Cell Modeling in Jupyter Notebook using CompuCell3D

TRINITY CHUNG[1]*, T. J. SEGO[2], JAMES A. GLAZIER[3]

[1]*University of California, Berkeley, CA 94720, USA.*

[2]*Department of Medicine, University of Florida, Gainesville, FL 32611, USA.*

[3]*Biocomplexity Institute and Department of Intelligent Systems Engineering, Indiana University, Bloomington, IN 47408, USA.*

**trinityc@berkeley.edu*

Abstract: CompuCell3D (CC3D) is an open-source software framework for building and executing multi-cell biological virtual-tissue models. It represents cells using the Glazier–Graner–Hogeweg model, also known as Cellular Potts model. The primary CC3D application consists of two separate tools, a smart model editor (Twedit++) and a tool for model execution, visualization and steering (Player). The CompuCell3D version 4.x release introduces support for Jupyter Notebooks, an interactive computational environment, which brings the benefits of reproducibility, portability, and self-documentation. Since model specifications in CC3D are written in Python and CC3DML and Jupyter supports Python and other languages, Jupyter can naturally act as an integrated development environment (IDE) for CC3D users as well as a live document with embedded text and simulations. This update follows the trend in software to move away from monolithic freestanding applications to the distribution of methodologies in the form of libraries that can be used in conjunction with other libraries and packages. With these benefits, CC3D deployed in Jupyter Notebook is a more natural and efficient platform for scientific publishing and education using CC3D.

Keywords: CompuCell3D, Jupyter Notebook, cell modeling, virtual tissue, biology education

Introduction

Computational biological modeling enables scientists to quantitatively describe complex biological systems, test existing biological knowledge and generate new biological hypotheses. Cell-based computational biological modeling describes biological systems on the basis of individual cells [1], and can include dynamic deterministic or stochastic descriptions of cell location and state. Research using computational modeling has produced novel quantitative descriptions of the underlying mechanisms of many developmental processes like somitogenesis [2], vasculogenesis [3, 4] and gastrulation [5], and has provided new biomedical insights in various problems of health and disease such as acetaminophen metabolism [6], autosomal dominant polycystic kidney disease [7], antiviral therapies [8, 9], and influenza infection and host-pathogen interactions [10].

There exist multiple software for cell-based computational biological modeling, each of which employs various numerical methods and provides different features and specializations, including Artistoo [11], Biocellion [12], CHASTE [13], Morpheus [14], PhysiCell [15], Simmune [16], and TissueSimulationToolkit [17] among others. CompuCell3D (CC3D) [18], which implements the Glazier–Graner–Hogeweg model [19], also known as Cellular Potts model, to simulate multicellular systems, has a long legacy as an open-source, cross-platform modeling and simulation environment meant to be accessible to all levels of biologists, from students to veteran researchers. To provide accessibility to a broad user base across multiple disciplines, CC3D is distributed with supporting graphical user interfaces (GUIs) that streamline model and simulation specification (e.g., automated project generators, code snippets and browsable documentation) and project sharing (e.g., well-defined project file structures, built-in project archive import/export) and provide interactive simulation execution and real-time, customizable data visualization.

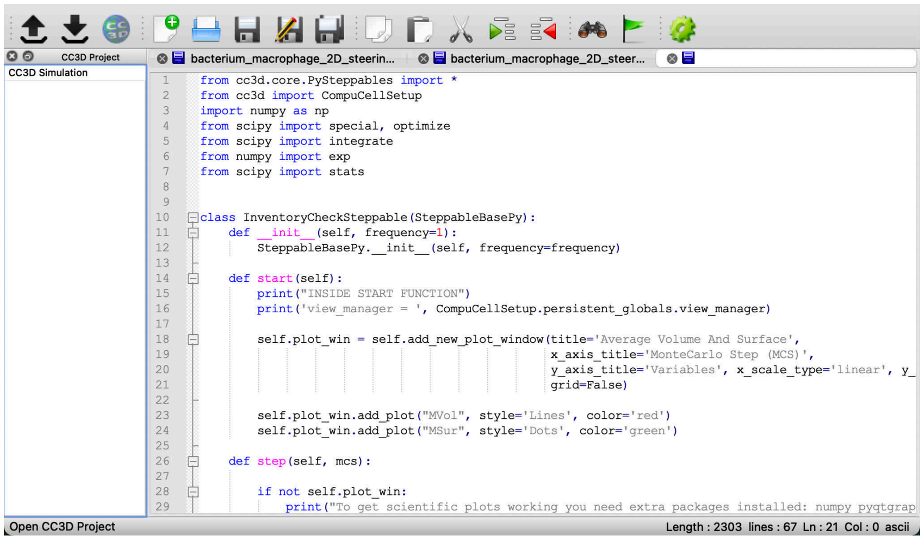
CC3D version 4.x development added support for Jupyter Notebooks, an interactive computational environment that provides the benefits of reproducibility, portability, and self-documentation in the form of executable, interactive scripts in a web browser. Jupyter Notebook acts as one integrated environment, in which text and graphics can be presented alongside executable code [20]. Jupyter was created in 2013 and is the most widely used computational notebook [21], with more than 2.5 million notebooks in GitHub since September of 2018 [22]. While CC3D was originally developed to support model specifications written in Python and CC3DML (an XML-based language), recent CC3D developments expanded deployment support to include specifications defined purely in Python. Since Jupyter supports Python and other languages, Jupyter can naturally act as an integrated development environment (IDE) for CC3D users as well. This update follows the current trend in software development and distribution to move away from monolithic, freestanding applications and towards the distribution of libraries that can be used in conjunction with other software. With these benefits, CC3D deployed in Jupyter Notebook is a powerful resource to expose students to computational biology as well as for scientists publishing work using CC3D. In this paper, we describe the basic features of CC3D simulation visualization in Jupyter Notebook as relevant to students, educators and scientists interested in integrating Jupyter-based computational biological modeling and simulation into their classrooms and projects.

Software Design

CC3D is written in C++ and provides Python language bindings, including an interface for runtime simulation execution and control in Python. The current distribution of the CC3D software includes two GUIs: Twedit++, a text editor for writing model specification code controlling the simulation (Fig. 1.1), and CC3D Player, a freestanding application to run and interact with a simulation (Fig. 1.2). Twedit++ and CC3D Player are both built on the PyQt framework, and CC3D Player uses the CC3D Python interface for runtime simulation execution. Likewise, the CC3D visualization pipeline employs infrastructure from the Visualization Toolkit (VTK) [23], which also provides Python language bindings and supporting widgets for interactive visualization in a Jupyter Notebook. These important software features permit both control of CC3D simulation execution and real-time rendering and visualization of CC3D simulation data within Jupyter. Furthermore, CC3D Player integrates VTK support for PyQt to provide interactive visualization, while VTK also provides implementations of those same widgets deployed in CC3D Player but for Jupyter Notebook. CC3D support for Jupyter Notebook integrates such features from VTK to provide a comparable user experience between simulation execution and visualization in CC3D Player and a Jupyter Notebook.

Player offers flexible visualization specifically tailored to the needs of CC3D users. While general standard visualization tools exist, they can be cumbersome for non-expert users. Player provides a simple way of representing complex visual data in real time by allowing the user to run/pause/stop execution of the simulation while rendering the graphics in real time. Users can create multiple graphics windows to view different properties, locations and objects of the simulation simultaneously. Rendering settings such as outlines, colors, bounds, and more can be configured to communicate the information the user needs. The view on the frames can be manipulated directly during the simulation with natural controls for pan, tilt, and zoom, switch between two- and three-dimensional views and setting view coordinates for two-dimensional visualization. Player also supports on-demand rendering of visualized simulation data and saving to file, as well as scheduling rendering at regular intervals of simulation time.

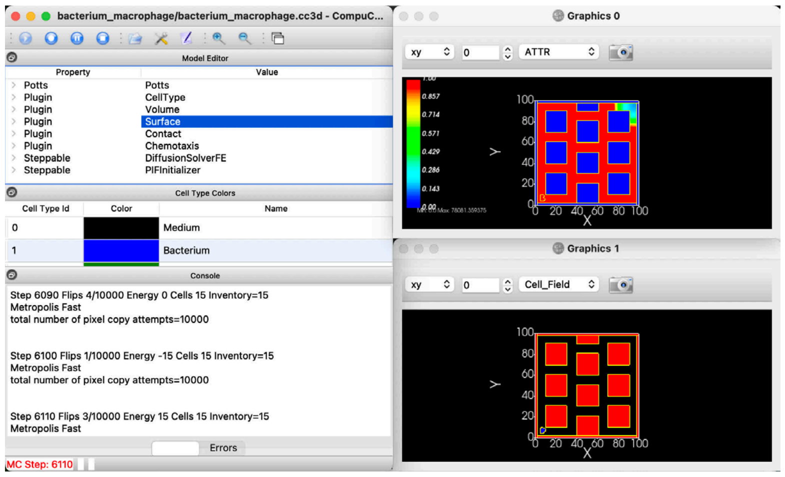
The Jupyter Notebook implementation of CC3D includes the rich features provided by Player, but in an interactive environment supporting user-specified visualization along with the implementation of their models, simulation specifications and documentation. Users can specify an arbitrary number of graphics frames, each of which can be individually customized to visualize simulation data in ways that complement, clarify or demonstrate the ideas communicated in their Jupyter Notebook. Each graphics frame can be interactively configured, and each graphics frame configuration can be stored to file during development in a human-readable JSON format and reloaded during subsequent executions of a Jupyter Notebook (e.g., when shared with others). CC3D visualization in Jupyter Notebook supports displaying individual graphics frames or grids of frames that effectively communicate complex simulation data over multiple fields (e.g., reaction-diffusion fields), which are common to models developed in CC3D that target intercellular signaling and/or dynamic environmental conditions. To this end, CC3D promotes deployment of multiple interactive visualization frames through multiprocessing and an application-specific message-passing interface. Visualization of each frame is executed in a separate process and message passing shares serialized simulation data from the computational core to each visualization process, and likewise instructions from user interactions are shared from each visualization process back to the computational core for real-time manipulation of data visualization (Fig. 1.3).



CC3D Project
CC3D Simulation
bacterium_macrophage_2D_steerin...
bacterium_macrophage_2D_steer...
Open CC3D Project
Length : 2303 lines : 67 Ln : 21 Col : 0 ascii

```python
from cc3d.core.PySteppables import *
from cc3d import CompuCellSetup
import numpy as np
from scipy import special, optimize
from scipy import integrate
from numpy import exp
from scipy import stats


class InventoryCheckSteppable(SteppableBasePy):
    def __init__(self, frequency=1):
        SteppableBasePy.__init__(self, frequency=frequency)

    def start(self):
        print("INSIDE START FUNCTION")
        print('view_manager = ', CompuCellSetup.persistent_globals.view_manager)

        self.plot_win = self.add_new_plot_window(title='Average Volume And Surface',
                                                 x_axis_title='MonteCarlo Step (MCS)',
                                                 y_axis_title='Variables', x_scale_type='linear', y_
                                                 grid=False)

        self.plot_win.add_plot("MVol", style='Lines', color='red')
        self.plot_win.add_plot("MSur", style='Dots', color='green')

    def step(self, mcs):

        if not self.plot_win:
            print("To get scientific plots working you need extra packages installed: numpy pyqtgrap
```

Fig. 1.1. Editing a simulation file using Twedit++ on MacOS



Model Editor

Property	Value
Potts	Potts
Plugin	CellType
Plugin	Volume
Plugin	Surface
Plugin	Contact
Plugin	Chemotaxis
Steppable	DiffusionSolverFE
Steppable	PIFInitializer

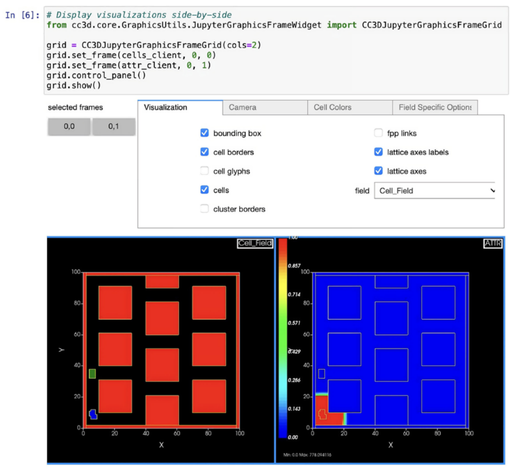
Cell Type Colors

Cell Type Id	Color	Name
0	⬛	Medium
1	🟦	Bacterium

Console

```
Step 6090 Flips 4/10000 Energy 0 Cells 15 Inventory=15
Metropolis Fast
total number of pixel copy attempts=10000


Step 6100 Flips 1/10000 Energy -15 Cells 15 Inventory=15
Metropolis Fast
total number of pixel copy attempts=10000


Step 6110 Flips 3/10000 Energy 15 Cells 15 Inventory=15
Metropolis Fast
```

Fig. 1.2. Visualizing a simulation in CompuCell3D Player on MacOS



```python
In [6]: # Display visualizations side-by-side
        from cc3d.core.GraphicsUtils.JupyterGraphicsFrameWidget import CC3DJupyterGraphicsFrameGrid

        grid = CC3DJupyterGraphicsFrameGrid(cols=2)
        grid.set_frame(cells_client, 0, 0)
        grid.set_frame(attr_client, 0, 1)
        grid.control_panel()
        grid.show()
```

✓ bounding box
☐ fpp links
✓ cell borders
✓ lattice axes labels
☐ cell glyphs
✓ lattice axes
✓ cells
field: Cell_Field
☐ cluster borders

Fig. 1.3. Visualizing a CompuCell3D simulation in a Jupyter Notebook

Previous CompuCell3D releases are primarily designed to run natively on Windows, Mac or Linux operating systems with a GUI. This meets the needs of a majority of users but makes remote client-server computing setups difficult since the remote server needs to render and stream its GUI. For instance, CompuCell3D version 4 is available through nanoHUB [24] and is rendered using the X11 default window manager (Fig. 2). The nanoHUB deployment of CC3D is fully functional and can demonstrate all the capabilities of CC3D, however, there are challenges to using it as a development environment. For one, the simulation execution speed on this platform is slower. In addition, since the remote desktop is an isolated environment, some basic computer functionality including keyboard shortcuts such as copy/paste are not transferable between the client and remote machine.

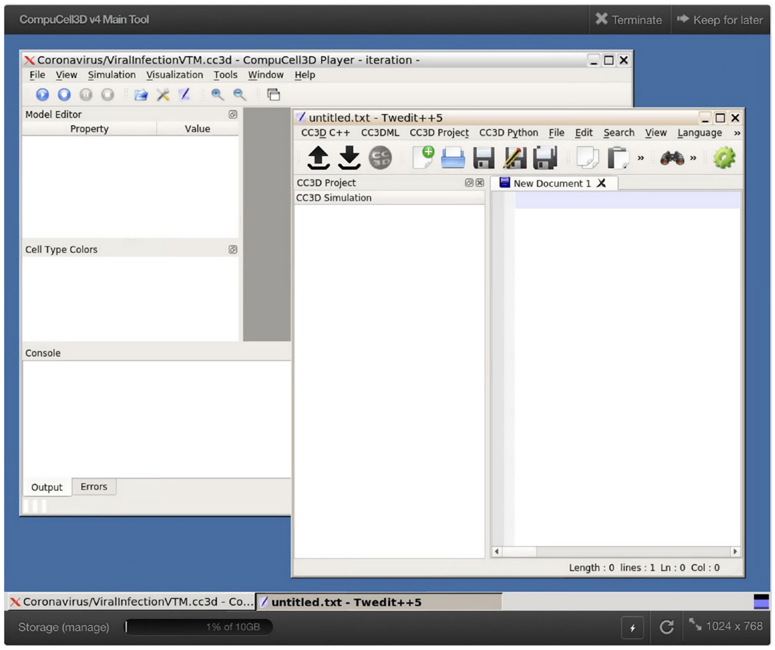

CompuCell3D v4 Main Tool
Terminate
Keep for later
Coronavirus/ViralInfectionVTM.cc3d - CompuCell3D Player - iteration -
File View Simulation Visualization Tools Window Help
Model Editor
Property
Value
Cell Type Colors
Console
Output
Errors
untitled.txt - Twedit++5
CC3D C++ CC3DML CC3D Project CC3D Python File Edit Search View Language
CC3D Project
CC3D Simulation
New Document 1
Length : 0 lines : 1 Ln : 0 Col : 0
Coronavirus/ViralInfectionVTM.cc3d - Co...
untitled.txt - Twedit++5
Storage (manage)
1% of 10GB
1024 x 768

Fig. 2. CompuCell3D Player and Twedit++ running on nanoHUB X11 window manager

On the other hand, a client-server interface is inherent to Jupyter, making Jupyter Notebooks that specify and describe a CC3D simulation easily deployable and efficient on cloud computing systems (Fig. 3). Jupyter can be hosted on a cloud server and accessed using a link by the client user. In such deployments, computations are performed on the server and the graphics are rendered locally, which can significantly improve user experience by increasing the performance of visualization interactivity.

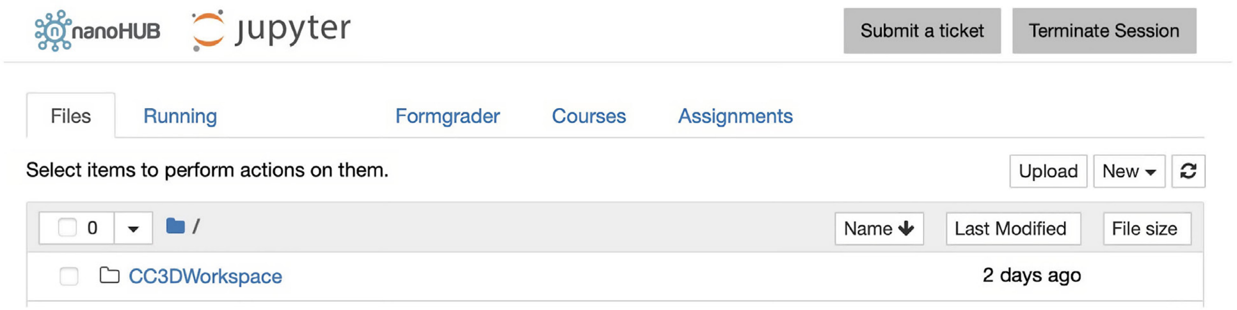


Fig. 3. Jupyter instance of nanoHUB with CompuCell3D files

Discussion

The flexible, portable, and reproducible format of Jupyter Notebook makes it an appealing environment to use as an educational resource and to supplement scientific publications. A single Jupyter Notebook can display instructions, figures, live code and simulations in a single browser window, forming a coherent "computational narrative" [20]. In comparison, working with the native CC3D application would require three different components, where code is written in Twedit++, simulations run in CC3D Player, and instructions are outlined in a separate document. In addition, the Twedit++ and CC3D Player applications have menus and features which may be irrelevant for a particular presentation, while a Jupyter Notebook can selectively load and display relevant components. This customizable and interactive format makes it simpler for audiences to follow and understand the concepts, technical details and overall scope of a CC3D-based project in Jupyter Notebook than with native CC3D (Fig. 4).

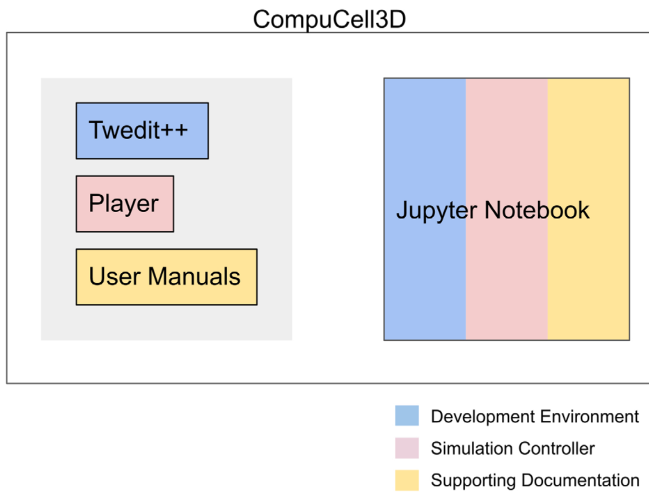
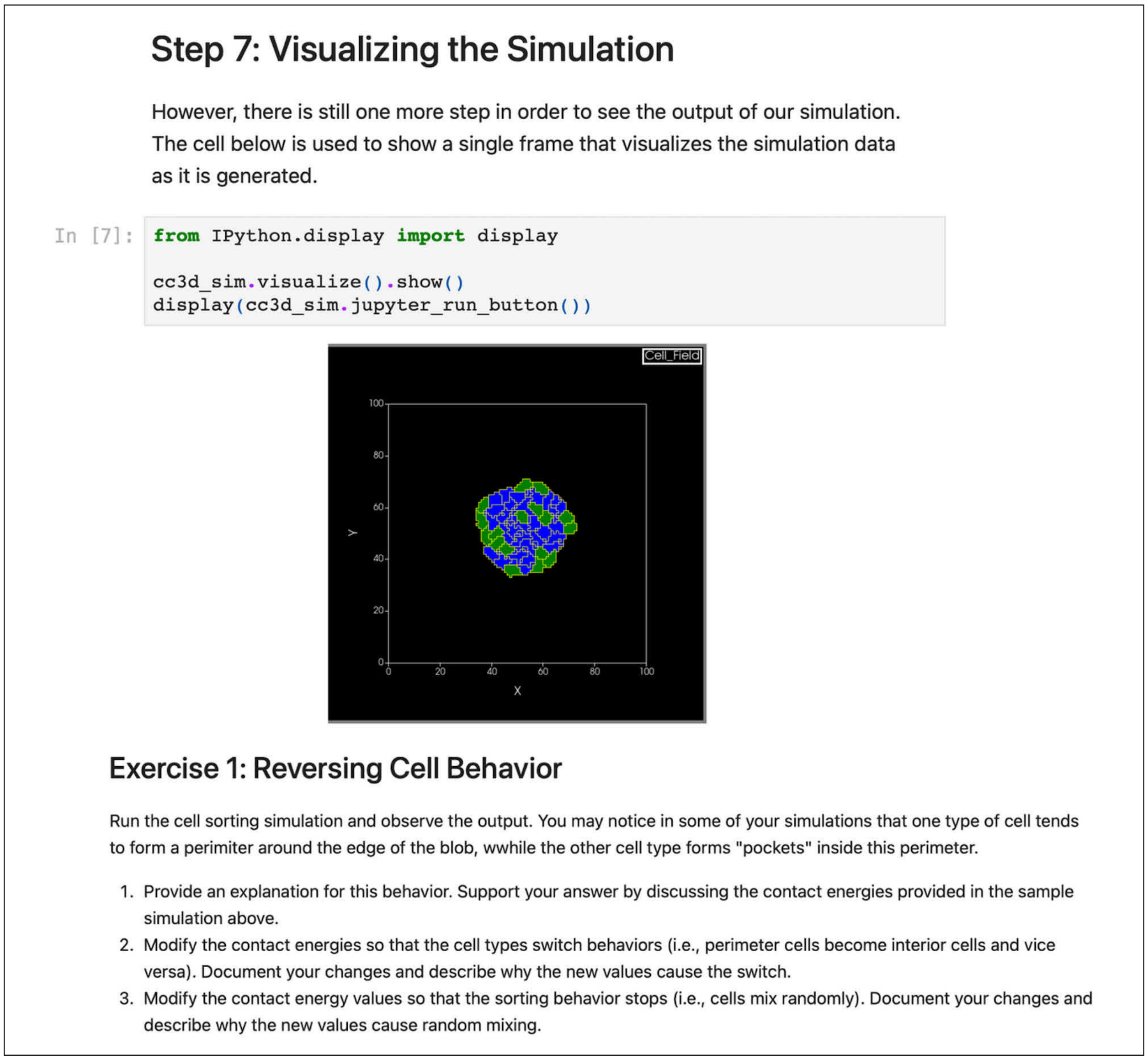


Fig. 4. With the CompuCell3D native applications, the development environment (Twedit++), simulation controller (Player), and supporting documentation (User Manuals) are separate. With Jupyter Notebook, these three aspects are seamlessly integrated as one.

The portability of Jupyter Notebook also makes it easier for teachers to distribute and collect Notebooks as assignments. Jupyter Notebook files may be uploaded onto file-sharing sites such as GitHub or other institutional platforms. In addition, existing online tools for Jupyter Notebooks such as nbviewer [25] allow for more ways to access/view a Notebook. With the flexibility and portability benefits that Jupyter offers, CC3D can be more effectively used in the classroom to demonstrate and reinforce biological concepts to students. Fig. 5. demonstrates an example of CC3D in Jupyter Notebook for educational use, including steps to run CC3D and additional exercises The full example notebook can be found in .pdf and Jupyter Notebook formats in the supplementary materials. In this fashion, a curriculum can be created with interactive Notebooks in place of worksheets. Previously, CC3D would have been difficult to deploy in classrooms because of its device-dependent and multicomponent nature. The development of CC3D for Jupyter Notebook enables a simpler replication and distribution process for instructors. The time to set up the CC3D environment for learning and to create new lesson materials would be reduced because of the streamlined format. The format benefits students as well: a Notebook allows a more focused way of learning rather than having to use and reference multiple applications to operate CC3D, as discussed above. The interactive format also encourages exploration, providing opportunity for a different kind of learning format. Previous research has shown that working with simulated models improves foundational and conceptual skills in biology students [26]. In addition, given the increasing need for online learning, simulations could be an alternative to exercises in a physical laboratory [27].



Step 7: Visualizing the Simulation

However, there is still one more step in order to see the output of our simulation. The cell below is used to show a single frame that visualizes the simulation data as it is generated.

```
In [7]:  from IPython.display import display

         cc3d_sim.visualize().show()
         display(cc3d_sim.jupyter_run_button())
```

Exercise 1: Reversing Cell Behavior

Run the cell sorting simulation and observe the output. You may notice in some of your simulations that one type of cell tends to form a perimiter around the edge of the blob, wwhile the other cell type forms "pockets" inside this perimeter.

1. Provide an explanation for this behavior. Support your answer by discussing the contact energies provided in the sample simulation above.
2. Modify the contact energies so that the cell types switch behaviors (i.e., perimeter cells become interior cells and vice versa). Document your changes and describe why the new values cause the switch.
3. Modify the contact energy values so that the sorting behavior stops (i.e., cells mix randomly). Document your changes and describe why the new values cause random mixing.

Fig. 5. Excerpts from the SortingDemo_ExtendedContent.ipynb file included in supplementary materials. Instructions for running the simulation and additional exercises are seamlessly integrated into one interactive environment through Jupyter Notebook.

Another beneficial use for CC3D in Jupyter Notebook is to embed CC3D simulations into research publications as supplementary media to help other researchers understand, interrogate and reproduce published work. As reports indicate, a current crisis of reproducibility across published science, especially for the field of computational biology [28], has filled much of the recent literature with works that cannot be reproduced because of factors like insufficient descriptions of methodology. CC3D support for Jupyter Notebook can thus help mitigate the waning reproducibility in computational biological modeling by providing a straightforward way for scientists to publish research in a format that makes it easier for others to trace, understand and reproduce their work. Since Jupyter Notebooks can be shared via accessible file hosting or cloud computing platforms like GitHub and nanoHUB, sharing or accessing a published Notebook is trivial, and many researchers are already familiar with these tools.

One limitation to using CC3D in Jupyter Notebook is that performance can vary depending on how much algorithmic work is done in Python. While the cost of pure backend calculations are unaffected, certain algorithms such as loops implemented in Python are less efficient than the C++ counterpart, which the Jupyter environment does not currently support. As a result, simulations have longer execution time than executed in the native desktop application, which may make Jupyter Notebook unsuitable for larger-scale simulations. For example, a simple benchmark running a 2D cell sorting demo demonstrated an execution time of 35 seconds in a Jupyter Notebook, compared to 2 seconds in Player and 0.5 seconds in a pure Python implementation (see Supporting Materials 1). While simulation performance varies depending on the machine and setup, and there are techniques to improve simulation performance, which is out of the scope of this paper, an implementation of a CC3D simulation may not be appropriate in Jupyter Notebook when developing, testing and applying computationally expensive algorithms specified in Python. However, a CC3D simulation implementation in a Jupyter Notebook still provides value for the purposes of sharing, demonstration and communication, and so CC3D-based research projects that do not primarily use Jupyter Notebook to generate published results should still provide a published implementation in a Jupyter Notebook to support reproducibility, as well as to showcase published work in an accessible and engaging medium.

Conclusion

CC3D is a software tool for computational biologists to develop, test and apply cell-based computational models of biological systems, while Jupyter Notebook is a generic interactive computing environment. Providing support for Jupyter in CC3D pushes computational biological modeling towards modern software practices and better accessibility for more users of different backgrounds. The Jupyter implementation of CC3D preserves the intuitiveness of CC3D Player while also serving as an IDE.

The authors encourage biology educators and student researchers to try using this new feature of CC3D and share any work done using this tool with the community of CC3D developers and users. Engagement and feedback allow the developers to continue improving and adding features which are most useful to users. In addition, learning resources will be continually added to support a growing user base.

Acknowledgments. We thank Hayden Fennell, postdoctoral researcher at Indiana University, for contributing his time and expertise to create an example Jupyter Notebook using CC3D for educational use.

We additionally thank Prof. Jared Ashcroft and the Micro Nano Technology Education Center of Pasadena City College for helping initiate this collaboration and for ongoing support. We acknowledge support from National Science Foundation grants NSF 2120200, NSF 2000281 and NSF 1720625 and National Institutes of Health grants U24 EB028887 and R01 GM122424.

Disclosures. The authors declare no conflicts of interest.

Supporting Materials. Source code for simulation benchmarks using CC3D Player, CC3D Python API and a Jupyter Notebook.

References

[1] R. M. H. Merks, J. A. Glazier, "A cell-centered approach to developmental biology." Phys. A: Stat. Mech. Appl. 352(1), 113-130 (2005).

[2] S. D. Hester, et al., "A multi-cell, multi-scale model of vertebrate segmentation and somite formation." PLoS Comput. Biol. 7(10), e1002155 (2011).

[3] R. M. H. Merks, et al., "Cell elongation is key to in silico replication of in vitro vasculogenesis and subsequent remodeling," Developmental Biology 289(1), 44-54 (2006).

[4] R. M. H. Merks, et al., "Contact-inhibited chemotaxis in de novo and sprouting blood-vessel growth," PLoS Comput. Biol. 4(9), e1000163 (2008).

[5] B. Vasiev, et al., "Modeling gastrulation in the chick embryo: formation of the primitive streak," PLoS One 5(5) e10571 (2010).

[6] J. P. Sluka, et al., "A liver-centric multiscale modeling framework for xenobiotics," PloS one 11(9), e0162428 (2016).

[7] J. M. Belmonte, et al., “Virtual-tissue computer simulations define the roles of cell adhesion and proliferation in the onset of kidney cystic disease,” Mol. Biol. Cell. 27(22), 3673-3685 (2016).

[8] T. J. Sego, et al., “A modular framework for multiscale, multicellular, spatiotemporal modeling of acute primary viral infection and immune response in epithelial tissues and its application to drug therapy timing and effectiveness,” PLoS Comput. Biol. 16(12), e1008451 (2020).

[9] J. F. Gianlupi, et al., “Multiscale Model of Antiviral Timing, Potency, and Heterogeneity Effects on an Epithelial Tissue Patch Infected by SARS-CoV-2,” Viruses 14(3), 605 (2022).

[10] T. J. Sego, et al., “A multiscale multicellular spatiotemporal model of local influenza infection and immune response,” J. Theor. Biol. 532, 110918 (2022).

[11] I. M. Wortel, and J. Textor, “Artistoo, a library to build, share, and explore simulations of cells and tissues in the web browser,” *ELife* 10, e61288 (2021), https://doi.org/10.7554/eLife.61288.

[12] S. Kang, S. Kahan, J. McDermott, N. Flann, and I. Shmulevich, “Biocellion: Accelerating computer simulation of multicellular biological system models,” Bioinformatics 30(21), 3101–3108 (2014), https://doi.org/10.1093/bioinformatics/btu498.

[13] J. Pitt-Francis, P. Pathmanathan, M. O. Bernabeu, R. Bordas, J. Cooper, A.G. Fletcher, G. R. Mirams, P. Murray, J. M. Osborne, A. Walter, S. J. Chapman, A. Garny, I. M. M. van Leeuwen, P.K. Maini, B. Rodríguez, S. L. Waters, J. P. Whiteley, H. M. Byrne, & D. J. Gavaghan, “Chaste: A test-driven approach to software development for biological modelling,” Comput. Phys. Commun. 180(12), 2452–2471 (2009), https://doi.org/10.1016/j.cpc.2009.07.019.

[14] J. Starruß, W. de Back, L. Brusch, A. Deutsch, “Morpheus: A user-friendly modeling environment for multiscale and multicellular systems biology,” Bioinformatics 30(9), 1331–1332 (2014), https://doi.org/10.1093/bioinformatics/btt772.

[15] A. Ghaffarizadeh, R. Heiland, S. H. Friedman, S. M. Mumenthaler, P. Macklin, “PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems,” PLoS Comput. Biol. 14(2), e1005991 (2018), https://doi.org/10.1371/journal.pcbi.1005991

[16] M. Meier-Schellersheim, G. Mack, “SIMMUNE, *a tool for simulating and analyzing immune system behavior*,” arXiv (1999), https://doi.org/10.48550/arXiv.cs/9903017.

[17] J. T. Daub and R. M. H. Merks, “Cell-Based Computational Modeling of Vascular Morphogenesis Using Tissue Simulation Toolkit,” in *Vascular Morphogenesis: Methods and Protocols*, D. Ribatti, ed., Methods in Molecular Biology (Springer, 2015), pp. 67–127, https://doi.org/10.1007/978-1-4939-1462-3_6.

[18] M. H. Swat, G. L. Thomas, J. M. Belmonte, A. Shirinifard, D. Hmeljak, and J. A. Glazier, “Multi-Scale Modeling of Tissues Using CompuCell3D,” in *Methods in Cell Biology*, A. R. Asthagiri and A. P. Arkin, eds., Computational Methods in Cell Biology (Academic Press, 2012), Vol. 110, pp. 325–366, https://doi.org/10.1016/B978-0-12-388403-9.00013-8.

[19] F. Graner, J. A. Glazier, “Simulation of biological cell sorting using a two-dimensional extended Potts model,” Phys. Rev. Lett. 69(13), 2013–2016 (1992), https://doi.org/10.1103/PhysRevLett.69.2013.

[20] T. Kluyver, B. Ragan-Kelley, F. Perez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, C. “Jupyter Notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, (IOS Press 2016), pp. 87–90, https://doi.org/10.3233/978-1-61499-649-1-87.

[21] H. Shen, “Interactive notebooks: Sharing the code,” Nature News 515(7525) 151 (2014).

[22] J. M. Perkel, “Why Jupyter is data scientists’ computational notebook of choice,” Nature 563(7729), 145–146 (2018), https://doi.org/10.1038/d41586-018-07196-1.

[23] F. Perez, B. E. Granger, “IPython: A System for Interactive Scientific Computing,” Comput. Sci. Eng. 9(3), 21–29 (2007), https://doi.org/10.1109/MCSE.2007.53.

[24] J. Gianlupi, T. J. Sego, "CompuCell3D v4 Main Tool (4.2.5.1)," nanoHUB. (2021), https://doi.org/10.21981/85GJ-SC30.

[25] M. Bussonnier, M. RK, "nbviewer: Jupyter Notebook Viewer," Project Jupyter (2012), https://nbviewer.org/.

[26] H. E. Bergan-Roller, N. J. Galt, C. J. Chizinski, T. Helikar, J. T. Dauer, "Simulated computational model lesson improves foundational systems thinking skills and conceptual knowledge in biology students," BioScience, 68(8), 612–621 (2018), https://doi.org/10.1093/biosci/biy054

[27] C. S. Pillay, "Analyzing biological models and data sets using Jupyter notebooks as an alternate to laboratory-based exercises during COVID-19," Biochem. Mol. Biol. Educ. 48(5), 532–534 (2020). https://doi.org/10.1002/bmb.21443.

[28] J. A. Papin, F. M. Gabhann, H. M. Sauro, D. Nickerson, A. Rampadarath, "Improving reproducibility in computational biology research," PLoS Comput. Biol. 16(5), e1007881 (2020), https://doi.org/10.1371/journal.pcbi.1007881.