Visualizing Cells and their Connectivity Graphs for CompuCell3D

Randy Heiland*
CREST, Pervasive Technology Institute
Indiana University

Maciek Swat†
Biocomplexity Institute
Indiana University

James Sluka‡
Biocomplexity Institute
Indiana University

Benjamin Zaitlen§
Biocomplexity Institute
Indiana University

Abbas Shirinifard¶
Biocomplexity Institute
Indiana University

Gilberto L. Thomas‖
Instituto de Fisica
Universidade Federal do Rio Grande do Sul

Andrew Lumsdaine**
CREST, Pervasive Technology Institute
Indiana University

James A. Glazier††
Department of Physics,
Biocomplexity Institute
Indiana University

ABSTRACT

Developing models that simulate the behavior of different types of interacting biological cells can be a very time consuming and error prone task. CompuCell3D is an open source application that addresses this challenge. It provides interactive and customizable visualizations that help a user detect when a model is producing the desired behavior and when it is failing. It also allows for high quality image generation for publications and presentations. CompuCell3D uses the Python programming language which allows for easy extensions. Examples are provided for performing graph analyses of cell connectivity.

Index Terms: J.3 [Computer Applications]: Life and Medical Sciences—Biology and Genetics; D.3.2 [Programming Languages]: Language Classifications—Very high-level languages

1 INTRODUCTION

Developmental biology is a fascinating subject and provides significant challenges for computational science. The fact that each of us began as a single cell and developed into trillions of cells, producing a breathing, moving, thinking organism should be a source of wonder. Many of the underlying processes for this development are not unique to humans, however; life on this planet is, fortunately, very diverse and many developmental processes are common.

Since the discovery of the structure of DNA in the 1950s, biological research has, quite understandably, been focused on the molecular level. While this has led to tremendous insights, it has inadvertently lessened the importance of the cell as a structural unit. And yet, experimental biology is still observed at the multi-cell level. This is where, for example, we see cells move, divide, adhere, secrete, aggregate, communicate and die. It therefore seems logical to conduct *computational* biology at a multi-cell level as well. Sydney Brenner, a Nobel laureate in Physiology or Medicine, has stated [10]:

> I believe very strongly that the fundamental unit, the correct level of abstraction, is the cell and not the genome.

*e-mail:heiland@indiana.edu
†e-mail:maciekswat@gmail.com
‡e-mail:jsluka@indiana.edu
§e-mail:bzaitlen@indiana.edu
¶e-mail:ashirinifard@gmail.com
‖e-mail:glt@if.ufrgs.br
**e-mail:lums@indiana.edu
††e-mail:glazier@indiana.edu

Wolpert [14] offers a fascinating story for nonspecialists about the lives of cells, explaining mitosis, growth, gastrulation, cancer, and much more.

We provide an overview of CompuCell3D (CC3D; compucell3d.org), a software application to simulate the behaviors of *generalized cells*. Our focus in this paper is on the visualization of those cells and their connectivity with each other. CompuCell3D is primarily used to develop models for multi-cellular biology, however, it is also used for non-biological models, as we will see.

Another topic that we discuss is the Python programming language and the key role that it plays in CompuCell3D.

2 COMPUCELL3D

CC3D [8][12] is an open source software application to simulate models that have generalized cells as their fundamental objects. Users can download the source code and build it themselves or download binaries that are ready to run. The application comes with several example models. Documentation and a community forum are accessible from the web site.

All CC3D models are based on the more general Glazier-Graner-Hogeweg (GGH) mathematical model [3]. GGH is defined on a uniform lattice domain and each generalized cell is a spatially defined contiguous subset of pixels (voxels) on the 2D (3D) lattice. A 2D lattice can be either square or hexagonal (with 3D analogues). Therefore, for relatively small lattices, rendered cells can be quite pixelated. Each generalized cell shares a common *cell index* (σ) and is of a defined *cell type* (τ). There will be very few cell types compared to the large number of cell indices. For example, in the human body, while there are trillions of cells, there are on the order of a few hundred cell types.

GGH uses an energy-based formalism to describe cell behaviors. It evolves cells using a (local) energy minimization algorithm with a Boltzmann acceptance function that simulates a constant temperature. The *effective energy* (H) is defined as follows:

$$H = \sum_{i,j} J(\tau(\sigma_i), \tau(\sigma_j))(1 - \delta(\sigma_i, \sigma_j)) + \sum_{\sigma} \lambda_{vol}(\sigma)(v(\sigma) - V_t(\sigma))^2 + ... \quad (1)$$

where i, j are neighbor lattice sites and σ and τ were defined above (cell index and cell type). The first sum in H calculates the *adhesion energy* between neighboring cells. Higher adhesion energies result in cell-cell repulsion whereas lower adhesion energies cause cells to adhere. The second sum in H represents a volume conservation constraint. This term will cause cells to reach a user-specified target volume and is defined in CC3D as a *plugin*. There are numerous plugins available in CC3D, each one contributing to a particular behavior of a cell, e.g. volume, surface area, polarity, etc.. More details about GGH can be found in the literature [12][11]. Most of the code base for CC3D implements the GGH algorithm and is written in C++. However, Python is also used extensively.

The CC3D graphical user interface (GUI) application is depicted in Figure 1. CC3D is built on an open source software stack that includes: Python (scripting), numpy (numerics), Qt and PyQt (GUI), SWIG (wrapping C++ code in Python), and the Visualization Toolkit (VTK; also wrapped in Python). VTK is well-known to the scientific visualization community. It uses a pipeline model to build a visualization: $data \rightarrow filter \rightarrow mapper \rightarrow actor \rightarrow render$. With dozens of filters available, there is considerable flexibility in creating a visualization, as we shall see.

CC3D is a multi-threaded application inside the Python interpreter, allowing the simulation to run while a user operates the GUI. In addition to making interactive changes to the visualization, it is also possible to *steer* the simulation by changing model parameters. Python provides a very flexible and extensible environment. Developers can rapidly prototype new ideas for visualizations, for example. But more importantly, users can create extremely flexible models by writing their own Python *steppables*. A Python steppable is simply a Python module that gets executed inside the GGH algorithm. The user can specify how frequently it is invoked during a simulation.

In addition to providing the flexibility of writing Python steppables that affect the model, CC3D also makes it possible for users to write Python-wrapped VTK scripts. These will allow custom visualizations to be rendered in the GUI. Template scripts are provided to help users get started, just as we provide template steppables.

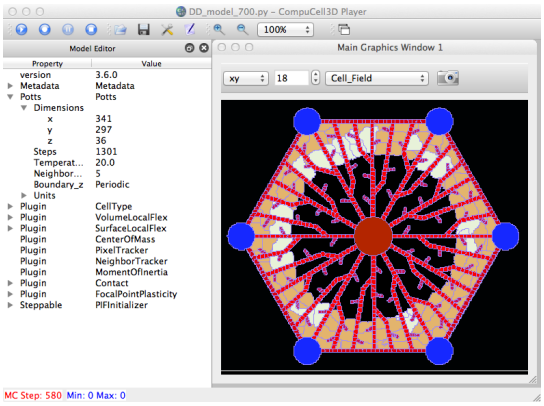

Figure 1: CC3D application.

3 Example Simulations

We present three example simulations from CC3D. Two are of biological models and the third is non-biological, but demonstrates visualization techniques that are relevant to biology.

3.1 Cell sorting

Cell sorting is a biologial phenomenon whereby cells of a particular cell type adhere to one another (or to another cell type) to varying degrees, leading to certain patterns. CC3D provides several example models of cell sorting (both 2D and 3D). By changing the adhesion energies between cell types, we can simulate different cell sorting behaviors. Previously, we reported on the extensive variety of patterns from a parameter sweep of the 2D cell sorting model [5]. In Figure 2, we show four snapshots in time from one such 2D cell sorting simulation. (Note that CC3D simulations have *Monte Carlo steps (MCS)* as units of time.) The parameters of the model were chosen so that the less cohesive *noncondensing* type of cells engulf the more cohesive *condensing* type. The latter tend to form clusters and eventually form a single central cluster. The default visualization for 2D cells is to color all cells of the same cell type with a user-specified color (Figure 3) and to draw a boundary line around each cell (also with a user-specified color). Another option is to render cells as glyphs (Figure 4), possibly scaled by volume (note that we use "volume" as a cell attribute in CC3D; it refers to "area" for 2D cells). The boundaries of the 2D cells are still visible, but this too is optional. Glyphs can also be used for 3D simulations, as we shall see in the next example. When there are a very large number of cells, low-resolution glyphs can be used to speed up the rendering step.

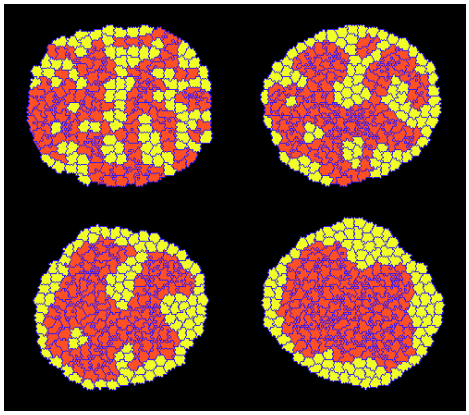

Figure 2: Four snapshots of a 2D cellsort simulation.

Type	Name	Color
0	Medium	
1	Condensing	
2	NonConden...	

Figure 3: GUI for changing colors of cell types.

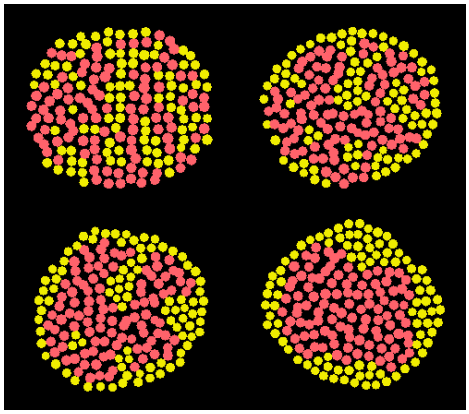

Figure 4: Cells represented as circular glyphs.

If a user includes a plugin that tracks neighbor cells in their model, CC3D will have access to all neighbors of each cell. Moreover, access to the entire list of cells and their neighbor lists will be available from Python. As a result, a Python steppable can be written that accesses all cells' neighbors, giving the user considerable control for fine-tuning a model. It also makes possible a graph analysis of cell-neighbor connectivity which could lead to metrics for determining the state of a simulation, as has been demonstrated elsewhere in a different context [9].

To perform a graph analysis, we use NetworkX [4], an open source Python-based software package for creating and analyzing networks. While not a part of CC3D proper, it demonstrates the power of Python's extensibility. A user can simply install NetworkX into their Python environment and thereby make it available to a CC3D steppable. Moreover, a Python plotting package, matplotlib [7], can also be installed and used to easily draw graphs defined in NetworkX. To demonstrate, we performed a graph analysis of cell neighbors for the 2D cell sorting simulation. Some results for each of the four snapshots are listed in Table 1. The *connected components* is perhaps the most meaningful graph measure to serve as a metric for the simulation's state.

Figure 5 shows plots of the graphs for MCS=20 using a *spring* layout of the nodes. Figure 6 are plots of the same graphs, but using the cells' centers of mass as positions for the nodes.

The following code snippet is from the NetworkX Python script that retrieves relevant information for an undirected graph, G1, that represents the neighbor connectivity for cells of a particular cell type.

```
G1.number_of_nodes()
G1.number_of_edges()
max(nx.degree(G1).values())
len(nx.connected_components(G1))
```

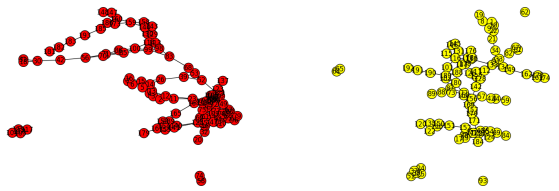

Figure 5: Spring layout of cell neighbors connectivity graph for condensing (left) and noncondensing (right) cells (MCS 20).

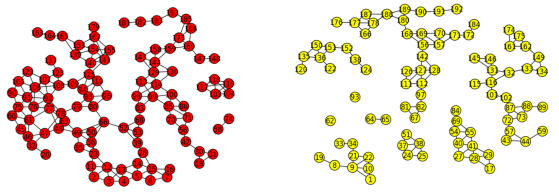

Figure 6: Position layout (cells' centers) of same graphs in Figure 5.

Table 1: Graph analysis for 2D cell sorting (193 cells).

Cell Type (count)	MCS	# edges	max deg	# conn comp	biconnected
condensing (103)	20	196	7	3	False
	1020	260	9	2	False
	1960	281	9	1	False
	9040	327	11	1	True
noncondensing (90)	20	117	5	9	False
	1020	135	6	6	False
	1960	148	7	2	False
	9040	175	7	1	True

3.2 Liver lobule

The Virtual Liver is a research project of the US Environmental Protection Agency that seeks to develop and test computer models to estimate the potential of chemicals to cause chronic diseases in the human liver [13]. The basic functional unit of the human liver is a hexagonal-shaped structure called the hepatic lobule. The lobule contains a large central vein, a network of blood vessels (called sinusoids), portal venules at the vertices of the hexagon, and hepatocyte cells that make up the bulk of the liver mass. Each of these components have a different cell type in CC3D. Fig. 1 shows a 2D slice during the simulation of a liver lobule model. In the model, hepatocytes (beige colored) grow in between the sinusoids from the hexagon boundary inward; hepatocytes in the process of growing (pre-mitosis) are colored white.

Three-dimensional renderings of the liver lobule are shown in Figures 7-9. Figure 7 show isocontours for each hepatocyte cell (each cell id), together with a single contour of the entire sinusoid network (isovalue = sinusoid cell type), and another single contour of the surrounding portal venules. In Figure 8, we use two different types of glyphs (ellipse and superquadric) to represent the (fully formed) hepatocytes, but continue to use isocontours for the growing hepatocytes (colored white). In the left image, the glyphs have no orientation; whereas in the right image, they are oriented according to a simple vector field that emanates from the center of the lobule. Figure 9 shows a final rendering that re-incorporates the vasculature and inserts cell nucleii (small spherical glyphs). We also show a single growing hepatocyte cell rendered as an isocontour in Figure 10. In all three figures, we insert clipping planes into the VTK pipeline to reveal the inner structures of the lobule.

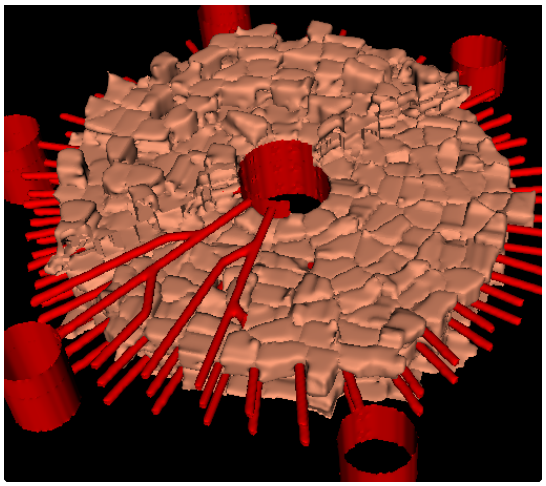

Figure 7: Contours for individual hepatocytes.

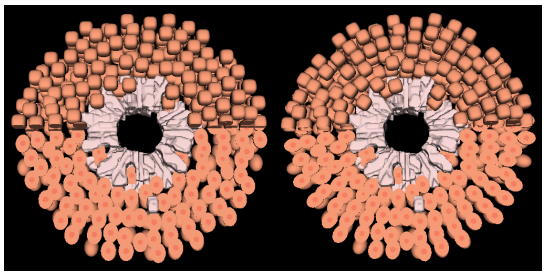

Figure 8: Nonoriented (left) and oriented (right) glyphs for hepatocytes, together with contours of growing hepatocytes.

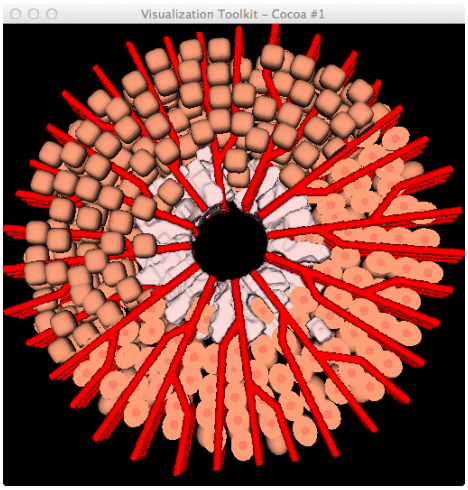

Figure 9: Liver lobule with mixture of glyphs, contours and cutting planes.

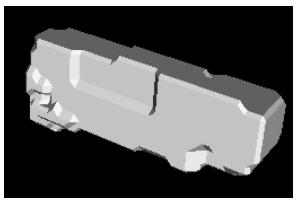

Figure 10: Single growing hepatocyte cell embedded in the lattice.

3.3 Foams

While the primary use of CC3D is to simulate models of cell behavior for developmental biology, recall that the GGH algorithm operates on *generalized* cells. In this example, we show results from foam simulations [1] where foam bubbles are the cells. For a dry foam simulation, there is only one cell type (bubble); for wet foams, there are two (bubble and water). Foams can undergo a process called coarsening in which some bubbles grow, others shrink and disappear, causing the number of bubbles to decrease over time. Coarsening is also found in metallic grains and polycrystals.

CC3D provides example models for foams. Figure 11 is an image from a 2D dry foam simulation, showing the bubble boundaries and circular glyphs scaled by bubble area. Also shown is an image from a graph analysis that highlights those bubbles (nodes) with an area smaller than neighboring bubbles. This provides a good prediction of which bubbles will be eliminated.

Figure 12 shows the progression of a 3D dry foam coarsening simulation, using semi-transparent contours of the bubbles. Figure 13 depicts two different renderings of a 3D wet foam simulation. On the left, we contour individual bubbles (cell ids) and render opaque surfaces. On the right, we eliminate those bubbles that intersect the lattice boundary and render only the interior bubbles, semi-transparently. Figure 14 show the progression of a 3D wet foam coarsening simulation using this second technique.

4 Model Fabrication

As a final example, we present a model of a 3D tumor and a surrounding vasculature network [11]. Figure 15 shows three snapshots from the simulation. Because this model was still in its development phase, the lattice domain was relatively small (100x100x160). This created challenges for creating presentation-quality visualizations. Fortunately, VTK's smoothing filters helped

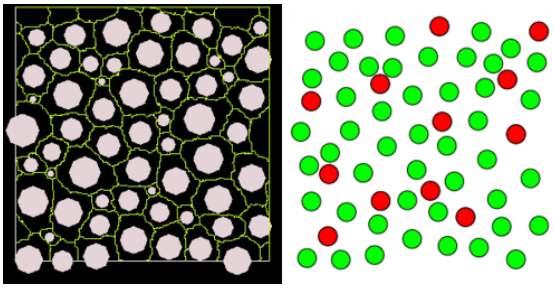

Figure 11: 2D foam: bubble boundaries and area-scaled glyphs (left); graph analysis (right) showing nodes (red) with bubble volume less than each of its neighbor's.

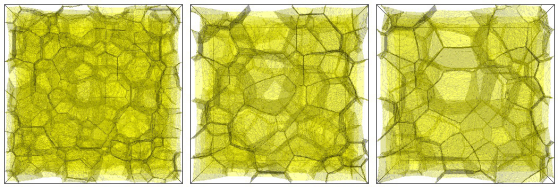

Figure 12: Semi-transparent renderings of 3D dry foam coarsening.

considerably. Figure 16 shows a zoom-in of the tumor's necrotic core before smoothing and after. During the development of this model, we also had an opportunity to fabricate it using a 3D printer.

Fab labs [2] are, in some ways, the manufacturing analogue to the open source software movement. Digital fabrication has become much more economical in recent years which has spurred a growing interest from a variety of people – industrial designers, engineers, musicians, and sculptors, to name a few. Products range from practical, industrial widgets to whimsical, artistic creations. Fab labs offer a variety of machines, including rapid prototyping and other 3D printers, lathes, various cutters, and more. Indiana University is in the process of creating its own fab lab and we were fortunate enough to have access to a 3D printer.

Using one of VTK's *Writer* objects, we were able to write a stereo lithography (STL) file of the 3D tumor model. The STL model captured contours of cell types, with cutting planes to reveal an inner necrotic core of the tumor (Figure 17). Since part of the vasculature was discontinuous, we needed to manually insert artificial (spherical) connectors into the model. Generating a 3D physical model provides an interesting manipulative object to demonstrate at CC3D workshops and to use in K-12 school classrooms and public outreach settings.

5 Related Work

When considering related work to CC3D, one must distinguish between the code that simulates the cells (the GGH algorithm) and the GUI that performs the visualization. We will address only the latter, as the former has been addressed in the literature. Since we use VTK as our primary visualization library, it would seem reasonable to adopt an existing VTK-based visualization application. ParaView [6] is one such open source application, from Kitware, the same company that develops VTK. The reasons we chose to develop our own GUI for CC3D is (1) we can customize the visualization choices to fit our application and (2) we need to provide synchronized, interactive visualizaton and co-processing for a running simulation. If we discover that ParaView or any other VTK-based application can help meet these two criteria, we would seriously consider adopting it.

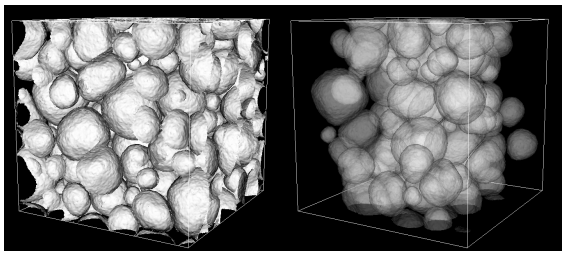

Figure 13: Comparing contouring and rendering options.

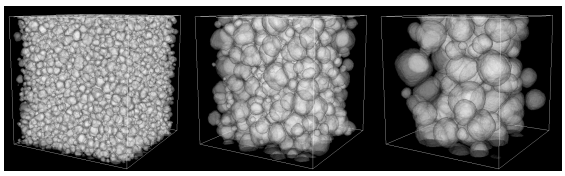

Figure 14: Three snapshots of a wet foam simulation.

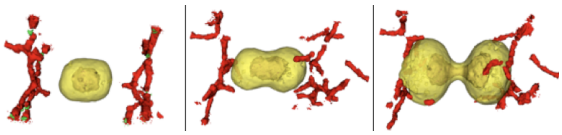

Figure 15: Three snapshots from the 3D tumor simulation.

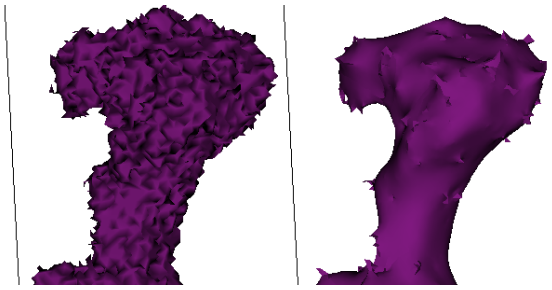

Figure 16: Applying a smoothing filter to the tumor's necrotic core.

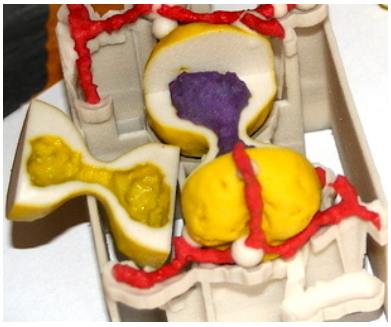

Figure 17: Tumor model from a 3D printer.

6 Conclusion

We have provided an overview of CompuCell3D (CC3D), an open source application for simulating and visualizing models of generalized cells, with applications in both biology and physics. The fact that CC3D uses the lattice-based GGH model means that cells have the potential to be quite pixelated when visualized. A variety of rendering options in CC3D, including the use of glyphs and smoothed surfaces, have been demonstrated. CC3D uses the Visualization Toolkit (VTK) as its primarily visualization library. VTK offers a simple pipeline approach for rendering data and a rich set of filters that can improve the final rendered image.

Although CC3D is written primarily in C++, we have extensively incorporated the Python programming language as well. In addition to the graphical user interface being in Python (PyQt), CC3D also provides a mechanism for extending and fine-tuning a model via Python *steppables*. A steppable can also create or modify visualizations.

Because Python is easily extensible, we were also able to demonstrate how one could perform a graph analysis on cells' neighbors connectivity using the NetworkX Python package. We believe this approach has the potential to define metrics for a simulation's progress and outcome.

We welcome ideas from and conversations with other researchers in both the computational biology and visualization communities.

Acknowledgements

The authors gratefully acknowledge support from the National Institutes of Health, National Institute of General Medical Sciences grants R01 GM077138 and R01 GM076692, the Environmental Protection Agency grant R835001, the Lilly Endowment Inc. and the Biocomplexity Institute at Indiana University. Indiana Universitys Research Technologies (RT/UITS), provided time and assistance with BigRed and Quarry clusters for simulations. The Advanced Visualization Lab in RT helped generate the tumor model on their 3D printer. Early versions of CompuCell3D were developed at the University of Notre Dame by JAG, Dr. Mark Alber, Dr. Jesus Izaguirre, Joseph Coffland, and collaborators with the support of National Science Foundation, Division of Integrative Biology, grant IBN-00836563. Other developers (current and past) from Indiana University include Mitja Hmeljak, Chris Mueller and Alex Dementsov. We wish to thank all the open source software communities who have provided a foundation for CC3D and for the results in this paper: Python, numpy, IPython, VTK, NetworkX and matplotlib. Thanks also to Nick Edmonds and Clayton Davis for useful discussions about graphs. Most of all, we thank the community of CC3D modelers who have been our collaborators throughout its development.

References

[1] I. Fortuna, G. L. Thomas, R. M. C. de Almeida, and F. Graner. Growth laws and self-similar growth regimes of coarsening two-dimensional foams: Transition from dry to wet limits. *ArXiv e-prints*, Mar. 2012.

[2] N. Gershenfeld. *Fab: The Coming Revolution on Your Desktop–from Personal Computers to Personal Fabrication*. Basic Books, 2005.

[3] F. Graner and J. A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Phys. Rev. Lett.*, 69:2013–2016, Sep 1992.

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug. 2008.

[5] R. Heiland, M. Swat, B. Zaitlen, J. Glazier, and A. Lumsdaine. Workflows for parameter studies of multi-cell modeling. In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim '10, pages 94:1–94:6, San Diego, CA, USA, 2010. Society for Computer Simulation International.

[6] A. Henderson. *ParaView Guide, A Parallel Visualization Application*. Kitware Inc., 2007.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, May-Jun 2007.

[8] J. A. Izaguirre, R. Chaturvedi, C. Huang, T. M. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. S. Alber, G. Hentschel, S. A. Newman, and J. A. Glazier. Compucell, a multi-model framework for simulation of morphogenesis. *Bioinformatics*, pages 1129–1137, 2004.

[9] L. McKeen-Polizzotti, K. M. Henderson, B. Oztan, C. C. Bilgin, B. Yener, and G. E. Plopper. Quantitative metric profiles capture three-dimensional temporospatial architecture to discriminate cellular functional states. *BMC Med Imaging*, 11:11, 2011.

[10] D. Noble. *The Music of Life: Biology Beyond Genes*. Oxford University Press, 2008.

[11] A. Shirinifard, J. S. Gens, B. L. Zaitlen, N. J. Popawski, M. Swat, and J. A. Glazier. 3d multi-cell simulation of tumor growth and angiogenesis. *PLoS ONE*, 4(10):e7190, 10 2009.

[12] M. H. Swat, S. D. Hester, A. I. Balter, R. W. Heiland, B. L. Zaitlen, and J. A. Glazier. Multicell simulations of development and disease using the CompuCell3D simulation environment. *Methods Mol. Biol.*, 500:361–428, 2009.

[13] J. Wambaugh and I. Shah. Simulating microdosimetry in a virtual hepatic lobule. *PLoS Comput Biol*, 6(4):e1000756, 04 2010.

[14] L. Wolpert. *How we live and why we die: the secret lives of cells*. Faber and Faber, 2009.