Determination of vertices in cell models

Tim Vogel

January 8, 2025

Abstract

abstract

1 Introduction

The arrangement of vertices within cellular structures plays a critical role in determining their mechanical and functional properties. Mathematical modeling of these vertices enables researchers to better understand phenomena ranging from tissue morphogenesis to artificial material design. Despite significant progress, challenges remain in accurately modeling vertex positions, particularly in irregular or dynamic cell configurations. Here, we aim to address these challenges by developing a robust framework for vertex determination that integrates theoretical principles and computational methods.

2 Description of the problem

We are provided with real-life data of biological cells, including sample images obtained from a suitable experimental setup. Phase-field models of these cells have already been derived, representing a foundational abstraction of their structure. The next step is to further simplify the cell model by computing polygonal representations, where vertices are defined as two-dimensional points and edges connect these points, forming polygonal structures. For individual cells, computing discrete polygonal shapes is straightforward. However, complications arise when considering the system of all cells as a whole. We aim to achieve a seamless, gapless configuration in which cells fit together perfectly without air pockets. This requires shared vertices between

adjacent cells, with some vertices being common to more than two cells. The primary focus of this work is to determine the positions of these shared vertices. Although existing approaches, such as Voronoi models, address related problems, they do not fully align with the specific requirements of our system. We will build upon this existing knowledge and develop an effective solution, including an efficient implementation for computational purposes.

3 Current code

The provided Python script implements a computational workflow for processing and analyzing biological cell models represented as 2D polygonal meshes. The script is tailored to simulate, extract, and analyze cell boundaries, focusing on shared vertices between neighboring cells. Below, the script's key components and workflow are described.

3.1 Purpose and Context

The script processes data related to biological cells, using the Visualization Toolkit (VTK) and other numerical libraries. It aims to:

- Extract vertices and contours from cell data.
- Interpolate and resample scalar fields (e.g., phase data) on a fine computational grid.
- Compute distances and identify shared vertices between neighboring cells, accounting for periodic boundary conditions.

The workflow is customized for a specific research project and contains hardcoded parameters for integration into an existing computational setup.

3.2 Key Components and Functions

- **Setup and Imports**: The script imports various libraries, such as `vtk`, `numpy`, and `pandas`, for data manipulation and file handling. Paths to dependencies and output directories are predefined.
- **Contour Extraction** (`calculateInnerContour`): This function extracts and organizes contour points from a scalar field within a VTK file, allowing for sorted representation of cell boundaries.

- **Cell Midpoints** (`all_my_midpoints`): Midpoints for individual cells are calculated from positional data in CSV files for a fixed time frame.
- **Scalar Field Processing**:
  - `interpolate_phi_on_fine_grid`: Interpolates scalar field data onto a fine computational grid using a Gaussian kernel.
  - `resample_phi_on_fine_grid`: Resamples scalar fields onto a fine grid using VTK's resampling tools.
- **Distance Computation**:
  - `calculate_unsigned_dist`: Computes unsigned distance fields for a given scalar threshold using Fast Marching Methods.
  - `all_my_distances`: Computes distances between cells, appends them to the grid, and stores the data in output files.
- **Vertex Identification** (`all_my_vertices`): This function identifies shared vertices between cells by comparing unsigned distance functions. The identified vertices are adjusted for periodic boundary conditions and stored for further analysis.
- **Utility Functions**:
  - `adjust_point`: Adjusts point coordinates to ensure periodicity.
  - `read_fine_grid`: Extracts grid coordinates from a VTK file.
  - `recalculate_indices`: Maps fine grid coordinates to integer indices for efficient computation.

3.3 Workflow

1. **Input Data**: The script processes:

4 Improvements

4.1 old times

$TIMESAREALLFORNoCells = 5$
$all_my_idpointsexecutedin0.4201seconds$

$read_{f}ine_{g}ridexecutedin11.3103seconds$
$recalculate_{i}ndicesexecutedin3.0082seconds$
$resample_{p}hi_{o}n_{f}ine_{g}ridexecutedin7.3366seconds$
$calculate_{u}nsigned_{d}istexecutedin0.6039seconds$
$resample_{p}hi_{o}n_{f}ine_{g}ridexecutedin7.2603seconds$
$all_{m}y_{d}istancesneedstime : NoCells*(resemple_{p}hi+calculate_{u}nsigned_{d}ist)+$
$all_{m}y_{v}ertices$
$all_{m}y_{v}erticesexecutedin2.5098seconds(WITHNoCellssetto5)$
$all_{m}y_{d}istancesexecutedin72.8017seconds(WITHNoCellssetto5)$
$adjust_{p}ointexecutedin0.0000seconds$
$clean_{a}nd_{c}ollect_{m}y_{v}erticesneedstime : N_{C}ell * functionality$
$clean_{a}nd_{c}ollect_{m}y_{v}erticesexecutedin0.0140seconds$

5 Results

s

6 Outlook