Each protrusion in the model can be composed of multiple segments, each containing several particles that decrease in radius as they extend outward from the main cell (Fig. 11). This setup mimics the thinning appearance of cellular protrusions in nature, where proximal parts (closer to the cell body) are generally larger than distal ones.

The following steps were implemented to simulate protrusion biology:

1. Initial protrusion formation
2. Protrusion extension
3. Protrusion retraction
4. Protrusion interaction
5. Active protrusion movement
6. Main cell splitting (division)

In the following, I will describe the implementation of each of these processes in detail.

4.2 Initial protrusion formation

The process of protrusion formation is summarized in Fig. 12. I implemented time-based triggers to simulate the choice to form new protrusions at regular intervals. An event-driven method periodically

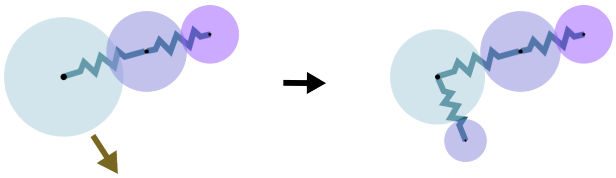


Figure 12: Illustration of protrusion formation. Algorithm for protrusion formation: (1) Choose to create a protrusion, (2) choose a direction, (3) add a particle, (4) bond it to the main cell, (5) shrink the main cell to conserve volume, and (6) adjust bond rest lengths to maintain connections with other protrusions.

checks if the number of protrusions attached to the main cell has reached a predefined limit ($n_{\max} = 20$ protrusions, in this case) to prevent overcrowding. This limit ensures a realistic density of protrusions around the cell, avoiding excessive interactions or unrealistic cellular morphologies. It also reflects biological reality, where the availability of cytoskeletal components and limited intracellular space constrain the number of protrusions a cell can form simultaneously [20].

Once the choice has been made to form a protrusion, the model creates a particle extending outward from the main cell in a random 3D direction. The random direction ensures that protrusions simulate natural, unplanned extensions of cellular structures, but in future versions this could be changed so that protrusions form in response to environmental signals. The model calculates this direction by choosing spherical coordinates θ and ϕ, which determine the azimuthal and polar angles, respectively. These angles are then converted into Cartesian coordinates to yield a unit direction vector $\vec{d}$:

$$\begin{aligned} d_x &= \sin\phi \cdot \cos\theta \\ d_y &= \sin\phi \cdot \sin\theta \\ d_z &= \cos\phi \end{aligned}$$

The position of the new protrusion's first segment $\vec{s}_1$ is calculated based on the main cell's radius R_{main} and the protrusion particle's radius $R_{\text{protrusion}}$, using:

$$\vec{s}_1 = \vec{c} + \vec{d} \cdot (R_{\text{main}} + R_{\text{protrusion}})$$

where $\vec{c}$ is the center position of the main cell. $R_{\text{protrusion}}$ is the initial radius of a newly formed protrusion consisting of one segment.

As each new protrusion is formed, it removes volume from the main cell to conserve the total volume of the system. This is crucial for biological accuracy, as cells do not spontaneously gain volume with new protrusions; instead, they reallocate their existing volume. The model calculates the volume of the spherical main cell V_{main} before protrusion formation as in Eq. 1. Each protrusion particle, treated as a sphere, has its volume $V_{\text{protrusion}}$ calculated by:

$$V_{\text{protrusion}} = \frac{4}{3}\pi R_{\text{protrusion}}^3$$

After adding a protrusion, the main cell's new volume $V_{\text{new main}}$ is:

$$V_{\text{new main}} = V_{\text{main}} - V_{\text{protrusion}}$$

The new radius $R_{\text{new main}}$ of the main cell is recalculated to preserve this volume:

$$R_{\text{new main}} = \left(\frac{3V_{\text{main}}}{4\pi}\right)^{\frac{1}{3}}$$

To anchor protrusions to the main cell, I implemented harmonic bonds using a spring potential model. Each bond connects a protrusion to the main cell and maintains an adjustable rest length based on the sum of the main cell's radius and the protrusion's radius. This bond's potential energy U follows Hooke's Law:

$$U = \frac{1}{2}k(r - r_0)^2 \tag{2}$$

where:

- k is the stiffness constant of the spring, representing bond strength,

- r is the current distance between the main cell and protrusion particle centers, and
- r_0 is the equilibrium rest length of the bond.

This harmonic bonding mechanism ensures that protrusions stay attached to the main cell, allowing flexibility for movement without straying too far, thus simulating the elastic and adaptable nature of cellular extensions.

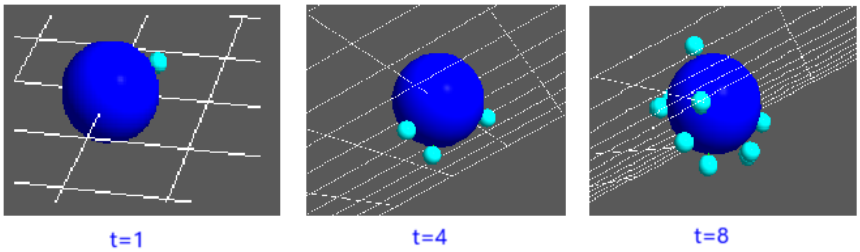


Figure 13: Simulation screenshots showing different time points. Each panel shows the same cell from different view points to highlight new protrusions. As time increases, new protrusions form, each made of only one segment (cyan particles).

4.3 Protrusion extension

A newly-formed protrusion consists of a single segment, represented by one particle. The protrusion extension process involves adding more segments to protrusions. Each subsequent segment is modeled with a particle that decreases in size with distance from the main cell, simulating a tapered extension (Fig. 14).

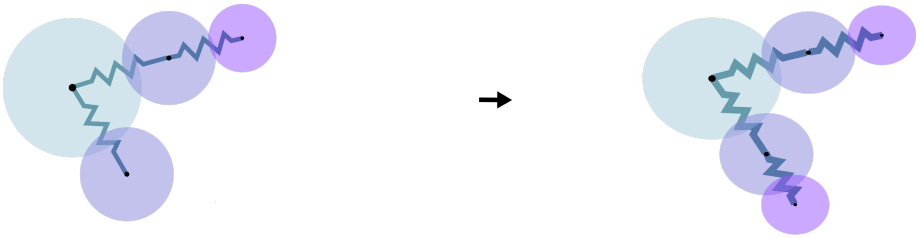


Figure 14: Illustration of protrusion extension. Algorithm for the protrusion extension: (1) The decision is made to extend a given protrusion. (2) A particle is added at the tip and bonded to the previous tip particle of the chosen protrusion; the initial radius of the new tip particle is based on the old tip particle's radius. (3) The main cell's radius is adjusted to conserve total volume; rest lengths of bonds connecting the main particle to all protrusions are rescaled accordingly.

First, the decision is taken to extend an existing protrusion. In this initial implementation, protrusion extension is triggered periodically by time-based events. At regular intervals, the model loops over all protrusions and checks if a protrusion has fewer than $n_{\max,s} = 4$ segments. If it has fewer than $n_{\max,s}$ segments, it grows by appending one new segment. Only one segment is added per iteration to ensure gradual elongation. The position of the new segment is along the direction vector of the protrusion defined as the vector connecting the main cell to the first protrusion segment:

$$\vec{d} = \frac{\vec{c} - \vec{s}_1}{||\vec{c} - \vec{s}_1||}$$

where $\vec{c}$ is the center position of the main cell, and $\vec{s}_1$ is the center position of the first segment of the protrusion, and $||x||$ represents the Euclidean norm to make it a unit vector.

To simulate the tapering observed in biological protrusions, the radius of each subsequent segment decreases. Call the tip segment of a protrusion segment i with radius R_i. Then the radius R_{i+1} of the new tip segment $i + 1$ is calculated as:

$$R_{\mathrm{i+1}} = \max(R_i - r_{\mathrm{taper}}, R_{\mathrm{protrusion,\ min}}).$$

where $r_{\text{taper}} = 0.1$ is the radius reduction per segment, and $R_{\text{protrusion, min}} = 0.2$ prevents the radius from becoming unrealistically small. The position of the new segment is then given by

$$\vec{s}_{i+1} = \vec{d} \cdot (R_{i+1} + R_i)$$

A harmonic bond connects each segment to the previous one, with a rest length equal to $(R_{i+1} + R_i)$ with $k = 1$ (Eq. 2).

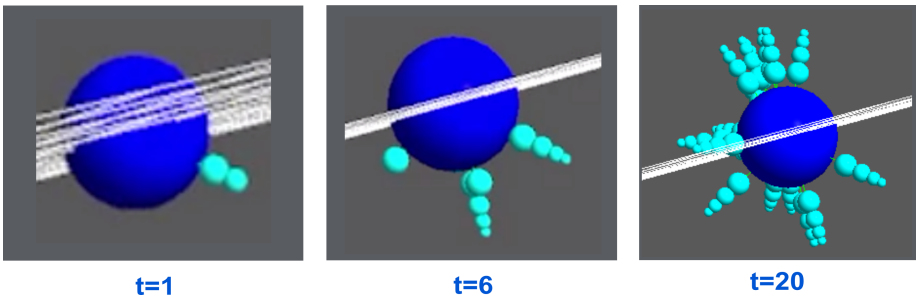


Figure 15: Simulation screenshots showing different time points. Each panel shows the same cell from different view points to highlight that new protrusions form and extend.

As new segments are added, their volume is deducted from the main cell to maintain total volume conservation. For a segment with radius R_{new}, the volume is:

$$V_{\text{segment}} = \frac{4}{3}\pi R_{\text{new}}^3$$

The main cell adjusts its radius based on the reduced volume:

$$V_{\text{new main}} = V_{\text{initial main}} - V_{\text{segment}} \tag{3}$$

$$R_{\text{new main}} = \left(\frac{3V_{\text{new main}}}{4\pi}\right)^{\frac{1}{3}} \tag{4}$$

This mechanism ensures that the addition of protrusions does not violate the conservation of mass or volume.

4.4 Protrusion retraction

In real life, cells can choose to extend new protrusions or retract existing ones. TECs undergo significant morphological changes during thymic development and in response to stress or aging [6]. These changes include the elongation and retraction of cytoplasmic projections, which influence interactions with thymocytes. To model protrusion retraction, I chose to have protrusions retract by immediately deleting the segments one by one, starting from the tip of the protrusion. This process continues until the entire protrusion is fully retracted. This process can repeat until the entire protrusion disappears. This behavior (Fig. 16) ensures that the system remains dynamic, with old protrusions being removed to make way for new ones.

The volume of the removed particle, calculated using the formula for the volume of a sphere $\left(V = \frac{4}{3}\pi R^3\right)$, is then added back to the main cell's volume. As the volume of the main cell increases, its radius is updated to accommodate this volume change using the formula in Eq. 4, ensuring volume conservation.

The bond rest lengths between all involved particles is adjusted based on the current radius of each particle pair. The new rest length is defined as the sum of the radii of the bonded particles. These approaches enable the model (Fig. 17) to simulate protrusion retraction effectively while conserving total cell volume and maintaining the stability of cellular bonds. The combined mechanisms of particle removal and shrinkage offer flexibility in mimicking biological retraction behaviors.

4.5 Protrusion interactions

To accurately represent the spatial organization of protrusions, the model includes mechanisms to prevent different protrusions from intersecting with each other (Fig. 18). The model achieves this through a repulsive force between segments of different protrusions. This repulsion is defined by an interaction

Figure 16: Illustration of protrusion retraction. The algorithm for protrusion retraction: (1) Choose a protrusion to be decreased in size. (2) The tip segment of this protrusion decreases in size. (3) If this decrease makes the tip segment radius less than a certain minimum radius threshold, the number of particles decreases as the tip particle and all of its bonds are removed. Then, the volume of the tip particle is reincorporated into the main cell, which grows to compensate.

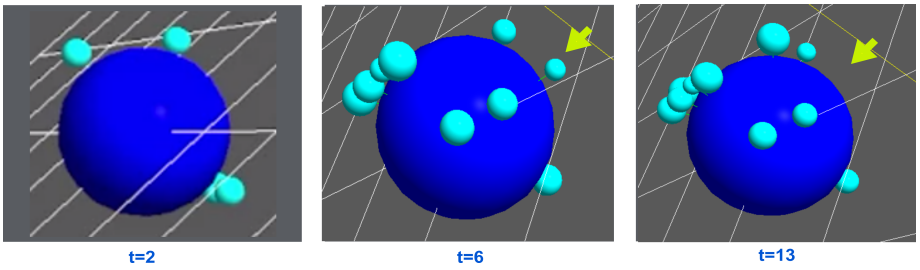


Figure 17: Simulation screenshots at various time points. Each panel displays the same cell from different angles to emphasize the formation of new protrusions, chose one tip protrusion randomly (t=6), and then delete that chosen protrusion (t=13) shown by a yellow arrow.

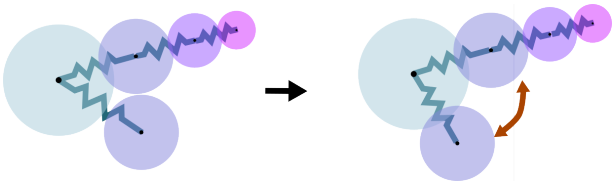


Figure 18: Segments of different protrusions interact via a repulsive potential (Morse potential) to prevent intersection, with repulsion specific to different protrusions and absent within the same protrusion.

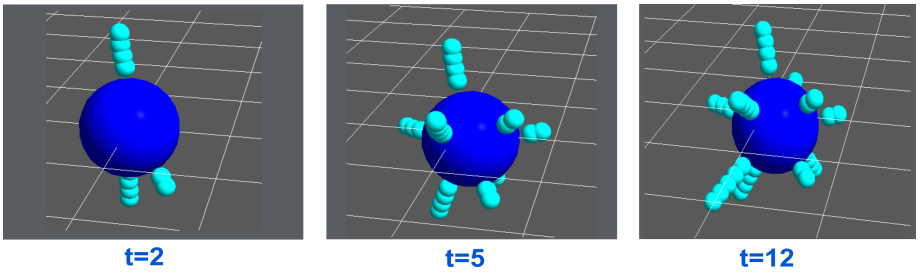


Figure 19: Simulation screenshots showing different time points. Each panel shows the same cell from different view points to highlight the interaction between different protrusions that enable to dynamically redistribute on the surface of the main cell.

potential. Here, I chose to implement the Morse potential, which is widely used in molecular dynamics to simulate attractive or repulsive forces between particles.

The Morse potential allows the model to control the distance at which repulsion between segments becomes effective. Mathematically, the Morse potential is defined as:

$$U(r) = d\left(1 - e^{-a(r - r_{0,\text{Morse}})}\right)^2 \tag{5}$$

Where $U(r)$ is the interaction potential at a distance r, d is the depth of the potential well (related to the strength of attraction), a scales the width of the potential and $r_{0,\text{Morse}}$ is the position of the minimum. In

this model, the potential is tuned to be repulsive by setting the interaction depth d to a negative value, ensuring that particles experience a force pushing them apart as they approach each other closer than $r_{0,\text{Morse}}$. This configuration effectively prevents overlap or clustering of protrusions while allowing them to move independently within the defined 3D space.

Importantly, the repulsive forces only act between segments of different protrusions; segments within the same protrusion do not repel each other. Instead, they are connected by harmonic bonds (Eq. 2) and are allowed to overlap.

Together, these interaction mechanisms (repulsive forces for inter-protrusion spacing, harmonic bonds for intra-protrusion cohesion) allow the model (Fig. 19) to simulate realistic protrusion distribution.

4.6 Active protrusion movement

Cells use protrusions to probe their surroundings. Long thin protrusions are generally made of straight F-actin polymers that elongate and shrink mainly in the direction of the protrusion [20]. These dynamic structures, including filopodia and tunneling nanotubes, play crucial roles in cellular functions, ranging from environmental sensing and cell migration to intercellular communication and tissue patterning [20]. To model this process, I introduced forces that alternately pull or push along the direction of the protrusion (Fig. 20).

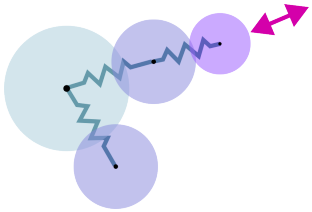


Figure 20: Illustration of protrusion movement. The movement is driven by forces on the tip cells. Forces are only allowed to act along the direction vector of the protrusion.

Forces are applied to all of the protrusions. The magnitude of the force applied to all protrusion segments is given by the parameter $f = 1.5$, and its direction is along the protrusion:

$$\vec{F} = w \cdot f \cdot \frac{\vec{c} - \vec{s}_1}{||\vec{c} - \vec{s}_1||}$$

where w is either equal to 1 or -1, $\vec{c}$ is the center position of the main cell, and $\vec{s}_1$ is the center position of the first segment of the protrusion, and $||x||$ represents the Euclidean norm to make it a unit vector. The value of w is chosen to alternate between 1 and -1 at regular intervals to simulate oscillating motion.

4.7 Main cell splitting (division)

To implement cell division, I created a time-triggered event. At regular intervals, this event decides if a cell can divide into two daughter cells with a probability $\Pr[\text{division}] = 0.1$. Before division, all protrusions are retracted to ensure the main cell returns to its original spherical shape. The division process removes all protrusions by destroying the particles associated with the protrusions and clearing their references. The main cell's radius is reset to its original value.

When a cell $[i]$ divides, its two daughter cells $[i, 1]$ and $[i, 2]$ are created at positions randomly displaced from the original cell's position. The position of the daughter cells $\vec{c}_{i,j}$ is determined using a random unit vector $\vec{r}$ specified with spherical coordinates, and a specified distance (l) from the original cell $\vec{c}_i$:

$$\vec{c}_{i,j} = \vec{c}_i + l \cdot \vec{r},$$

where $j \in [1, 2]$. These new main cells inherit half of the original cell's volume, ensuring volume conservation. After the division, the original cell is destroyed, ensuring an accurate cell count and preventing overlapping or duplication. The daughter cells begin without any protrusions, allowing them to initialize their growth and interactions independently in the simulation.

At each time step, the simulation in Fig. 21 randomly decides whether a main cell will grow new protrusions or divide. The model tracks these processes while enforcing volume conservation and ensuring all interactions (bonds and protrusions) follow predefined physical rules.

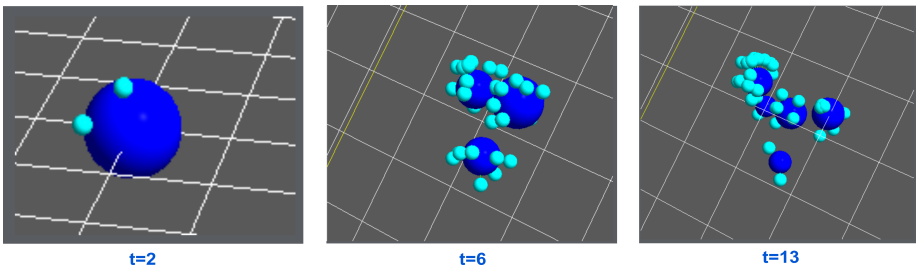
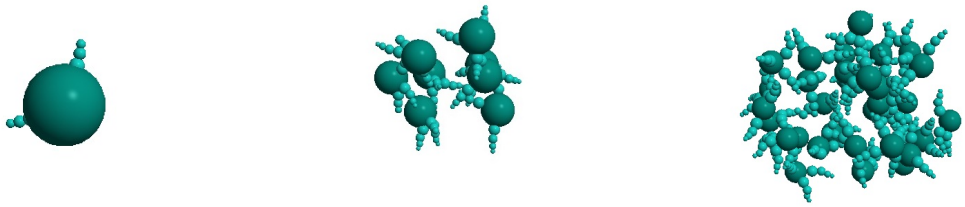


Figure 21: Simulation screenshots illustrate various time points, with each panel displaying the same cells from different perspectives. These views highlight the process of main cell splitting, where the main cell divides into two by time-triggered event, and continue to make division. However, the protrusions do not reduce in size despite the main cell shrinking during division.

4.8 The whole model



(a) (b) (c)

Figure 22: The snapshots of my model at t=75, t=300, and t=500, respectively

The model integrates all previously described mechanisms (protrusion formation, extension, retraction, interaction, movement, and main cell splitting) into a unified framework specifically designed to simulate the dynamic behavior of TECs within the thymus. As covered in previous sections, these mechanisms were explained individually, from protrusion formation to main cell splitting. With these processes integrated, the model now simulates the dynamic interaction and spatial organization of TECs. In terms of visual representation, I modified the color scheme of the model in Fig. 22 to align more closely with existing literature, such as the model of TECs in Medaka fish as described in [2]. Specifically, I changed the color of the main cells (initially blue) to dark turquoise and the protrusions (initially light blue) to light turquoise, ensuring consistency with existing TEC representations.

To explore the model's behavior, I varied three key parameters:

- Number of protrusions (n_p) $\rightarrow[3; 6; 9]$.
- Protrusion length or the maximum number of segments per protrusion ($n_{max,s}$) $\rightarrow[2; 4; 6]$.
- Morse repulsion constant (d) $\rightarrow[-0.075; -0.1; -0.25]$.

These parameters were selected because they directly influence the spatial dynamics and interactions of TECs, which are essential for simulating realistic thymic behavior. In the thymus, cortical thymic epithelial cells (cTECs) form dense clusters that create specialized niches for thymocyte selection, while medullary thymic epithelial cells (mTECs) require a more dispersed organization to efficiently present self-antigens [21]. By adjusting protrusion number and repulsion strength, we aim to explore how these factors contribute to TEC organization and movement in a way that may reflect these biological patterns. The bracketed values indicate the parameter variations. Each parameter was adjusted individually, without combining multiple variations, to isolate its specific effect on the system. This approach ensures a clearer understanding of how each factor shapes TEC spatial distribution. The next section will provide a short analysis of the model's behavior and present the results.

5 Model analysis and results

To address the central question of **how the physical scaffold formed by TECs develops**, I analyzed the spatial and temporal distribution of TECs under various parameter settings. The formation of thymic structure is driven by biological processes like protrusion extension from TECs and cell-to-cell mechanical interactions.

For the analysis, I ran 90 simulations over 500 steps with different parameter variations. Each 5 periods of the total simulation time generated a detailed JSON file containing data on the positions and types of main cells and protrusions. From these files, I extracted the x, y, and z coordinates of the particles for further spatial and quantitative analysis. The spatial structure and dynamics of the TEC population were analyzed using multiple methods. Voronoi diagrams helped assess spatial organization, distinguishing between main cells and protrusions. Density plots were used to visualize population trends over time, revealing changes in the distribution of particles. Volume analysis was conducted by comparing the total volume occupied by particles to the convex hull, providing insight into how efficiently TECs utilize the available space. I controlled parameters like the randomness (variability in the timing of cell divisions), the number of protrusions growing from each main cell, protrusion length, and Morse repulsion constant to see how these factors influenced spatial structure and population dynamics. The results were compared using the same suite of analytical tools, which allowed for a detailed understanding of how different biological processes impact TEC behavior.

5.1 Thymic structure and TEC distribution

This section focuses on how variations in the number of protrusions, protrusion length, and Morse repulsion influence the morphology of the thymic scaffold and the spatial distribution of TECs.

5.1.1 Effect of protrusion number and repulsion on thymic morphology

The morphology of the simulated thymus is influenced by the balance between cellular protrusions and repulsion forces acting on TECs. Figure 23 shows how these parameters shape thymic structure over time.

In the reference condition (a-b-c), with 6 protrusions, length = 4, Morse repulsion = -0.1, and without randomness or $r = 0$, the TECs form a steadily expanding yet cohesive structure. The convex hull encloses most cells, and the growth curve (panel j) shows a gradual increase in TEC count, reaching 30 cells at t=500.

Reducing the number of protrusions to 3 (d-e-f) results in fewer outward extensions, but the overall convex hull size remains similar. TECs appear more locally clustered, though cell count over time (panel k) does not significantly differ from the reference.

Weakening Morse repulsion to -0.075 (g-h-i) leads to a denser TEC cluster with a smaller convex hull, indicating stronger local adhesion and reduced spatial dispersion. Despite this, the growth curve (panel l) remains comparable to the other conditions, suggesting that population size is not significantly affected.

These findings suggest that protrusions primarily influence the extent of structural expansion, while repulsion regulates TEC clustering and dispersion. Future refinements, such as incorporating angle bonds or adaptive adhesion or repulsion forces that change based on local TEC density, could improve the model's ability to capture the complexity of thymic architecture.

5.1.2 Effect of randomness on thymic dynamics

Randomness in particle dynamics plays a significant role in shaping the spatial behavior and organization of TECs within the thymic scaffold. In this model, randomness refers to the variability in the timing and direction of biological processes, such as the division of main cells and the growth or retraction of protrusions. This temporal and directional variability introduces fluctuations in the cell dynamics and influences the overall spatial structure of the TECs. A closer look at different time frames in Fig. 24 reveals how varying levels of randomness influence the interplay between TEC clustering, dispersion, and structural stability. The red lines in each figure represent experimental data, used to compare with the model's results. The model's predictions do not fully align with the experimental data, suggesting overestimations or underestimations in certain spatial distributions. This discrepancy underscores the need for further adjustments to the model to better match real-world observations.

At early time points (t<100), TECs are tightly packed, occupying most of the convex hull volume with minimal variability. As time progresses (100<t<250), fluctuations in spatial distribution emerge due to randomness in cell division timing, protrusion angles (the directions in which protrusions extend), and

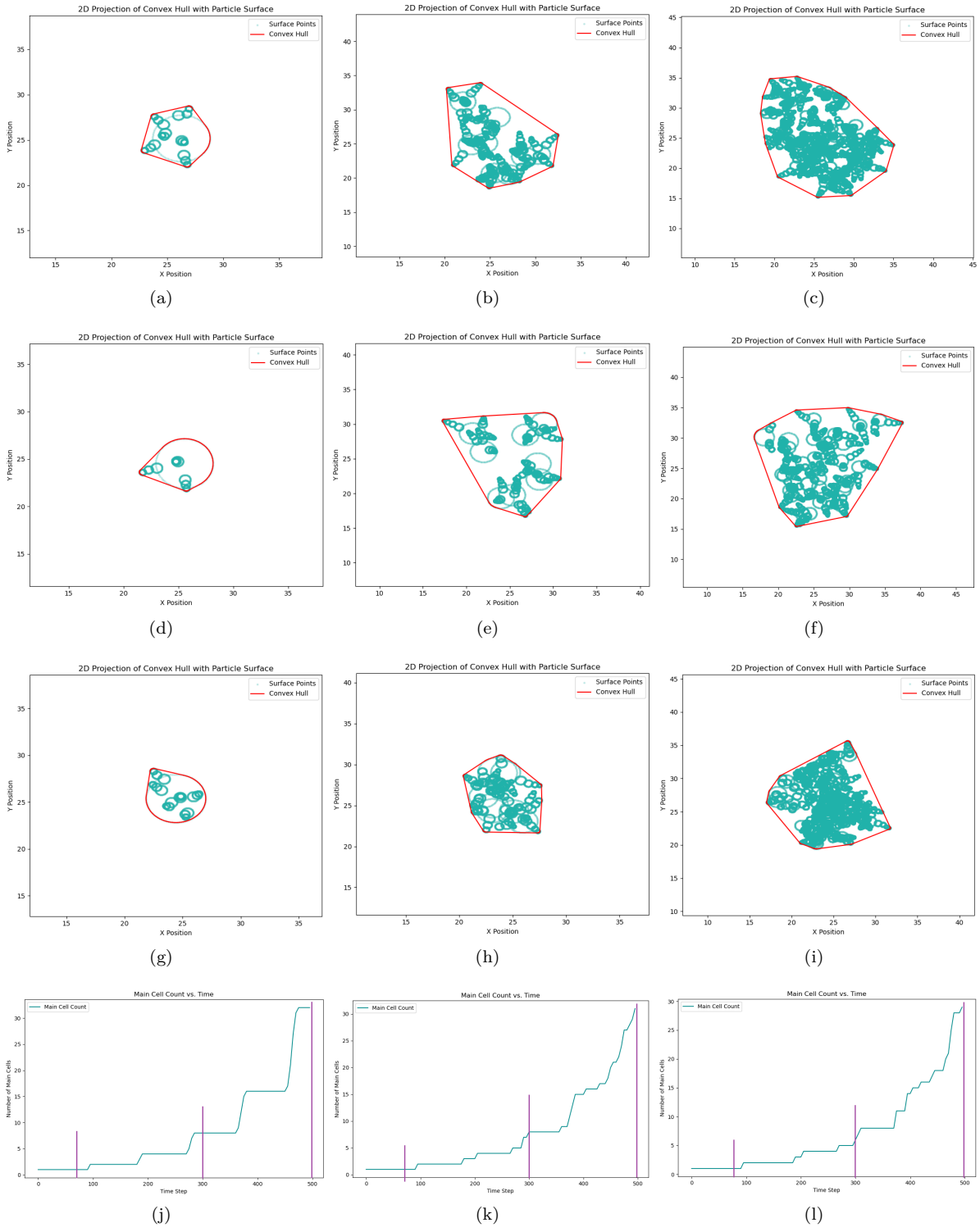


Figure 23: 2D projections of the convex hull of the model at three time points (t = 75, 300, and 500) under different parameter variations. The first row (a-b-c) serves as a reference without randomness or $r = 0$, showing the model with number of protrusions = 6, length = 4, and Morse repulsion = -0.1. The second row (d-e-f) explores the effect of reducing the number of protrusions to 3, while the third row (g-h-i) examines the impact of weakening Morse repulsion to -0.075, both following the same time progression as the reference row. The red convex hull outlines the thymus gland, while the turquoise-colored particles represent individual elements within the model. The last row (j-k-l) contains line charts showing the increase in the TEC number (as main cells in the model) over time (t = 0–500), with purple vertical lines marking t = 75, 300, and 500 as reference points. This structured layout allows for direct comparison of how changes in protrusion number and interaction forces influence system evolution while maintaining consistency in time points for clear temporal analysis.

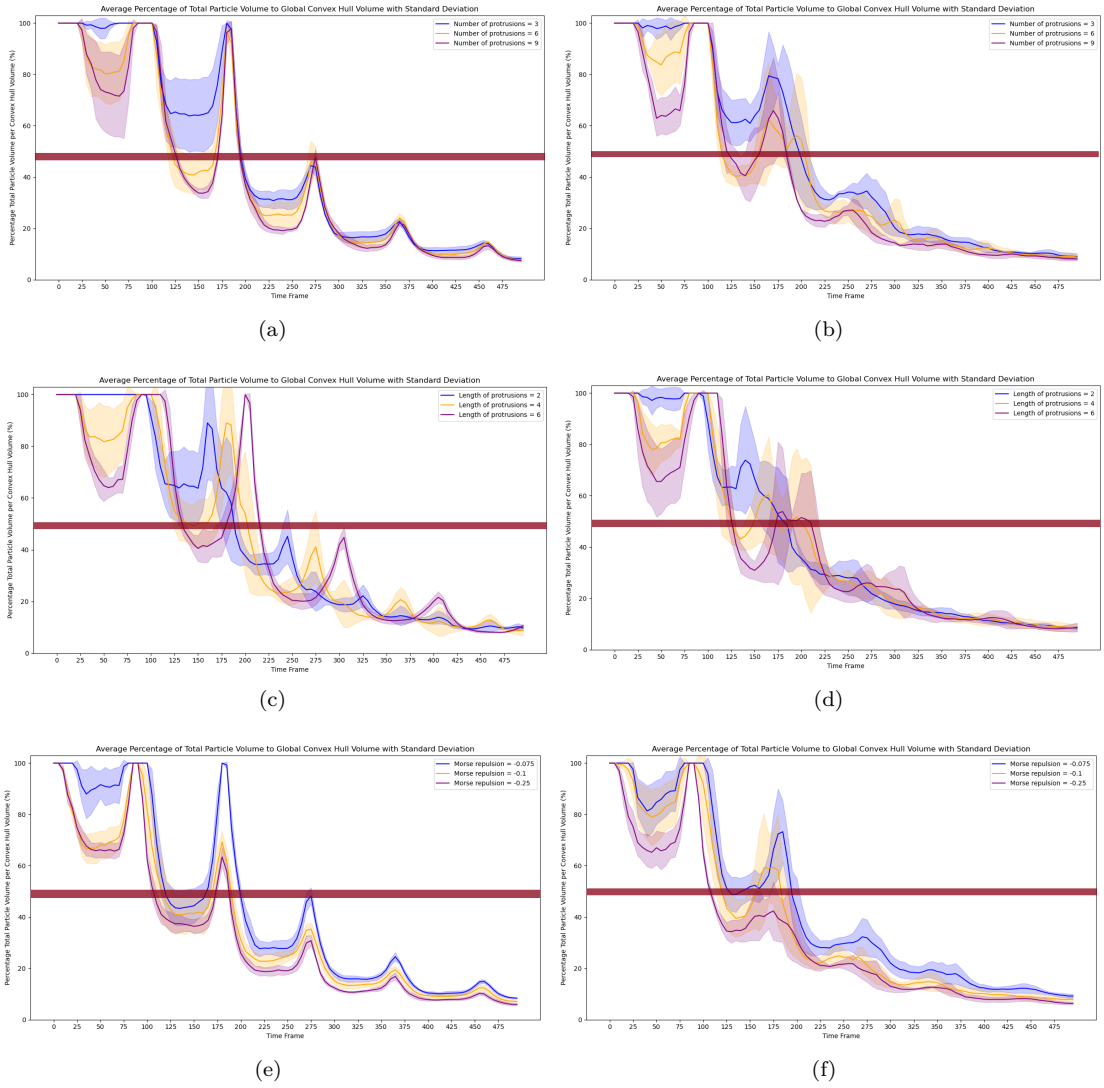


Figure 24: Effect of protrusion number (a-b), protrusion length (c-d), and Morse repulsion (e-f) on TEC spatial distribution, with and without randomness. Each plot shows the average percentage of TEC volume within the convex hull over time, with shaded regions representing standard deviation. The rows correspond to different parameters: protrusion number, protrusion length, and Morse repulsion, while the columns compare randomness = 0 (deterministic) and randomness = 5 (stochastic variation). The horizontal red lines in each figure represent the experimental percentage data at $47 \pm 3\%$ over time [3], providing a point of comparison for the simulated TEC dynamics.

TEC interactions. These fluctuations create oscillations in the TEC volume fraction within the convex hull, driven by the cyclic process of protrusion retraction and cell division: when protrusions retract, only the main cell remains, reducing the overall occupied volume. This is followed by cell division, where each new TEC extends protrusions, increasing the volume fraction. This cycle of retraction, division, and protrusion growth drives the dynamic changes in TEC spatial organization over time.

By t = 400, randomness has a stronger effect, with TEC clusters continuously forming and dissolving due to repulsion forces and variable protrusion angles. The irregular timing of cell division further amplifies these fluctuations, resulting in a highly dynamic and unstable spatial organization where TECs do not maintain fixed clusters. As the simulation progresses (t>400), the convex hull volume stabilizes, but TEC density remains lower than in earlier stages. This could indicate that the system has reached a minimum dispersion level beyond which further compaction is not possible.

The degree of randomness critically determines the behavior of the system, as evidenced by the distinct structural patterns that emerge under varying conditions. When randomness is high (r=5), simulations display significant fluctuations and variability across runs, highlighting the unpredictability in TEC move-

ment and organization. In this case, the TECs exhibit more varied configurations, with a wider range of spatial distribution observed. On the other hand, when randomness is low (r=0), the TEC distribution remains more consistent across simulations, but they are still dispersed across the convex hull, although with less variation. This suggests that even under low randomness, the TECs do not form tight clusters but maintain a relatively spread-out pattern across the simulation space.

Interestingly, the interplay between randomness and repulsion further accentuates these effects. High randomness coupled with strong repulsion (M=−0.25) destabilizes the structure, while low randomness and weak repulsion (M=−0.075) promote clustering and stability. This balance is critical for maintaining thymic cohesion and functionality. Excessive repulsion can lead to fragmentation, while insufficient repulsion may result in overcrowding, both of which hinder the thymus's ability to support T-cell development effectively.

5.2 Spatiotemporal dynamics of TECs

The development of TECs is a highly dynamic process [22], yet the precise spatial and temporal changes that occur during their proliferation, migration, and interaction with thymocytes remain poorly understood [21]. One of the key challenges is determining how TECs transition from an initially scattered state into a highly organized network essential for T-cell development [22]. Understanding this transformation requires a detailed analysis of TEC distribution, clustering behavior, and structural adaptations over time.

To explore this, I examined TEC growth and clustering patterns using 3D representations (Fig. 25 a, b, c). At early stages ($t = 75$, Fig. 25 a), TECs appear sparsely distributed. As time progresses ($t = 300$, Fig. 25 b), TEC clustering increases, forming denser groups. By the later stages ($t = 500$, Fig. 25 c), TECs form a visibly interconnected network. Since the model incorporates both proliferation and a limited degree of migration, these structural changes are expected as cells divide and spread. However, the extent to which migration contributes to TEC organization requires further analysis with additional simulations.

Next, I analyzed Voronoi diagrams to gain an initial qualitative understanding of spatial TEC organization at different time points (Fig. 25 d, e, f). This analysis was performed on a single simulation, selected to illustrate a representative pattern of TEC arrangement. At early stages ($t = 75$, Fig. 25 d), TECs are widely spaced, with large and irregular Voronoi cells reflecting the low cell density. As proliferation progresses ($t = 300$, Fig. 25 e), Voronoi cells become more uniform, suggesting a shift toward a more evenly distributed TEC network. By $t = 500$ (Fig. 25 f), TECs exhibit a more compact arrangement with smaller Voronoi cells, indicative of increased spatial proximity among cells. While this visualization provides useful insights, a more comprehensive analysis across multiple simulations is needed to quantify these patterns systematically.

Beyond spatial organization, I also examined how TEC concentration evolved over time using 2D density plots (Fig. 25 g, h, i). At early stages ($t = 75$, Fig. 25 g), TECs display localized clustering, with only a few high-density regions. As the system develops ($t = 300$, Fig. 25 h), distinct dense areas emerge, indicating a transition toward a more structured arrangement. By $t = 500$ (Fig. 25 i), the density plot suggests a widespread and interconnected TEC distribution. However, due to the limited number of cells in the simulation and the absence of additional regulatory factors such as thymocyte interactions, the observed patterns should be interpreted with caution.

This exploratory analysis highlights how TECs transition from an initially dispersed state into a structured network, aligning with known biological thymic organization [21]. However, because these observations are based on a single simulation, further quantitative analysis is needed to confirm whether these trends hold across different parameter settings and replicates. Future improvements, including increasing TEC numbers and integrating additional biological constraints, will be discussed in the next section.

6 Discussion

In this Master thesis, I developed a subcellular-element model of dynamic TECs including the processes of protrusion formation, physical interaction, retraction, extension, and cell division as a central aspect of TEC proliferation, critical for the expansion and maintenance of the thymus. Together, these mechanisms allow the model to simulate complex behaviors of cellular protrusions. This approach is particularly valuable for studying cellular morphology and function within a tissue such as the thymus, where dynamic interactions between cells and their micro-environment are crucial for tissue organization and development.

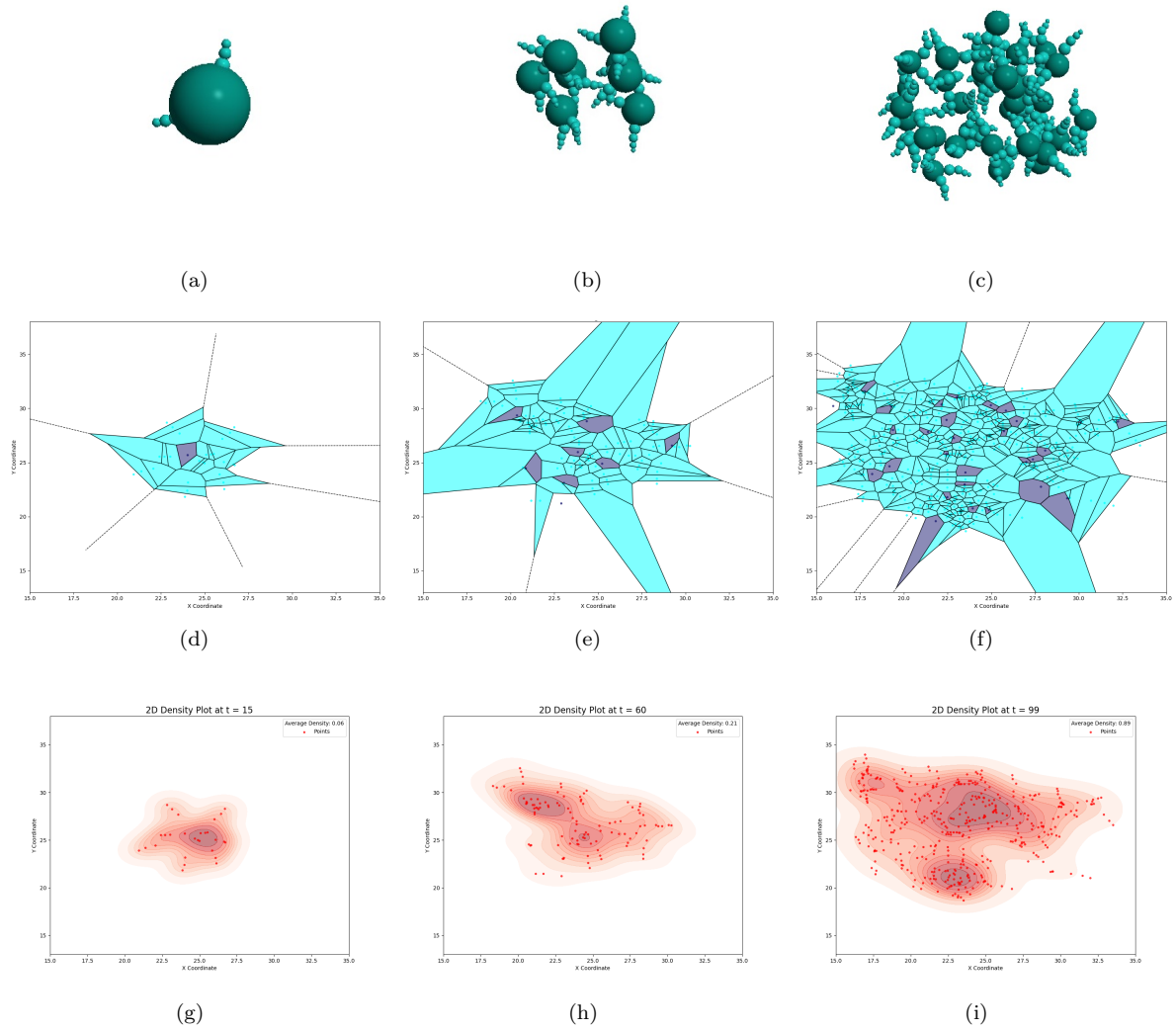


Figure 25: Visual analysis of particle distribution and density evolution over time, including my 3D model snapshots, Voronoi diagrams, and 2D density plots. The top row (a–c) shows snapshots of my model at t=75, t=300, and t=500, respectively. At t=75, a main cell with protrusions is visible, with the observed side featuring two protrusions, each consisting of four segments. As time progresses (t=300 and t=500), the model grows according to the previously described algorithm, with an increasing number of particles and more complex structural organization. The middle row (d–f) presents Voronoi diagrams corresponding to each snapshot, illustrating the spatial organization of particles. The bottom row (g–i) displays 2D density plots, highlighting the concentration and distribution of particles at the same time steps.

The use of the Morse potential in this model is effective for controlling intercellular spacing and preventing overlap between TECs. However, its application may oversimplify the intricate balance of forces in a biological setting. Exploring alternative interaction potentials, such as Lennard-Jones (LJ), could offer a more biologically realistic representation. The LJ potential, with its steep repulsion at short distances and mild attraction at intermediate distances, may better reflect the mechanical constraints and adhesive interactions between TECs. Adhesion forces play a significant role in the spatial organization and stability of cell clusters [22], and their inclusion in the model could enhance the realism of TEC interactions. Implementing LJ potential and adhesion forces in future iterations could help refine the balance between TEC aggregation and dispersion, offering a closer approximation of the thymic structural dynamics.

Aging in the thymus is primarily associated with the retraction of TEC projections, particularly in

cortical thymic epithelial cells (cTEC), rather than a direct loss of TECs themselves [22]. The process of cTEC projection contraction reduces the complexity and surface area of these cells, which in turn decreases the thymic cortex volume and its ability to support thymocyte development. While the total number of TECs may not significantly decrease with age, the shrinkage of their projections reduces the available surface area for interactions with thymocytes, impacting T-cell maturation and selection efficiency. This is in contrast to the current model, which is limited to TEC division and fails to account for the morphological changes that occur with aging. Specifically, at t=500, the model reaches its maximum simulation time, with approximately 30 TECs, a significant underrepresentation of the thymic architecture. A more accurate simulation would require increasing the TEC population to around 550 TECs, as indicated by the static model of [3], which used 55 TECs to represent 10% of the thymus. To address this, future iterations should implement a retraction algorithm, regulating TEC projection shrinkage based on thymocyte density and incorporating explicit thymocyte dynamics. These adjustments would better capture the morphological and functional changes of thymic aging, enhancing the model's ability to simulate both thymic development and degeneration.

In addition to the processes described above, I planned to add angular bonds between the tip segment of a protrusion and the main cell to regulate protrusion stability and enable curved or concave movements (Fig. 26). While the current model incorporates dynamic protrusions, they remain mostly straight due to the absence of angle-bound forces. Introducing angular bonds would allow protrusions to bend or fold inward, potentially reducing the overall convex hull volume. As a result, this could increase the TEC volume fraction within the convex hull, leading to a spatial distribution that better reflects the interconnected TEC network observed *in vivo* [22]. However, due to time constraints, I was not able to complete the implementation. Future work should incorporate this mechanism to evaluate its impact on TEC spatial organization.

Disruptions in TEC protrusions and cell division are closely linked to pathological conditions, including autoimmune diseases, infections, and aging [23]. TEC protrusions facilitate interactions with thymocytes and maintain tissue structure, while cell division is critical for sustaining TEC populations and thymic regeneration. Aberrant signaling, involving pathways like Notch, Wnt, and Foxn1, can lead to defective TEC morphology, disrupted thymic architecture, and immune dysfunction. For instance, mutations in Foxn1 or age-related declines in its expression are associated with thymic atrophy, impaired thymocyte interactions, and reduced TEC proliferation [24]. In addition, changes to the cTEC network may affect positive selection, a process critical for the survival and maturation of T cells responsive to major histocompatibility complex molecules, which are essential for antigen presentation [24]. Similarly, impaired cell splitting can reduce TEC density, weakening the thymic niche and contributing to thymic atrophy or disease states such as leukemia [3]. By incorporating equations that describe TEC protrusions or TEC cell division, my model provides a foundation for exploring the quantitative and qualitative effects of TEC dynamics on thymus biology.

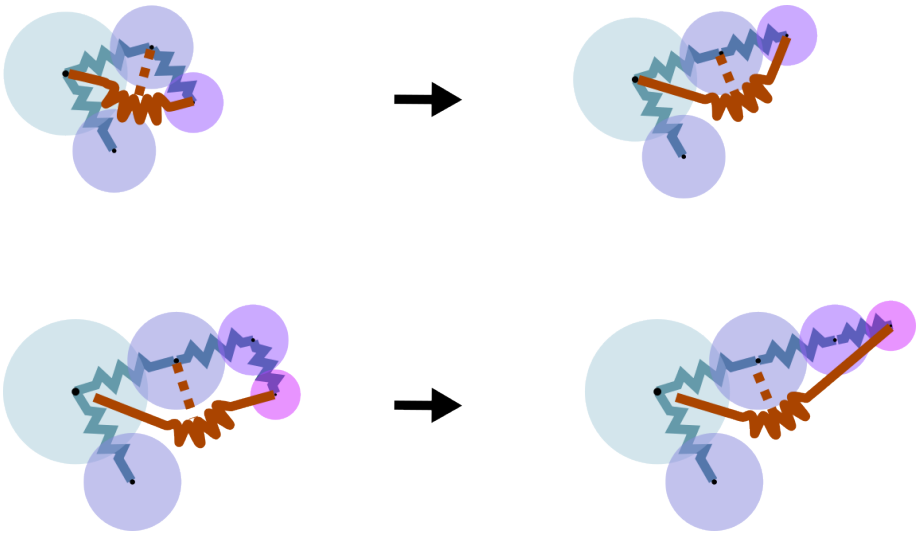


Figure 26: Illustration of the angular bonds between the base, tip segments, and main cell generate restoring forces to keep the protrusion straight, requiring bond creation for protrusions with at least two segments and updates with each new extension.

In this work, I successfully implemented main cell splitting and protrusion interactions using the Morse potential. Meanwhile, the model currently lacks thymocytes, limiting its ability to explore interactions between TECs and thymocytes, as well as processes like cytokine diffusion and Notch signaling. Future work should focus on incorporating thymocytes and molecular signaling pathways to better understand TEC function in immune development. The model is based on *in vivo* observations of TEC behavior, but available data is limited to the percentage of TECs within the thymus, with no numerical data provided for direct application. As more concrete data becomes available, future iterations can refine and validate the model more accurately.

Bibliography

[1] S. Fujimori and I. Ohigashi, "The role of thymic epithelium in thymus development and age-related thymic involution," *The Journal of Medical Investigation*, vol. 71, no. 1.2, pp. 29–39, 2024.

[2] N. Aghaallaei, A. M. Dick, E. Tsingos, D. Inoue, E. Hasel, T. Thumberger, A. Toyoda, M. Leptin, J. Wittbrodt, and B. Bajoghli, "$\alpha\beta/\gamma\delta$ T cell lineage outcome is regulated by intrathymic cell localization and environmental signals," *Science Advances*, vol. 7, no. 29, p. eabg3613, 2021.

[3] E. Tsingos, A. M. Dick, and B. Bajoghli, "Altered thymic niche synergistically drives the massive proliferation of malignant thymocytes," *eLife*, vol. 13, 2024.

[4] M. Ruiz Pérez, P. Vandenabeele, and P. Tougaard, "The thymus road to a T cell: migration, selection, and atrophy," *Frontiers in Immunology*, vol. 15, p. 1443910, Aug. 2024.

[5] N. Yayon, V. R. Kedlian, L. Boehme, C. Suo, B. T. Wachter, R. T. Beuschel, O. Amsalem, K. Polanski, S. Koplev, E. Tuck, E. Dann, J. Van Hulle, S. Perera, T. Putteman, A. V. Predeus, M. Dabrowska, L. Richardson, C. Tudor, A. Y. Kreins, J. Engelbert, E. Stephenson, V. Kleshchevnikov, F. De Rita, D. Crossland, M. Bosticardo, F. Pala, E. Prigmore, N.-J. Chipampe, M. Prete, L. Fei, K. To, R. A. Barker, X. He, F. Van Nieuwerburgh, O. A. Bayraktar, M. Patel, E. G. Davies, M. A. Haniffa, V. Uhlmann, L. D. Notarangelo, R. N. Germain, A. J. Radtke, J. C. Marioni, T. Taghon, and S. A. Teichmann, "A spatial human thymus cell atlas mapped to a continuous tissue axis," *Nature*, vol. 635, pp. 708–718, Nov. 2024.

[6] M. K. Lagou, D. G. Argyris, S. Vodopyanov, L. Gunther-Cummins, A. Hardas, T. Poutahidis, C. Panorias, S. DesMarais, C. Entenberg, R. S. Carpenter, *et al.*, "Morphometric analysis of the thymic epithelial cell (TEC) network using integrated and orthogonal digital pathology approaches," *bioRxiv*, 2024.

[7] F. P. Assen, "Multitier mechanics control stromal adaptations in the swelling lymph node," *Nature Immunology*, vol. 23, 2022.

[8] I. Hess and T. Boehm, "Intravital Imaging of Thymopoiesis Reveals Dynamic Lympho-Epithelial Interactions," *Immunity*, vol. 36, pp. 298–309, Feb. 2012.

[9] G. Anderson, E. J. Cosway, K. D. James, I. Ohigashi, and Y. Takahama, "Generation and repair of thymic epithelial cells," *Journal of Experimental Medicine*, vol. 221, p. e20230894, Oct. 2024.

[10] P. A. Robert, H. Kunze-Schumacher, V. Greiff, and A. Krueger, "Modeling the Dynamics of T-Cell Development in the Thymus," *Entropy*, vol. 23, p. 437, Apr. 2021.

[11] H. Souza-e Silva, W. Savino, R. A. Feijóo, and A. T. R. Vasconcelos, "A Cellular Automata-Based Mathematical Model for Thymocyte Development," *PLOS One*, vol. 4, p. e8233, Dec. 2009.

[12] J. B. Beltman, A. F. Marée, J. N. Lynch, M. J. Miller, and R. J. De Boer, "Lymph node topology dictates T cell migration behavior," *The Journal of Experimental Medicine*, vol. 204, pp. 771–780, Apr. 2007.

[13] T. J. Sego, J. P. Sluka, H. M. Sauro, and J. A. Glazier, "Tissue Forge: Interactive biological and biophysics simulation environment," *PLOS Computational Biology*, vol. 19, p. e1010768, Oct. 2023.

[14] J. Metzcar, Y. Wang, R. Heiland, and P. Macklin, "A review of cell-based computational modeling in cancer biology," *JCO Clinical Cancer Informatics*, vol. 2, pp. 1–13, 2019.

[15] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[16] E. Bisong and E. Bisong, "Matplotlib and seaborn," *Building machine learning and deep learning models on google cloud platform: A comprehensive guide for beginners*, pp. 151–165, 2019.

[17] T. E. Oliphant *et al.*, *Guide to numpy*, vol. 1. Trelgol Publishing USA, 2006.

[18] M. Waskom, O. Botvinnik, P. Hobson, J. Warmenhoven, J. B. Cole, Y. Halchenko, J. Vanderplas, S. Hoyer, S. Villalba, E. Quintero, *et al.*, "seaborn: v0. 6.0 (june 2015)," *Zenodo*, 2015.

[19] H. C. Berg, *Random walks in biology*. Princeton University Press, 1993.

[20] S. Belian, O. Korenkova, and C. Zurzolo, "Actin-based protrusions at a glance," *Journal of Cell Science*, vol. 136, no. 22, p. jcs261156, 2023.

[21] L. Sun, H. Li, H. Luo, and Y. Zhao, "Thymic epithelial cell development and its dysfunction in human diseases," *BioMed research international*, vol. 2014, no. 1, p. 206929, 2014.

[22] T. Venables, A. V. Griffith, A. DeAraujo, and H. T. Petrie, "Dynamic changes in epithelial cell morphology control thymic organ size during atrophy and regeneration," *Nature communications*, vol. 10, no. 1, p. 4402, 2019.

[23] N. R. Manley, E. R. Richie, C. C. Blackburn, B. G. Condie, and J. Sage, "Structure and function of the thymic microenvironment," *Front Biosci (Landmark Ed)*, vol. 16, no. 7, pp. 2461–77, 2011.

[24] J. Abramson and G. Anderson, "Thymic epithelial cells," *Annual Review of Immunology*, vol. 35, no. 1, pp. 85–118, 2017.

A Source Code for Initial Protrusion Formation

```
import math
import tissue_forge as tf
import random

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=20)

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = 2.5
    style = {'color': 'blue'}
    dynamics = tf.Overdamped

class ProtrusionType(tf.ParticleTypeSpec):
    """Protrusion particle"""
    radius = 0.5
    style = {'color': 'cyan'}
    dynamics = tf.Overdamped

# Get particle types
main_cell_type = MainCellType.get()
protrusion_type = ProtrusionType.get()

# Create the main cell at the center (25, 25, 25)
main_cell = main_cell_type(position=[25, 25, 25])

# Define stiffness for harmonic potential (spring)
stiffness = 1  # Spring stiffness

all_protrusions = []


# Function to create a single protrusion segment in a random direction
def create_protrusion_segment(main_cell, protrusion_radius=0.5):
    """Create a single protrusion segment from the main cell in a random direction."""

    # Choose a random unit direction in 3D space
    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)

    direction = [
        math.sin(phi) * math.cos(theta),  # x direction
        math.sin(phi) * math.sin(theta),  # y direction
        math.cos(phi)  # z direction
    ]

    # Calculate the new position for the protrusion segment
    main_cell_radius = main_cell.radius
    new_pos = [
        main_cell.position[0] + direction[0] * (main_cell_radius + protrusion_radius),
        main_cell.position[1] + direction[1] * (main_cell_radius + protrusion_radius),
        main_cell.position[2] + direction[2] * (main_cell_radius + protrusion_radius)
    ]

    # Create the protrusion particle at the calculated position
    protrusion = protrusion_type(position=new_pos)
```

```
    protrusion.radius = protrusion_radius

    # Create a harmonic bond (spring) between the main cell and the new protrusion
    rest_length = main_cell_radius + protrusion.radius
    spring_potential = tf.Potential.harmonic(k=stiffness, r0=rest_length)
    tf.Bond.create(spring_potential, main_cell, protrusion)

    # Return the new protrusion
    return protrusion

# Function to shrink the main cell to conserve volume using the volume property
def shrink_main_cell_for_volume_conservation(main_cell, protrusion):
    """
    Shrink the main cell to conserve total volume when a protrusion is added.
    The total volume of the main cell and the protrusion should remain constant.
    """
    # Get the initial volume of the main cell
    initial_main_cell_volume = main_cell.volume
    # Get the volume of the new protrusion
    protrusion_volume = protrusion.volume
    # New main cell volume after adding the protrusion
    new_main_cell_volume = initial_main_cell_volume - protrusion_volume
    # Adjust the main cell volume
    new_radius = ((3/(4*math.pi)) * new_main_cell_volume)**(1/3)
    main_cell.radius = new_radius


# Function to form a protrusion
def form_protrusion(event):
#check if  protrusion<=20, yes..
    """Form a protrusion from the main cell."""
    if len(all_protrusions)<=20:
        protrusion_radius = 0.5  # Radius of the protrusion
        # Create a new protrusion segment in a random direction
        new_protrusion = create_protrusion_segment(main_cell, protrusion_radius)
        all_protrusions.append([new_protrusion])
        # Shrink the main cell to conserve volume
        shrink_main_cell_for_volume_conservation(main_cell, new_protrusion)

        # Print debug information
        print(f"Main cell volume after shrinking: {main_cell.volume})#, New protrusion
        ↪  volume: {new_protrusion.volume}")
    else:
        print('too many protrusions')

# Event to trigger protrusion formation
tf.event.on_time(invoke_method=form_protrusion, period=2.0, start_time=1, end_time=-1)

# Visualization
tf.irun()
```

B Source Code for Protrusion Extension

```
import math
import tissue_forge as tf
import random

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=20)

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = 2.5
    style = {'color': 'blue'}
    dynamics = tf.Overdamped

class ProtrusionType(tf.ParticleTypeSpec):
    """Protrusion particle"""
    radius = 0.5
    style = {'color': 'cyan'}
    dynamics = tf.Overdamped

# Get particle types
main_cell_type = MainCellType.get()
protrusion_type = ProtrusionType.get()

# Create the main cell at the center (25, 25, 25)
main_cell = main_cell_type(position=[25, 25, 25])

# Constants
stiffness = 1       # Spring stiffness
r1 = 0.1            # Radius reduction per segment
min_radius = 0.2    # Minimum allowable radius for segments

# To hold all protrusions with their chain of segments and direction vector
all_protrusions = []

# Function to create the initial protrusion segment in a random direction
def create_initial_protrusion(main_cell, protrusion_radius=0.5):
    """Create the first segment of a protrusion in a random direction from the main
    ↪  cell."""
    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)

    direction = [
        math.sin(phi) * math.cos(theta),
        math.sin(phi) * math.sin(theta),
        math.cos(phi)
    ]

    # Position for the first protrusion particle
    new_pos = [
        main_cell.position[0] + direction[0] * (main_cell.radius + protrusion_radius),
        main_cell.position[1] + direction[1] * (main_cell.radius + protrusion_radius),
        main_cell.position[2] + direction[2] * (main_cell.radius + protrusion_radius)
    ]

    # Create the initial segment
    protrusion = protrusion_type(position=new_pos)
```

```
    protrusion.radius = protrusion_radius

    # Bond between main cell and initial protrusion segment
    rest_length = main_cell.radius + protrusion.radius
    spring_potential = tf.Potential.harmonic(k=stiffness, r0=rest_length)
    tf.Bond.create(spring_potential, main_cell, protrusion)

    return {"chain": [protrusion], "direction": direction}

# Function to add a new segment to an existing protrusion
def grow_protrusion_segment(protrusion_data):
    """Extend the protrusion further along its specified direction, decreasing
    ↪  radius."""
    protrusion_chain = protrusion_data["chain"]
    direction = protrusion_data["direction"]
    last_segment = protrusion_chain[-1]

    # Calculate the new radius with reduction by r1, but don't go below min_radius
    new_radius = max(last_segment.radius - r1, min_radius)

    # Calculate the position for the new segment
    new_pos = [
        last_segment.position[0] + direction[0] * 2 * new_radius,
        last_segment.position[1] + direction[1] * 2 * new_radius,
        last_segment.position[2] + direction[2] * 2 * new_radius
    ]

    # Create the new segment with the reduced radius
    new_segment = protrusion_type(position=new_pos)
    new_segment.radius = new_radius

    # Create bond between last segment and new segment
    rest_length = 2 * new_radius
    spring_potential = tf.Potential.harmonic(k=stiffness, r0=rest_length)
    tf.Bond.create(spring_potential, last_segment, new_segment)

    # Append the new segment to the protrusion chain
    protrusion_chain.append(new_segment)
    return new_segment

# Function to shrink the main cell to conserve total volume
def shrink_main_cell_for_volume_conservation(main_cell, added_volume):
    """Adjust the main cell radius to maintain a constant total volume."""
    initial_volume = main_cell.volume
    new_volume = initial_volume - added_volume

    if new_volume > 0:
        new_radius = ((3/(4*math.pi)) * new_volume)**(1/3)
        main_cell.radius = new_radius

# Track the creation of protrusions separately from their growth
def form_and_grow_protrusion(event):
    """Form one new protrusion or grow existing protrusions up to 3 segments each."""
    # Create one new protrusion if limit is not reached and no new protrusion created
    ↪  in this frame
    if len(all_protrusions) < 20:
        new_protrusion_data = create_initial_protrusion(main_cell)
        all_protrusions.append(new_protrusion_data)
```

```
        # Shrink main cell to conserve volume
        shrink_main_cell_for_volume_conservation(main_cell,
        ↪  new_protrusion_data["chain"][0].volume)

        # Return early to ensure only one new protrusion per frame
        return

    # Grow existing protrusions if they have fewer than 4 segments (1 initial + 3
    ↪  additional)
    for protrusion_data in all_protrusions:
        if len(protrusion_data["chain"]) < 4:  # Max segments per protrusion
            new_segment = grow_protrusion_segment(protrusion_data)
            shrink_main_cell_for_volume_conservation(main_cell, new_segment.volume)
            break  # Only grow one protrusion per time frame to ensure staggered growth

# Schedule protrusion formation and growth at regular intervals
tf.event.on_time(invoke_method=form_and_grow_protrusion, period=2.0, start_time=1,
↪  end_time=-1)

# Visualization
tf.irun()
```

C Source Code for Protrusion Retraction

```
import math
import tissue_forge as tf
import random

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=20)

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = 2.5
    style = {'color': 'blue'}
    dynamics = tf.Overdamped

class ProtrusionType(tf.ParticleTypeSpec):
    """Protrusion particle"""
    radius = 0.5
    style = {'color': 'cyan'}
    dynamics = tf.Overdamped

# Get particle types
main_cell_type = MainCellType.get()
protrusion_type = ProtrusionType.get()

# Create the main cell at the center (25, 25, 25)
main_cell = main_cell_type(position=[25, 25, 25])

R_min = 0.2

# Define stiffness for harmonic potential (spring)
stiffness = 1  # Spring stiffness

# List to store all protrusions and their bonds
all_protrusions = []
all_bonds = []

# Function to create a single protrusion segment in a random direction
def create_protrusion_segment(main_cell, protrusion_radius=0.5):
    """Create a single protrusion segment from the main cell in a random direction."""

    # Choose a random unit direction in 3D space
    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)

    direction = [
        math.sin(phi) * math.cos(theta),  # x direction
        math.sin(phi) * math.sin(theta),  # y direction
        math.cos(phi)  # z direction
    ]

    # Calculate the new position for the protrusion segment
    main_cell_radius = main_cell.radius
    new_pos = [
        main_cell.position[0] + direction[0] * (main_cell_radius + protrusion_radius),
        main_cell.position[1] + direction[1] * (main_cell_radius + protrusion_radius),
        main_cell.position[2] + direction[2] * (main_cell_radius + protrusion_radius)
    ]
```

```
    # Create the protrusion particle at the calculated position
    protrusion = protrusion_type(position=new_pos)
    protrusion.radius = protrusion_radius

    # Create a harmonic bond (spring) between the main cell and the new protrusion
    rest_length = main_cell_radius + protrusion.radius
    spring_potential = tf.Potential.harmonic(k=stiffness, r0=rest_length)
    bond = tf.Bond.create(spring_potential, main_cell, protrusion)

    # Store bond reference to potentially remove it later
    all_bonds.append((bond, protrusion))

    return protrusion

# Function to shrink the main cell to conserve volume when adding a protrusion
def shrink_main_cell_for_volume_conservation(main_cell, protrusion):
    initial_main_cell_volume = main_cell.volume
    protrusion_volume = protrusion.volume
    new_main_cell_volume = initial_main_cell_volume - protrusion_volume
    new_radius = ((3/(4*math.pi)) * new_main_cell_volume)**(1/3)
    main_cell.radius = new_radius

def delete_protrusion_segment(protrusion, segment=-1):
    """Protrusion is a list of segments.
       Each element of this list (a segment) is a tissue forge particle object."""
    protrusion[segment].destroy()
    del(protrusion[segment])

# Function to gradually shrink each protrusion while conserving total volume and remove
↪  bond if radius < 0.5
def shrink_protrusions(event):
    """Gradually shrink each protrusion, adding the lost volume back to the main cell,
    ↪  and remove bond if radius < 0.5."""
    shrink_rate = 0.01  # Rate of shrinking for each protrusion per time step

    for bond, protrusion in all_bonds[:]:  # Iterate over a copy to allow removal
    ↪  during loop
        current_volume = protrusion.volume
        current_radius = protrusion.radius

        if current_radius > shrink_rate:  # Ensure radius does not become negative
            # Reduce the radius of the protrusion
            new_radius = current_radius - shrink_rate
            protrusion.radius = new_radius

            # Calculate new volume and volume lost due to shrinkage
            new_volume = (4/3) * math.pi * new_radius**3
            volume_lost = current_volume - new_volume

            # Add the lost volume back to the main cell
            main_cell_volume = main_cell.volume + volume_lost
            main_cell.radius = ((3/(4*math.pi)) * main_cell_volume)**(1/3)

            # If radius is below 0.5, remove bond to main cell
            if new_radius < R_min:
                delete_protrusion_segment([protrusion])
                #all_bonds.remove((bond, protrusion))
```

```
                print("Bond removed from main cell and protrusion.")

            # Print debug information
            print(f"Main cell volume after conservation: {main_cell.volume}, Protrusion
            ↪  radius: {protrusion.radius}")
        else:
            print("Protrusion fully retracted")
            protrusion.radius = 0  # Set radius to 0 when it can no longer shrink

# Function to form a protrusion
def form_protrusion(event):
    """Form a protrusion from the main cell."""
    protrusion_radius = 0.5  # Radius of the protrusion
    new_protrusion = create_protrusion_segment(main_cell, protrusion_radius)
    all_protrusions.append(new_protrusion)
    shrink_main_cell_for_volume_conservation(main_cell, new_protrusion)

# Events to trigger protrusion formation and shrinking with a timing offset
tf.event.on_time(invoke_method=form_protrusion, period=2.0, start_time=1, end_time=-1)
tf.event.on_time(invoke_method=shrink_protrusions, period=1.0, start_time=3,
↪  end_time=-1)  # Shrink protrusions every 1 second

# Visualization
tf.irun()
```

D Source Code for Protrusion Interaction

```
import math
import tissue_forge as tf
import random

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=5)

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = 2.5
    style = {'color': 'blue'}
    dynamics = tf.Overdamped

class ProtrusionSegmentType(tf.ParticleTypeSpec):
    """Protrusion segment particle"""
    radius = 0.5
    style = {'color': 'cyan'}
    dynamics = tf.Overdamped

main_cell_type = MainCellType.get()
protrusion_segment_type = ProtrusionSegmentType.get()

# Define Morse potential for inter-protrusion repulsion
morse_repulsion = tf.Potential.morse(d=-1, a=1, min=0.1, max=4)
tf.bind.types(morse_repulsion, protrusion_segment_type, protrusion_segment_type)

# Add random force for particle motility
random_force = tf.Force.random(mean=0, std=1)
tf.bind.force(random_force, protrusion_segment_type)

# Define harmonic bond to keep protrusion segments connected
bond_stiffness = 10
segment_distance = 1 * protrusion_segment_type.radius
harmonic_bond = tf.Potential.harmonic(k=bond_stiffness, r0=segment_distance)

# Main cell at the center
main_cell = main_cell_type(position=[25, 25, 25])

# List to store all protrusion segments
protrusion_segments = []  # Global variable to track all protrusion segments

def create_initial_protrusion(main_cell, protrusion_radius=0.5):
    """Create the first segment of a protrusion in a random direction from the main
    ↪  cell."""
    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)

    direction = [
        math.sin(phi) * math.cos(theta),
        math.sin(phi) * math.sin(theta),
        math.cos(phi)
    ]

    # Position for the first protrusion particle
    new_pos = [
        main_cell.position[0] + direction[0] * (main_cell.radius + protrusion_radius),
```

```
            main_cell.position[1] + direction[1] * (main_cell.radius + protrusion_radius),
            main_cell.position[2] + direction[2] * (main_cell.radius + protrusion_radius)
        ]

        # Create the initial segment
        protrusion = protrusion_segment_type(position=new_pos)
        protrusion.radius = protrusion_radius

        # Bond between main cell and initial protrusion segment
        rest_length = main_cell.radius + protrusion.radius
        spring_potential = tf.Potential.harmonic(k=bond_stiffness, r0=rest_length)
        tf.Bond.create(spring_potential, main_cell, protrusion)

        return {"chain": [protrusion], "direction": direction}

    def create_protrusion(protrusion_data, num_segments=3):
        """Create a chain of protrusion segments from an initial segment."""
        global protrusion_segments
        chain = protrusion_data["chain"]
        direction = protrusion_data["direction"]

        origin = chain[-1].position
        last_segment = chain[-1]

        # Create the rest of the chain
        for _ in range(num_segments - 1):
            # Calculate position of the next segment
            new_pos = [
                origin[0] + direction[0] * segment_distance,
                origin[1] + direction[1] * segment_distance,
                origin[2] + direction[2] * segment_distance
            ]

            # Create new segment
            segment = protrusion_segment_type(position=new_pos)

            # Add harmonic bond to the previous segment
            tf.Bond.create(harmonic_bond, last_segment, segment)

            # Update the chain
            chain.append(segment)
            last_segment = segment
            origin = segment.position

        protrusion_segments.extend(chain)  # Add the full chain to global list

    # Simulation step
    def simulation_step(event):
        """Simulation step to create new protrusions and enforce interactions."""
        global protrusion_segments
        max_segments = 30  # Maximum number of segments

        if len(protrusion_segments) < max_segments:
            # Create an initial protrusion segment attached to the main cell
            protrusion_data = create_initial_protrusion(main_cell)
            # Extend the protrusion into a chain
            create_protrusion(protrusion_data, num_segments=random.randint(2, 5))
```

```
    # Stop event once maximum segments are reached
    if len(protrusion_segments) >= max_segments:
        print(f"Maximum segments reached. Total segments: {len(protrusion_segments)}")
        tf.event.remove(event)

# Register the simulation step event to run periodically
tf.event.on_time(invoke_method=simulation_step, period=1.0, start_time=1.0,
↪  end_time=50.0)

# Visualization
tf.irun()
```

E Source Code for Active Protrusion Movement

```
import math
import tissue_forge as tf
import random

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=20)

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = 2.5
    style = {'color': 'blue'}
    dynamics = tf.Overdamped

class ProtrusionType(tf.ParticleTypeSpec):
    """Protrusion particle"""
    radius = 0.5
    style = {'color': 'cyan'}
    dynamics = tf.Overdamped

# Get particle types
main_cell_type = MainCellType.get()
protrusion_type = ProtrusionType.get()

# Create the main cell at the center (25, 25, 25)
main_cell = main_cell_type(position=[25, 25, 25])

# Constants
movement_force_magnitude = 1.5  # Magnitude of random forces applied to tips
max_segments = 20   # Number of initial protrusion segments
segment_particles = 3  # Number of particles per segment
radius_decay = 0.2  # Amount by which each successive particle radius decreases
all_segments = []  # Stores each segment's particles

# Initialize multiple protrusion segments with decreasing radius
for _ in range(max_segments):
    segment = []

    # Random direction for the segment
    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)
    direction = [
        math.sin(phi) * math.cos(theta),
        math.sin(phi) * math.sin(theta),
        math.cos(phi)
    ]

    previous_particle = main_cell
    for i in range(segment_particles):
        # Position for each particle in the segment along the direction vector
        distance_from_main = main_cell.radius + (i + 1) * 0.5
        pos = [
            main_cell.position[0] + direction[0] * distance_from_main,
            main_cell.position[1] + direction[1] * distance_from_main,
            main_cell.position[2] + direction[2] * distance_from_main
        ]
```

```
        # Create the protrusion particle with decreasing radius
        initial_radius = max(0.5, 0.5 - i * radius_decay)
        protrusion = protrusion_type(position=pos)
        protrusion.radius = initial_radius
        segment.append(protrusion)

        # Create a bond between consecutive particles in the segment
        rest_length = previous_particle.radius + protrusion.radius
        spring_potential = tf.Potential.harmonic(k=1, r0=rest_length)
        tf.Bond.create(spring_potential, previous_particle, protrusion)

        previous_particle = protrusion

    # Add the segment to the list of all segments
    all_segments.append(segment)

# State variable to track if tips are moving away or towards the main cell
moving_away = True

# Function to apply movement force to a random segment's tip alternating between away
↪  and towards main cell
def apply_alternating_movement_force(event):
    """Applies a movement force to the tip of a randomly selected segment, alternating
    ↪  between away from and towards the main cell."""
    global moving_away

    # Randomly select one segment for movement
    selected_segment = random.choice(all_segments)

    # Choose the tip particle of the selected segment
    tip_particle = selected_segment[-1]

    # Calculate the direction from main cell to the tip
    direction_from_main_to_tip = [
        tip_particle.position[0] - main_cell.position[0],
        tip_particle.position[1] - main_cell.position[1],
        tip_particle.position[2] - main_cell.position[2]
    ]

    # Normalize the direction vector
    magnitude = math.sqrt(sum(x ** 2 for x in direction_from_main_to_tip))
    if magnitude > 0:  # Avoid division by zero
        # Determine the force direction based on moving_away flag
        if moving_away:
            normalized_force = [f * movement_force_magnitude / magnitude for f in
            ↪  direction_from_main_to_tip]
        else:
            normalized_force = [-f * movement_force_magnitude / magnitude for f in
            ↪  direction_from_main_to_tip]

        # Apply the force to the tip particle
        tip_particle.force_init = normalized_force

        # Debug output to track applied forces
        movement_direction = "away from" if moving_away else "towards"
        print(f"Applying force {normalized_force} to segment tip at
        ↪  {tip_particle.position}, moving {movement_direction} main cell")
```

```
    # Toggle moving_away for next time
    moving_away = not moving_away

# Schedule the alternating movement event
tf.event.on_time(invoke_method=apply_alternating_movement_force, period=0.1,
↪  start_time=0.25, end_time=-1)

# Visualization
tf.irun()
```

F Source Code for Main Cell Splitting (Division)

```
import math
import tissue_forge as tf
import random

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=20)

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = 2.5
    style = {'color': 'blue'}
    dynamics = tf.Overdamped

class ProtrusionType(tf.ParticleTypeSpec):
    """Protrusion particle"""
    radius = 0.5
    style = {'color': 'cyan'}
    dynamics = tf.Overdamped

# Get particle types
main_cell_type = MainCellType.get()
protrusion_type = ProtrusionType.get()

# Create the main cell at the center (25, 25, 25)
first_main_cell = main_cell_type(position=[25, 25, 25])

# Define stiffness for harmonic potential (spring)
stiffness = 1  # Spring stiffness

main_cell_list  = [first_main_cell]
all_protrusions = [ [] ]

# Function to retract all protrusions
def retract_protrusions(main_cell, main_cell_id):
    """Remove all protrusions and reset the main cell radius."""
    this_mc_protrusions = all_protrusions[main_cell_id]
    for protrusion in this_mc_protrusions:
        protrusion[0].destroy()  # Delete protrusion particle
    this_mc_protrusions.clear()  # Clear the list
    main_cell.radius = 2.5  # Reset radius to original
    print("All protrusions retracted for cell", main_cell_id)

# Function to create a single protrusion segment in a random direction
def create_protrusion_segment(cell, protrusion_radius=0.5):
    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)
    direction = [
        math.sin(phi) * math.cos(theta),  # x direction
        math.sin(phi) * math.sin(theta),  # y direction
        math.cos(phi)  # z direction
    ]

    cell_radius = cell.radius
    new_pos = [
        cell.position[0] + direction[0] * (cell_radius + protrusion_radius),
```

```
        cell.position[1] + direction[1] * (cell_radius + protrusion_radius),
        cell.position[2] + direction[2] * (cell_radius + protrusion_radius)
    ]

    protrusion = protrusion_type(position=new_pos)
    protrusion.radius = protrusion_radius
    rest_length = cell_radius + protrusion.radius
    spring_potential = tf.Potential.harmonic(k=stiffness, r0=rest_length)
    tf.Bond.create(spring_potential, cell, protrusion)
    return protrusion

# Function to shrink the main cell for volume conservation
def shrink_main_cell_for_volume_conservation(main_cell, protrusion):
    initial_main_cell_volume = main_cell.volume
    protrusion_volume = protrusion.volume
    new_main_cell_volume = initial_main_cell_volume - protrusion_volume
    new_radius = ((3/(4*math.pi)) * new_main_cell_volume)**(1/3)
    main_cell.radius = new_radius

# Function to form a protrusion
def form_protrusion_event(event):
    for mc_index, mainCell in enumerate(main_cell_list):
        if len(all_protrusions[mc_index]) <= 20:
            protrusion_radius = 0.5
            new_protrusion = create_protrusion_segment(mainCell, protrusion_radius)
            all_protrusions[mc_index].append([new_protrusion])
            shrink_main_cell_for_volume_conservation(mainCell, new_protrusion)
            print("protrusions in cell", mc_index, ":", len(all_protrusions[mc_index]))
        else:
            print('too many protrusions')


def create_random_position(particle, distance):

    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)
    direction = [
        math.sin(phi) * math.cos(theta),  # x direction
        math.sin(phi) * math.sin(theta),  # y direction
        math.cos(phi)  # z direction
    ]

    new_pos = [
        particle.position[0] + direction[0] * distance,
        particle.position[1] + direction[1] * distance,
        particle.position[2] + direction[2] * distance
    ]

    return(new_pos)


# Function to split the main cell into two
def split_main_cell_event(event):

    temporary_list = []
    mark_for_deletion = []

    for mc_index, mainCell in enumerate(main_cell_list):
```

```
        print("retract protrusions of cell", mc_index)
        retract_protrusions(mainCell, mc_index)  # Retract protrusions before splitting

        # Create two new main cells at positions slightly apart
        distance = 3
        pos1 = create_random_position(mainCell, distance)
        pos2 = create_random_position(mainCell, distance)

        main_cell1 = main_cell_type(position=pos1)
        main_cell2 = main_cell_type(position=pos2)

        mother_volume = mainCell.volume
        new_cell_volume = mother_volume / 2
        new_radius = ((3 / (4 * math.pi)) * new_cell_volume) ** (1 / 3)

        main_cell1.radius = new_radius
        main_cell2.radius = new_radius

        mainCell.destroy()  # Delete the original tissue forge particle object main
        ↪  cell
        mark_for_deletion.append(mc_index) # keep track of which cells will be deleted

        temporary_list.append(main_cell1)
        temporary_list.append(main_cell2)
        print("Main cell", mc_index ,"split into two.\n")

    # clean up: main cell list, protrusion list
    # index needs to be reversed, because each deletion changes the order of list items
    for item in mark_for_deletion[::-1]:
        del main_cell_list[item]
        del all_protrusions[item]

    # create new lists for the newly added cells
    for item in temporary_list:
        main_cell_list.append(item) # add the new cell to the main cell list
        all_protrusions.append([])  # add an empty list for the new cell's protrusions

    print("number of cells", len(main_cell_list), "\nprotrusion lists"
    ↪  ,len(all_protrusions))



# Function to form protrusions for a specific cell (for split cells)
def form_protrusion_cell(cell, cell_index):
    if len(all_protrusions[cell_index]) <= 20:
        protrusion_radius = 0.5
        new_protrusion = create_protrusion_segment(cell, protrusion_radius)
        all_protrusions[cell_index].append([new_protrusion])
        shrink_main_cell_for_volume_conservation(cell, new_protrusion)
        print("protrusions in cell", cell_index, ":", len(all_protrusions[cell_index]))
    else:
        print('too many protrusions')

def split_main_cell(mc_index, mainCell, old_volume):

    temporary_list = []
    mark_for_deletion = []
```

```
    print("retract protrusions of cell", mc_index)
    retract_protrusions(mainCell, mc_index)  # Retract protrusions before splitting

    # Create two new main cells at positions slightly apart
    distance = 3
    pos1 = create_random_position(mainCell, distance)
    pos2 = create_random_position(mainCell, distance)

    main_cell1 = main_cell_type(position=pos1)
    main_cell2 = main_cell_type(position=pos2)

    new_cell_volume = old_volume / 2
    new_radius = ((3 / (4 * math.pi)) * new_cell_volume) ** (1 / 3)

    main_cell1.radius = new_radius
    main_cell2.radius = new_radius

    mainCell.destroy()  # Delete the original tissue forge particle object main cell
    mark_for_deletion.append(mc_index) # keep track of which cells will be deleted

    temporary_list.append(main_cell1)
    temporary_list.append(main_cell2)
    print("Main cell", mc_index ,"split into two.\n")

    # clean up: main cell list, protrusion list
    # index needs to be reversed, because each deletion changes the order of list items
    for item in mark_for_deletion[::-1]:
        del main_cell_list[item]
        del all_protrusions[item]

    # create new lists for the newly added cells
    for item in temporary_list:
        main_cell_list.append(item) # add the new cell to the main cell list
        all_protrusions.append([])  # add an empty list for the new cell's protrusions

    print("number of cells", len(main_cell_list), "\nprotrusion lists"
    ↪  ,len(all_protrusions))




p_division_per_time_step = 0.1

def simulate_cell_behavior(event):
    print("\ntime", tf.Universe.time)

    for mc_index, mainCell in enumerate(main_cell_list):

        print(mainCell.radius)

        runif = random.uniform(0, 1)
        if runif <= p_division_per_time_step:
            print("cell will divide")
            split_main_cell(mc_index, mainCell, mainCell.volume)

        else:
            print("cell grows protrusions")
```

```
            form_protrusion_cell(mainCell, mc_index)



# Initial protrusion formation event for the original main cell
tf.event.on_time(invoke_method=form_protrusion_event, period=1.0, start_time=1,
↪  end_time=5)

# Event to trigger main cell splitting after retraction phase
tf.event.on_time(invoke_method=simulate_cell_behavior, period=1.0, start_time=6,
↪  end_time=50)

# Visualization
tf.irun()
```

G Source Code for The Whole Model

```
import json
import os
import math
import tissue_forge as tf
import random
from pathlib import Path
from scipy.spatial import ConvexHull
import numpy as np


# SIMULATION PARAMETERS
parameter_number_of_protrusions = 3 # 3 6 9
parameter_length_of_protrusions = 4 # 2 4 6

parameter_morse_repulsion = -0.1 # -0.075 -0.1 -0.25

parameter_cell_split_time = 14
parameter_cell_split_time_randomness = 15 #r= 0 5

parameter_cell_growth_rate = 0.01
parameter_prot_force = 0.1

par_stiffness = 0.5            # Spring par_stiffness
par_min_radius = 0.2           # Minimum allowable radius for segments
par_max_radius = 0.5           # Maximum allowable radius for segments
par_min_radius_mc = 0.75       # Minimum allowable radius for main cell
par_radius_reduct = 0.1        # Radius reduction per segment
par_radius_maincell = 2.5      # Radius of the initial main cell
par_rest_length_scaling = 0.85 # Scaling for spring rest length

# Initialize 3D simulation
tf.init(dim=[50, 50, 50], cutoff=20)

tf.system.decorate_scene(False) # remove grid
tf.system.set_background_color(tf.FVector3(1, 1, 1)) # set background color to white
tf.system.set_shininess(0.5 * tf.system.get_shininess())

# Define particle types for the main cell and protrusions
class MainCellType(tf.ParticleTypeSpec):
    """Main cell particle"""
    radius = par_radius_maincell
    style = {'color': 'lightseagreen'}
    dynamics = tf.Overdamped

class ProtrusionType(tf.ParticleTypeSpec):
    """Protrusion particle"""
    radius = par_max_radius
    style = {'color': 'mediumturquoise'}
    dynamics = tf.Overdamped

# Get particle types
main_cell_type = MainCellType.get()
protrusion_type = ProtrusionType.get()

# Create the main cell at the center (25, 25, 25)
main_cell = main_cell_type(position=[25, 25, 25])
```

```
# Define Morse potential for repulsion
# Scale down the width of the potential (parameter a) for smaller particle interactions
morse_repulsion_main_main = tf.Potential.morse(d=parameter_morse_repulsion, a=2,
↪  min=0.1, max=4)
morse_repulsion_main_prot = tf.Potential.morse(d=parameter_morse_repulsion, a=0.5,
↪  min=0.1, max=1)
morse_repulsion_prot_prot = tf.Potential.morse(d=parameter_morse_repulsion, a=0.1,
↪  min=0.1, max=1)

tf.bind.types(morse_repulsion_main_main, main_cell_type, main_cell_type)
tf.bind.types(morse_repulsion_main_prot, main_cell_type, protrusion_type)
tf.bind.types(morse_repulsion_prot_prot, protrusion_type, protrusion_type)

# List to hold all main cells and protrusions
main_cell_list = [main_cell]
all_protrusions = [[]]  # Protrusions for the first main cell

# For tracking cell states
cell_states = ["growing"]
cell_split_timer = [parameter_cell_split_time]


# Check if radius of particle 1 will fall under threshold if particle 2 added (removed
↪  if False)
def can_add_particle(particle1, particle2_volume, threshold):
    p1_volume = particle1.volume
    new_p1_volume = p1_volume - particle2_volume
    new_radius = ((3/(4*math.pi)) * new_p1_volume)**(1/3)

    if new_radius <= threshold:
        return(False)
    else:
        return(True)

# Function to shrink or grow the main cell to conserve volume using the volume property
def volume_conservation(main_cell, protrusion, retract = False):
    """
    Shrink or grow the main cell to conserve total volume when a protrusion is added or
    ↪  removed.
    The total volume of the main cell and the protrusion should remain constant.
    """
    # Get the initial volume of the main cell
    initial_main_cell_volume = main_cell.volume

    # Get the volume of the protrusion
    protrusion_volume = protrusion.volume

    if retract:
        # New main cell volume after adding the protrusion
        new_main_cell_volume = initial_main_cell_volume + protrusion_volume

    else:
        # New main cell volume after adding the protrusion
        new_main_cell_volume = initial_main_cell_volume - protrusion_volume

    # Adjust the main cell volume
    new_radius = ((3/(4*math.pi)) * new_main_cell_volume)**(1/3)
```

```
    main_cell.radius = new_radius


# Function to create the initial protrusion segment in a random direction
def create_initial_protrusion(main_cell, main_cell_index):
    """Create the first segment of a protrusion in a random direction from the main
    ↪  cell."""

    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)
    direction = [
        math.sin(phi) * math.cos(theta),
        math.sin(phi) * math.sin(theta),
        math.cos(phi)
    ]

    # Try adding protrusion with default radius. If not possible make it half the main
    ↪  cell, but not less than par_min_radius.
    protrusion_radius = max(min(par_max_radius, main_cell.radius * 0.5),
    ↪  par_min_radius)
    protrusion_volume = (4/3)*math.pi*(protrusion_radius**3)

    # Check if the main cell can support adding a new protrusion
    if can_add_particle(main_cell, protrusion_volume, par_min_radius_mc):
        new_pos = [
            main_cell.position[0] + direction[0] * (main_cell.radius +
            ↪  protrusion_radius),
            main_cell.position[1] + direction[1] * (main_cell.radius +
            ↪  protrusion_radius),
            main_cell.position[2] + direction[2] * (main_cell.radius +
            ↪  protrusion_radius)
        ]

        protrusion = protrusion_type(position=new_pos)

        # Calculate the new radius with reduction by par_radius_reduct, but don't go
        ↪  below par_min_radius
        protrusion.radius = protrusion_radius

        rest_length = par_rest_length_scaling * (main_cell.radius + protrusion.radius)
        spring_potential = tf.Potential.harmonic(k=par_stiffness, r0=rest_length)
        tf.Bond.create(spring_potential, main_cell, protrusion)

        # Add to the list of protrusions
        protrusion_data = {"chain": [protrusion], "direction": direction}
        all_protrusions[main_cell_index].append(protrusion_data)

        # Grow/shrink the main cell to conserve volume
        volume_conservation(main_cell, protrusion)


# Function to add a new segment to an existing protrusion
def grow_protrusion_segment(main_cell, protrusion_data):
    """Extend the protrusion further along its specified direction, decreasing
    ↪  radius."""

    protrusion_chain = protrusion_data["chain"]
    direction = protrusion_data["direction"]
```

```
    last_segment = protrusion_chain[-1]

    # Calculate the new radius with reduction by par_radius_reduct, but don't go below
    ↪  par_min_radius
    new_radius = max(last_segment.radius - par_radius_reduct, par_min_radius)

    protrusion_volume = (4/3)*math.pi*(new_radius**3)

    # Check if the main cell can support adding a new protrusion
    if can_add_particle(main_cell, protrusion_volume, par_min_radius_mc):
        # Calculate the position for the new segment
        scaling = par_rest_length_scaling * 0.7
        new_pos = [
            last_segment.position[0] + direction[0] * scaling * new_radius,
            last_segment.position[1] + direction[1] * scaling * new_radius,
            last_segment.position[2] + direction[2] * scaling * new_radius
        ]

        # Create the new segment with the reduced radius
        new_segment = protrusion_type(position=new_pos)
        new_segment.radius = new_radius

        # Create bond between last segment and new segment
        rest_length = par_rest_length_scaling * (new_radius + last_segment.radius)
        spring_potential = tf.Potential.harmonic(k=par_stiffness, r0=rest_length)
        tf.Bond.create(spring_potential, last_segment, new_segment)

        # Append the new segment to the protrusion chain
        protrusion_chain.append(new_segment)

        # Grow/shrink the main cell to conserve volume
        volume_conservation(main_cell, new_segment)

# Function to extend all protrusions of a cell before splitting
def extend_all_protrusions(main_cell_index):
    """Extend all protrusions for a given main cell before splitting."""
    main_cell = main_cell_list[main_cell_index]
    protrusions = all_protrusions[main_cell_index]
    i = 0
    for protrusion_data in protrusions:
        if len(protrusion_data["chain"]) < parameter_length_of_protrusions:
            grow_protrusion_segment(main_cell, protrusion_data)
        i += 1

# Function to split the main cell into two
def split_main_cell(main_cell_index):
    """Split a main cell into two new cells after retracting its protrusions."""
    extend_all_protrusions(main_cell_index)

    main_cell = main_cell_list[main_cell_index]
    distance = main_cell.radius  # Distance between new cells

    theta = random.uniform(0, 2 * math.pi)
    phi = random.uniform(0, math.pi)
    direction = [
        math.sin(phi) * math.cos(theta),
        math.sin(phi) * math.sin(theta),
        math.cos(phi)
```

```
    ]
    pos1 = [
        main_cell.position[0] + direction[0] * (distance/2),
        main_cell.position[1] + direction[1] * (distance/2),
        main_cell.position[2] + direction[2] * (distance/2)
    ]
    pos2 = [
        main_cell.position[0] - direction[0] * (distance/2),
        main_cell.position[1] - direction[1] * (distance/2),
        main_cell.position[2] - direction[2] * (distance/2)
    ]

    # Create two new cells
    main_cell1 = main_cell_type(position=pos1)
    main_cell2 = main_cell_type(position=pos2)

    # Set volumes of new cells
    mother_volume = main_cell.volume
    new_cell_volume = mother_volume / 2
    new_radius = ((3 / (4 * math.pi)) * new_cell_volume) ** (1 / 3)
    main_cell1.radius = new_radius
    main_cell2.radius = new_radius

    # Destroy old cell
    main_cell.destroy()

    # Add new cells to the list
    main_cell_list[main_cell_index] = main_cell1
    main_cell_list.append(main_cell2)

    # Add entries for new protrusions
    all_protrusions[main_cell_index] = []
    all_protrusions.append([])

    cell_states[main_cell_index] = "growing"
    cell_states.append("growing")

    # Set new cell split timer
    random_value_1 = random.uniform(-1*parameter_cell_split_time_randomness,
    ↪  parameter_cell_split_time_randomness)
    random_value_2 = random.uniform(-1*parameter_cell_split_time_randomness,
    ↪  parameter_cell_split_time_randomness)
    cell_split_timer[main_cell_index] = parameter_cell_split_time + random_value_1
    cell_split_timer.append(parameter_cell_split_time + random_value_2)


# Function to retract all protrusions of a cell
def retract_one_protrusion_segment(main_cell_index):
    """Retract one segment from the protrusions of a specific cell."""
    main_cell = main_cell_list[main_cell_index]
    protrusions = all_protrusions[main_cell_index]
    for protrusion_data in protrusions:
        if protrusion_data["chain"]:  # If there are still segments in the chain
            last_segment = protrusion_data["chain"].pop()  # Remove the last segment
            volume_conservation(main_cell, last_segment, retract = True)
            last_segment.destroy()
            # Stop after removing one segment
            #return
```

```
    # If all protrusions are empty, clear the list
    all_protrusions[main_cell_index] = [
        p for p in protrusions if p["chain"]  # Keep only non-empty protrusions
    ]

# Function to simulate cell behavior
def simulate_cell_behavior(event):
    """Main simulation: extension, retraction, and division of cells."""

    for mc_index, mc in enumerate(main_cell_list):

        if cell_states[mc_index] == "growing" and cell_split_timer[mc_index] < 1: #
        ↪  cell decides to divide
            if mc.radius >= par_min_radius_mc: # Divide only if large enough
                cell_states[mc_index] = "retracting" # switch to retracting
        else:
            cell_split_timer[mc_index] -= 1

        if cell_states[mc_index] == "growing":
            # Grow the main cell
            mc.radius += parameter_cell_growth_rate

            # Add new protrusions
            if len(all_protrusions[mc_index]) < parameter_number_of_protrusions:
                create_initial_protrusion(mc, mc_index)

            # Extend protrusions
            extend_all_protrusions(mc_index)

            # Move protrusion tips in random direction
            for protrusion in all_protrusions[mc_index]:

                direction = protrusion["direction"]
                random_force = random.uniform(-parameter_prot_force,
                ↪  parameter_prot_force)

                # Apply random force to all segments
                for segment in protrusion["chain"]:
                    segment.force_init = [random_force*d for d in direction]

        elif cell_states[mc_index] == "retracting":
            # Set segment force to 0
            for protrusion in all_protrusions[mc_index]:
                for segment in protrusion["chain"]:
                    segment.force_init = [0, 0, 0]

            retract_one_protrusion_segment(mc_index)
            if not all_protrusions[mc_index]:
                cell_states[mc_index] = "ready_to_split"

        elif cell_states[mc_index] == "ready_to_split":
            split_main_cell(mc_index)


def calculate_global_convex_hull_with_radius(main_cell_list, all_protrusions):
    """
    Calculate the global convex hull volume by incorporating the radius of each
    ↪  particle.
```

```
    """

    def add_sphere_points(center, radius, num_points=12):
        """Add points to represent the surface of a sphere using cosine and sine
        ↪  functions in 3D."""
        points = []
        phi_angles = np.linspace(0, np.pi, num_points // 2, endpoint=False)
        theta_angles = np.linspace(0, 2 * np.pi, num_points, endpoint=False)

        for phi in phi_angles:
            for theta in theta_angles:
                x = center[0] + radius * np.sin(phi) * np.cos(theta)
                y = center[1] + radius * np.sin(phi) * np.sin(theta)
                z = center[2] + radius * np.cos(phi)
                points.append([x, y, z])

        return points

    def collect_points():
        points = []

        # Collect points from main cells
        for main_cell in main_cell_list:
            center = np.array(main_cell.position)
            radius = main_cell.radius
            points.extend(add_sphere_points(center, radius))

        # Collect points from protrusion segments
        for protrusions in all_protrusions:
            for protrusion_data in protrusions:
                for segment in protrusion_data["chain"]:
                    center = np.array(segment.position)
                    radius = segment.radius
                    points.extend(add_sphere_points(center, radius))

        return np.array(points) if points else np.empty((0, 3))  # Ensure non-empty
        ↪  array

    # Collect all points
    points = collect_points()

    # Ensure enough points for convex hull
    if points.shape[0] >= 4:
        try:
            hull = ConvexHull(points)
            return hull.volume
        except Exception as e:
            print(f"Convex hull computation failed: {e}")
            return 0
    else:
        return 0  # Not enough points to fo

def calculate_total_particle_volume(main_cell, protrusions):
    total_volume = (4 / 3) * math.pi * (main_cell.radius ** 3)
    for protrusion_data in protrusions:
        for segment in protrusion_data["chain"]:
            total_volume += (4 / 3) * math.pi * (segment.radius ** 3)
    return total_volume
```

```
def calculate_empty_volume(main_cell, protrusions):
    convex_hull_volume = calculate_convex_hull_volume(main_cell, protrusions)
    total_particle_volume = calculate_total_particle_volume(main_cell, protrusions)
    return max(0, convex_hull_volume - total_particle_volume)

# Volume conservation and protrusion functions
def can_add_particle(particle1, particle2_volume, threshold):
    p1_volume = particle1.volume
    new_p1_volume = p1_volume - particle2_volume
    new_radius = ((3 / (4 * math.pi)) * new_p1_volume) ** (1 / 3)
    return new_radius > threshold

def volume_conservation(main_cell, protrusion, retract=False):
    initial_main_cell_volume = main_cell.volume
    protrusion_volume = protrusion.volume
    new_main_cell_volume = (initial_main_cell_volume + protrusion_volume) if retract
    ↪  else (initial_main_cell_volume - protrusion_volume)
    new_radius = ((3 / (4 * math.pi)) * new_main_cell_volume) ** (1 / 3)
    main_cell.radius = new_radius

# Add simulation behavior and export functions
# Export simulation state with total particle volume included
def export_simulation_state(event):
    global snapshot_count
    output_dir = Path("C:/Users/Bidayatul Masulah/Downloads/Internship
    ↪  Utrecht/simulation/92")
    output_dir.mkdir(parents=True, exist_ok=True)

    # Export tissue forge simulation state
    output_path = output_dir / f"all_simulation_{snapshot_count}.json"
    tf.io.toFile(str(output_path.resolve()))

    # Calculate global convex hull volume (with surface)
    global_convex_hull_volume =
    ↪  calculate_global_convex_hull_with_radius(main_cell_list, all_protrusions)

    # Calculate total volume of all particles
    total_particle_volume_global = sum(
        calculate_total_particle_volume(main_cell_list[i], all_protrusions[i])
        for i in range(len(main_cell_list))
    )

    # Calculate global empty volume
    global_empty_volume = max(0, global_convex_hull_volume -
    ↪  total_particle_volume_global)

    # Export metrics
    metrics = {
        "snapshot": snapshot_count,
        "global_convex_hull_volume": global_convex_hull_volume,
        "total_particle_volume": total_particle_volume_global,
        "global_empty_volume": global_empty_volume
    }

    metrics_path = output_dir / f"metrics_{snapshot_count}.json"
    with open(metrics_path, "w") as f:
        json.dump(metrics, f, indent=4)
```

```
    screenshot_path = output_dir / f"simulation_{snapshot_count:03}.jpg"
    tf.system.screenshot(str(screenshot_path.resolve()))

    snapshot_count += 1


snapshot_count = 0

# Event scheduling
tf.event.on_time(
    invoke_method=simulate_cell_behavior,
    period=5,
    start_time=1,
    end_time=500
)
tf.event.on_time(
    invoke_method=export_simulation_state,
    period=5,
    start_time=0.0,
    end_time=500
)

# Run simulation
tf.irun()
```