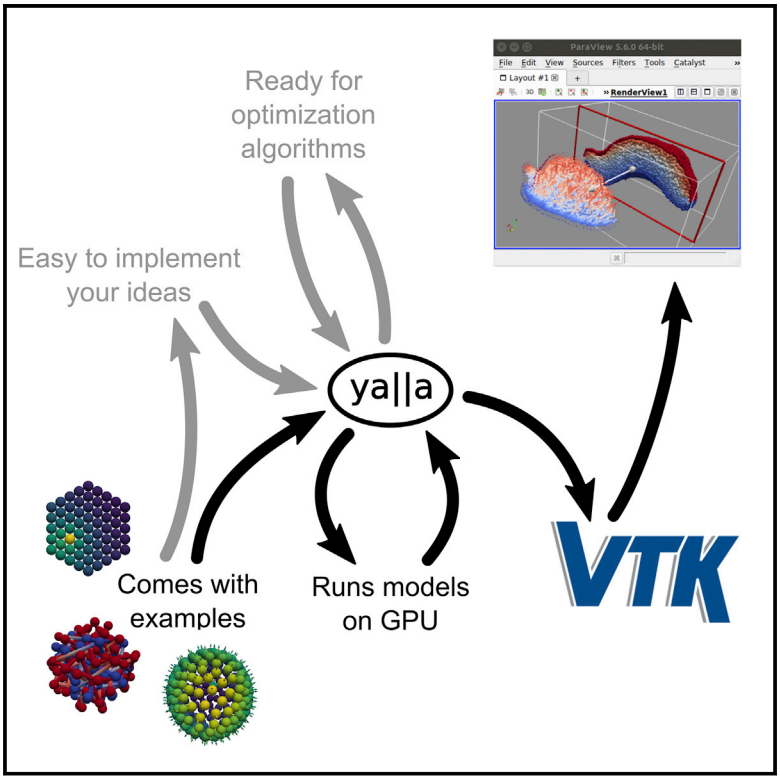
Brief Report

Cell Systems

ya||a: GPU-Powered Spheroid Models for Mesenchyme and Epithelium

Graphical Abstract



Authors

Philipp Germann, Miquel Marin-Riera, James Sharpe

Correspondence

philipp.germann@embl.es (P.G.), james.sharpe@embl.es (J.S.)

In Brief

ya||a simulates morphogenesis based on cellular behaviors in mesenchyme and epithelia. It is simple, flexible, and much faster than conventional frameworks because it runs on graphics cards.

Highlights

- ya||a simulates morphogenesis on GPUs much faster than conventional models
- Natively supports many mesenchymal and epithelial cellular behaviors
- Flexible and simple because it is written in concise, plain CUDA/C++
- Available and maintained with many examples at github.com/germannp/yalla

Germann et al., 2019, Cell Systems *8*, 261–266
March 27, 2019
https://doi.org/10.1016/j.cels.2019.02.007

CellPress

Cell Systems
Brief Report

ya||a: GPU-Powered Spheroid Models for Mesenchyme and Epithelium

Philipp Germann,[1,2,3,5,*] Miquel Marin-Riera,[1,2,3] and James Sharpe[1,2,3,4,*]
[1]Centre for Genomic Regulation (CRG), The Barcelona Institute of Science and Technology, 08003 Barcelona, Spain
[2]European Molecular Biology Laboratory, 08003 Barcelona, Spain
[3]Pompeu Fabra University, 08002 Barcelona, Spain
[4]Institució Catalana de Recerca i Estudis Avançats, 08010 Barcelona, Spain
[5]Lead Contact
*Correspondence: philipp.germann@embl.es (P.G.), james.sharpe@embl.es (J.S.)
https://doi.org/10.1016/j.cels.2019.02.007

SUMMARY

Simulating morphogenesis of both mesenchyme and epithelium has typically required complex and computationally expensive models. To meet this challenge, we developed ya||a—yet another parallel agent-based model. Our model extends the spheroid model by the addition of spin-like polarities to simulate epithelial sheets and tissue polarity using pairwise interactions only. This design is simple and lends itself to parallelization, and we implemented it together with recent models for protrusions and migration for GPUs for high performance. ya||a is written in concise, plain CUDA/C++ and available at github.com/germannp/yalla under the MIT license.

INTRODUCTION

Embryonic tissues come in two basic states, epithelial and mesenchymal. Epithelial cells typically create compact tissues with strong intercellular contacts, thus acting as physical barriers to other cells and molecules. Epithelial tissues are organized into sheets with stable neighborhoods and show a marked apical-basal polarity across the depth of the sheet. Such sheets undergo complex 3D deformations by means of active cell behaviors, which result in folding, bending, or twisting of the sheet, or by means of passive mechanical forces exerted by the surrounding tissues (Honda, 2017). Mesenchymal cells, on the other hand, are typically looser 3D tissues with abundant extracellular matrix. Mesenchymal cells come in various shapes, which are typically highly dynamic because of the formation and retraction of protrusions (such as filopodia and lamelopodia). Mesenchymal tissues change shape by proliferation, extracellular matrix secretion and remodeling, and intercalation (Wayne Brodland and Chen, 2000). Cell transitions from epithelial to mesenchymal type, and interactions between both tissue types, are common throughout disease and development. For example, limb bud development starts with an epithelial to mesenchymal transition (Gros and Tabin, 2014), and then outgrowth is orchestrated by a feedback loop between epithelial and mesenchymal signals (Zeller et al., 2009).

To understand how shapes emerge from such complex interactions, computational models are a useful tool to generate testable hypotheses (Sharpe, 2017). As the correct biological mechanism is often not known or very complex, the development of phenomenological models and the exploration of their parameter space require flexible and efficient simulation frameworks. However, previous simulation frameworks for morphogenesis (Hoehme and Drasdo, 2010; Richmond et al., 2010; Tapia and D'Souza, 2011; Gorochowski et al., 2012; Rudge et al., 2012; Swat et al., 2012; Mirams et al., 2013; Sütterlin et al., 2013; Gord et al., 2014; Kang et al., 2014; Starruß et al., 2014; Yu and Yang, 2014; Cytowski and Szymanska, 2015; Barton et al., 2017; Somogyi and Glazier, 2017; Sussman, 2017; Ballet, 2018; Ghaffarizadeh et al., 2018; Song et al., 2018) often emphasized either epithelial or mesenchymal processes, e.g., vertex models describe the shapes of epithelial cells within sheets or cellular Potts models describe differential adhesion (Osborne et al., 2017). Recently, solutions to overcome this limitation were put forward: an extension of the spheroid model by torsion joints (Disset et al., 2015), a 3D implementation of the vertex model (Okuda et al., 2015), a sub-cellular element model using apical and basal elements for epithelial cells (Gord et al., 2014), a sub-cellular element model using cylindrical elements for epithelial cells (Marin-Riera et al., 2016), and most recently a spheroid model with apical-basal polarity (Delile et al., 2017).

However, none of these frameworks natively support all the diverse mesenchymal and epithelial cellular behaviors, and all are computationally more complex than necessary. We therefore chose to write a simulator from scratch, dedicated for running on graphics processing units (GPUs) to take advantage of their highly efficient parallelized speed. Our simulator extends the classical spheroid model by incorporating concepts from magnetism to simulate the apical-basal polarity of epithelia and the tissue polarity seen in mesenchyme. We also added an implementation of recent methods to model contractile protrusions (Belmonte et al., 2016; Palsson and Othmer, 2000) and individual cell migration (Delile et al., 2014).

RESULTS

In a spheroid model, a cell i is described by the center of its spheroid $\vec{x}_i$. Neighboring cells interact via spherical potentials with a repelling core and an attractive zone around it (Figure 1A). Cells in contact can exert friction on each other (Okuda et al., 2015) and can exchange chemical signals (Figures 1A and 1B). To enable convergent-extension and efficient cell sorting, we added contractile cellular protrusions, which allow more distant

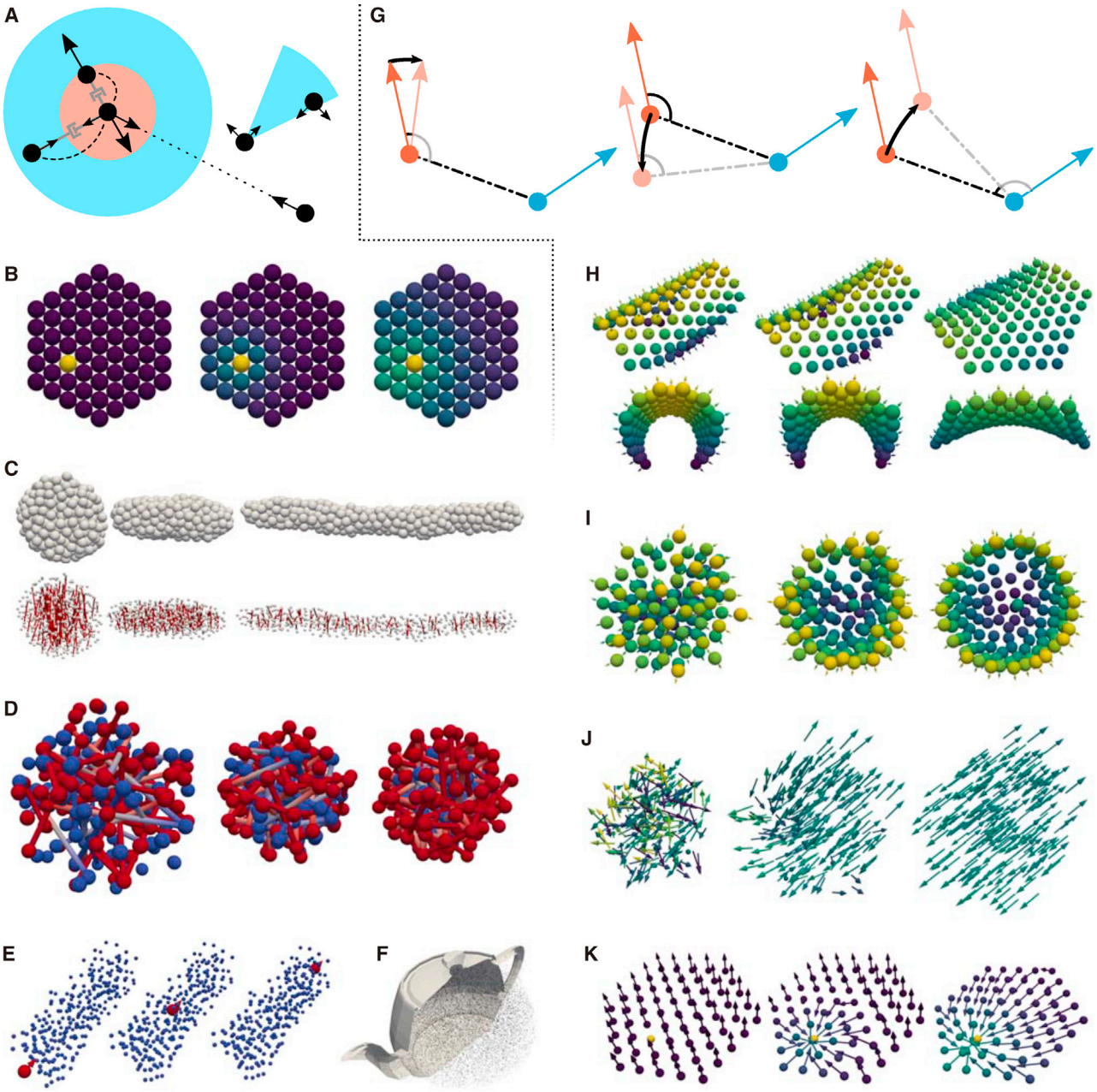


Figure 1. Spheroid Models Capture Many Cellular Behaviors

(A) Cells are described by their centers (black dots) surrounded by a repulsive core (red) and an attractive zone (blue). Cells within these ranges exert friction on each other (gray dashpots) and can exchange signals (dashed lines). More distant cells can pull toward each other by a contractile cellular protrusion (dotted line). Cells can migrate by pulling and pushing other cells (blue cone).

(B) Exchange of signals leads to distributions that represent diffusive gradients.

(C and D) (C) Contractile protrusions can drive convergent extension or (D) cell sorting.

(E) Migrating cell.

(F) Image-based modeling; only half of the teapot mesh shown.

(G) U_{Epi} at the red cell is minimized by the three depicted forces (black arrows).

(H and I) (H) Balancing these forces leads to layers with an elastic resistance to bending and (I) lets epithelial cells with initial polarities pointing radially outward self-organize on a sphere (cut shown, simulated with friction on the background).

(J and K) (J) U_{Pol} aligns mesenchymal tissue polarity and (K) U_{WNT} reorients it. The latter is shown in a plane for clarity.

cells to pull on each other (Belmonte et al., 2016; Palsson and Othmer, 2000) (dotted line in Figure 1A). These links are created and destroyed repeatedly over time, and when a field of such links is oriented in a non-random fashion, it leads to convergent-extension, as demonstrated in Belmonte et al. (2016) and in Figure 1C. Alternatively, if the contractile links are randomly oriented but preferentially link cells of the same type, then efficient cell sorting occurs (Figure 1D). Spheroid models without such membrane fluctuations often get stuck in partially sorted states (Osborne et al., 2017). Another feature we added from the literature is an implementation of individual cell migration: polarized cells can migrate by pulling toward and pushing aside cells in front (Figures 1A and 1E) (Delile et al., 2014, 2017). We also implement functionality to use meshes for image-based modeling (Figure 1F) and functionality to generate initial conditions, such as the regular hexagon in Figure 1B or the spherical distribution in Figure 1C. In sum, these extensions make the spheroid model an effective way to explore the consequences of cellular behaviors such as cell sorting in multi-cellular systems.

To this basic mesenchymal model, we add two types of cell polarity—apical-basal polarity for epithelial cells and tissue polarity for mesenchymal cells. We represent these polarities by a spin-like unit vector $\hat{p}_i$ at each cell. For epithelial cells, we introduce the potential

$$U_{\text{Epi}} = \sum_{\langle ij \rangle} (\hat{p}_i \cdot \hat{r}_{ij})^2 \Big/ 2,$$

where the sum is over all pairs of neighbors in the same epithelium. This potential is minimal when each $\hat{p}_i$ is orthogonal to all connections $\hat{r}_{ij} = (\vec{x}_i - \vec{x}_j)/|\vec{x}_i - \vec{x}_j|$ within the epithelium. The forces $\vec{F} = -\nabla U$ (Figure 1G, see Supplemental Information for a derivation of the equations of motion) minimizing such a potential let cells self-organize into layers suitable to describe epithelial sheets (Figures 1H and 1I).

Similarly, for mesenchymal cells' tissue polarity, we introduce the potential

$$U_{\text{Pol}} = -\sum_{\langle ij \rangle} (\hat{p}_i \cdot \hat{p}_j)^2 \Big/ 2,$$

where the sum is over all pairs of mesenchymal neighbors. This potential is minimal when all polarities $\hat{p}_i$ within the mesenchyme are parallel. It is therefore suitable to describe mesenchymal cells aligning because of tissue polarity (Figure 1J). Diffusing signals such as WNT are believed to act as an external influence to align tissue polarity (Yang and Mlodzik, 2015; Davey and Moens, 2017). Combining the ideas above, we can simulate such behavior. The potential

$$U_{\text{WNT}} = -\sum_{\langle ij \rangle} H(w_j - w_i) \cdot w_j \cdot (\hat{p}_i \cdot \hat{r}_{ij})^2 \Big/ 2,$$

where H is the Heaviside function and w the signals concentration, orients polarities toward cells with a higher concentration of w (Figure 1K). The mesenchymal potentials U_{Pol} and U_{WNT} only induce torques on the polarities $\hat{p}_i$ and leave the positions $\vec{x}_i$ fixed.

We implemented all these spheroid models for GPUs (see Supplemental Information for details), allowing us to simulate epithelial-mesenchymal interactions on a large scale. We demonstrate this by simulating how an epithelial Turing system induces branching (Figure 2A) and how epithelial signals shape a tissue by controlling intercalation (Figure 2B) (Menshykau et al., 2012). Similar models of actual biological processes could be used to explore robustness to noise (Figures S1A and S1B). Moreover, our implementation for GPUs operates orders of magnitude faster than conventional implementations (Figure 2C) (Marin-Riera et al., 2016; Mirams et al., 2013).

DISCUSSION

Simulation frameworks for morphogenesis can be categorized into continuum models and agent-based models (ABMs) (Tanaka, 2015). Continuum models are based on descriptions of materials and describe bulk properties such as viscosity or elasticity well at all timescales. However, it is difficult to interpret material properties such as stiffness in terms of cellular behaviors, and continuum models cannot be directly compared to cellular measurements such as dispersion (Mogilner and Manhart, 2016). Furthermore, to numerically solve continuous models, these must be discretized, usually using a mesh. Simulating growth and large deformations with meshes is challenging and computationally expensive (Chen and Brodland, 2008; Wittwer et al., 2016); however, often the bulk behavior on short timescales can be neglected. ABMs overcome the continuum model's difficulties because they model cellular behaviors directly, and agents provide a natural discretization.

To keep our simulations simple and fast, we thus extended the spheroid model to include polarized cell behaviors. Similar to Hazelwood and Hancock (2013), we simulate "tissue polarity" in 3D mesenchymal tissues, which could for instance be used to align columnar chondrocytes in long bone premordia (Yang and Mlodzik, 2015). Similar to Delile et al. (2017), we use cellular polarity for epithelia; however, our potential involves only pairwise interactions, which are easier to parallelize. Combining these novelties with previously proposed extensions for contractile protrusions (Palsson and Othmer, 2000; Belmonte et al., 2016) and individual cell migration (Delile et al., 2017) makes the spheroid model ideal for large-scale simulations of 3D morphogenesis with mesenchyme and epithelium on equal footing.

Agent-based approaches often lend themselves to parallelization—for example, the following models that were mentioned in the introduction (Richmond et al., 2010; Tapia and D'Souza, 2011; Gord et al., 2014; Yu and Yang, 2014; Cytowski and Szymanska, 2015; Harvey et al., 2015; Ballet, 2018; Ghaffarizadeh et al., 2018; Song et al., 2018). For high performance at low costs, we implemented our models into a GPU-based simulation framework. Current GPUs have thousands of cores, making them a cheap and comparably easy-to-program-and-use alternative to cluster computers. GPUs have become increasingly popular for scientific computing (Nobile et al., 2017), and their performance keeps improving. Even a cheap GPU allows us to calculate forces orders of magnitude faster than previous simulation packages (see Supplemental Information for hardware recommendations). Furthermore, outsourcing the heavy lifting to the GPU leaves the computer responsive enough for most other work during simulations.

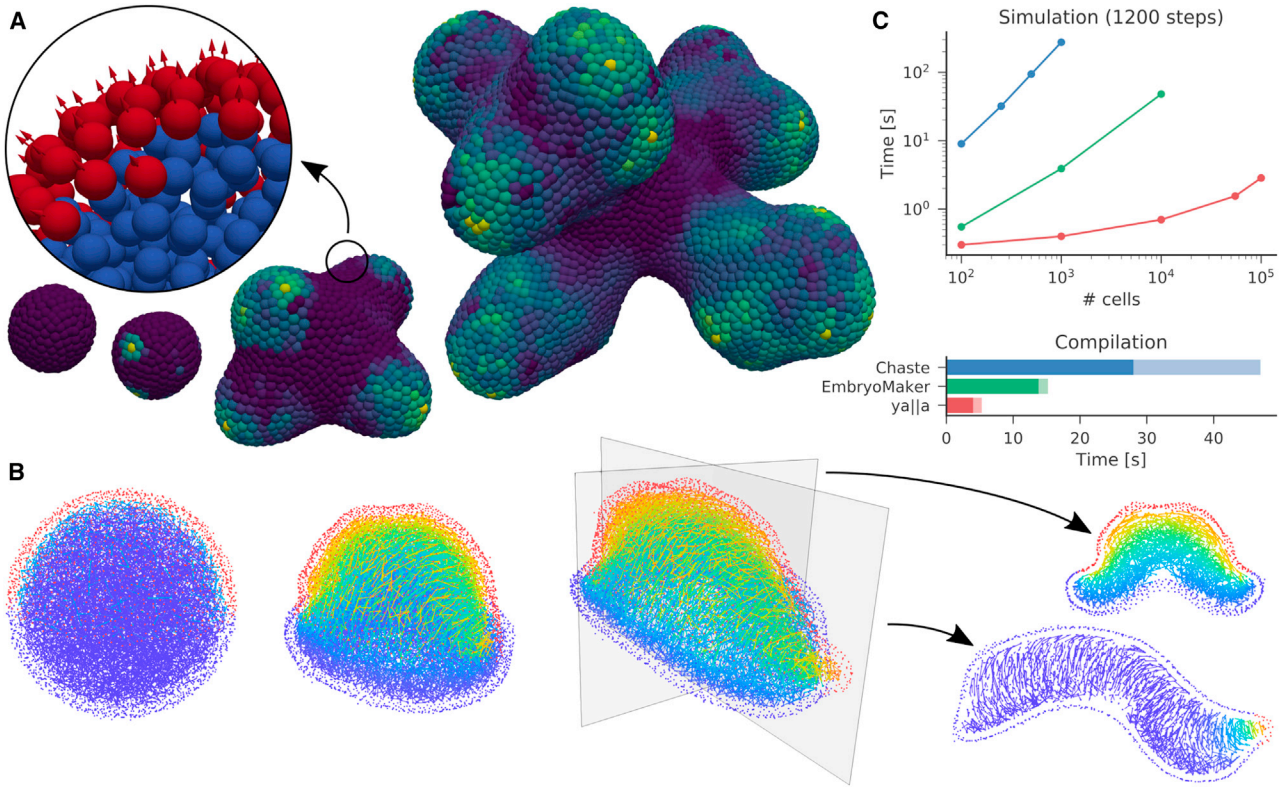


Figure 2. ya||a Is Suitable for Large-Scale Morphogenesis

(A) Branching driven by differential growth in the mesenchyme (blue) induced by Turing pattern on the epithelial surface (red). Coloring shows the concentration of the morphogen in the tissue in logarithmic scale, growing from 500 to roughly 40,000 cells in 4,000 time steps.

(B) Two epithelial signals shaping a tissue by controlling the distribution of protrusions, which induce intercalation. Cells intercalate along the gradient from top (shown in the series), except where this gradient is highest; there they intercalate normal to the second gradient from the tip (shown in the slice). The first rule folds the lower hemisphere into the upper hemisphere, and the second rule elongates the structure. Slices show the two gradients in the final configuration. Around 12,000 cells, 500 time steps.

(C) To compare computational performance, we simulate cells with a limited interaction range starting from a spherical, uncompressed, random distribution simulated until the forces were calculated 1,200 times in Chaste (Mirams et al., 2013), EmbryoMaker (Marin-Riera et al., 2016), and ya||a (see Figure S1C for the resulting simulations). ya||a scales sub-linearly for small systems because of overhead. We use an Intel i7-4770 @ 3.40GHz with an NVidia GeForce GTX 1060 6GB. There would be a parallel implementation of Chaste scaling well for up to 32 CPU cores (Harvey et al., 2015). Related to Figure S1.

We gain further performance over previous spheroid models (Mirams et al., 2013; Marin-Riera et al., 2016; Delile et al., 2017) by implementing friction among neighbors, which lets deformations propagate through large systems and hence requires fewer time steps (Okuda et al., 2015).

We deliberately kept ya||a simple. Other packages support several kinds of models (Osborne et al., 2017) or provide sophisticated user interfaces (Marin-Riera et al., 2016; Delile et al., 2017; Swat et al., 2012; Starruß et al., 2014). ya||a just works with spheroid models and relies on external programs for visualization. Thus, the numerous tests and examples, including all models used to generate the figures, can be quickly understood and easily extended because they are concise and plain CUDA/C++ (Nickolls et al., 2008). This also leads to shorter compilation times, accelerating model development. In addition, the avoidance of dependencies will make it easy to maintain ya||a in the future, and the modular design will make it easy to integrate ya||a into larger pipelines, e.g., for parameter optimization.

While we are developing ya||a specifically for limb bud morphogenesis (Hopyan et al., 2011), similar cellular behaviors drive other developmental processes such as tooth formation (Kim et al., 2017) or branching morphogenesis (Affolter et al., 2009). Recently, computational modeling is becoming increasingly popular in developmental biology (Sharpe, 2017). Furthermore, epithelial-to-mesenchymal transitions are at the core of many cancers, and cellular interactions are central to the immune system (Nagarsheth et al., 2017). The cell-centers $\vec{x}_i$ can even be reinterpreted as sub-cellular elements, and then our model for epithelia could be used to describe cellular membranes (Milde et al., 2014). We therefore believe that ya||a will be very valuable to a wide community.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- CONTACT FOR REAGENT AND RESOURCE SHARING

SUPPLEMENTAL INFORMATION

Supplemental Information can be found online with this article at https://doi.org/10.1016/j.cels.2019.02.007.

ACKNOWLEDGMENTS

We thank Alexis Naveros, Giuseppe Bilotta, and Lorenzo Pistone for helping with CUDA; Stefanie Marti, Xavier Diego, and Marco Musy for checking calculations; Xavier Diego for providing image-based meshes; Marco Musy and Antoni Matyjaszkiewicz for help with C++; Antoni Matyjaszkiewicz for tweaking the branching example; Marco Musy for careful reading of the manuscript; and James Osborne and the Chaste mailing list for helping with Chaste. This work was supported by the Spanish Ministry of Economy and Competitiveness through Centro de Excelencia Severo Ochoa 2013-2017, SEV-2012-0208, the Swiss National Science Foundation through Sinergia grants CR23I3_156234 and CRSII3_141918, and the European Research Council through SIMBIONT (project no. 670555).

AUTHOR CONTRIBUTIONS

P.G. and M.M.-R. developed ya||a under the supervision of J.S. All authors wrote the manuscript.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: April 23, 2018
Revised: August 3, 2018
Accepted: February 19, 2019
Published: March 20, 2019

REFERENCES

Affolter, M., Zeller, R., and Caussinus, E. (2009). Tissue remodelling through branching morphogenesis. Nat. Rev. Mol. Cell Biol. *10*, 831–842.

Ahrens, J., Geveci, B., and Law, C. (2005). 36-ParaView: an end-user tool for large-data visualization. In The visualization handbook, C.D. Hansen and C.R. Johnson, eds. (Elsevier Inc.), pp. 717–732.

Ballet, P. (2018). SimCells, an advanced software for multicellular modeling application to tumoral and blood vessel co-development. https://hal.archives-ouvertes.fr/hal-01853293.

Barton, D.L., Henkes, S., Weijer, C.J., and Sknepnek, R. (2017). Active Vertex Model for cell-resolution description of epithelial tissue mechanics. PLoS Comput. Biol. *13*, e1005569.

Belmonte, J.M., Swat, M.H., and Glazier, J.A. (2016). Filopodial-tension model of convergent-extension of tissues. PLoS Comp. Biol. *12*, e1004952.

Blender Online Community (2017). Blender - a 3D Modelling and Rendering Package (Blender Foundation, Blender Institute).

Wayne Brodland, G.W., and Chen, H.H. (2000). The mechanics of cell sorting and envelopment. J. Biomech. *33*, 845–851.

Chen, X., and Brodland, G.W. (2008). Multi-scale finite element modeling allows the mechanics of amphibian neurulation to be elucidated. Phys. Biol. *5*, 015003.

Cytowski, M.l., and Szymanska, Z. (2015). Large-scale parallel simulations of 3D cell colony dynamics: the cellular environment. Comput. Sci. Eng. *17*, 44–48.

Davey, C.F., and Moens, C.B. (2017). Planar cell polarity in moving cells: think globally, act locally. Development *144*, 187–200.

Delile, J., Doursat, R., and Peyriéras, N. (2014). Computational modeling and simulation of animal early embryogenesis with the MecaGen platform. In Computational Systems Biology, R. Elis and A. Kriete, eds. (Elsevier), pp. 359–405.

Delile, J., Herrmann, M., Peyriéras, N., and Doursat, R. (2017). A cell-based computational model of early embryogenesis coupling mechanical behaviour and gene regulation. Nat. Commun. *8*, 13929.

Disset, J., Cussat-Blanc, S., and Duthen, Y. (2015). MecaCell: an open-source efficient cellular physics engine. In Proceedings of the European Conference on Artificial Life 2015. 13th European Conference on Artificial Life (ECAL 2015), Jul 2015, York, United Kingdom, pp. 67.

Ghaffarizadeh, A., Heiland, R., Friedman, S.H., Mumenthaler, S.M., and Macklin, P. (2018). PhysiCell: an open source physics-based cell simulator for 3-D multicellular systems. PLoS Comput. Biol. *14*, e1005991.

Gord, A., Holmes, W.R., Dai, X., and Nie, Q. (2014). Computational modelling of epidermal stratification highlights the importance of asymmetric cell division for predictable and robust layer formation. J. R. Soc. Interface *11*, 20140631.

Gorochowski, T.E., Matyjaszkiewicz, A., Todd, T., Oak, N., Kowalska, K., Reid, S., Tsaneva-Atanasova, K.T., Savery, N.J., Grierson, C.S., and di Bernardo, M. (2012). BSim: an agent-based tool for modeling bacterial populations in systems and synthetic biology. PLoS One *7*, e42790.

Green, S. (2013). Particle Simulation Using Cuda, pp. 1–12.

Gros, J., and Tabin, C.J. (2014). Vertebrate limb bud formation is initiated by localized epithelial-to-mesenchymal transition. Science *343*, 1253–1256.

Harvey, D.G., Fletcher, A.G., Osborne, J.M., and Pitt-Francis, J. (2015). A parallel implementation of an off-lattice individual-based model of multicellular populations. Comput. Phys. Commun. *192*, 130–137.

Hazelwood, L.D., and Hancock, J.M. (2013). Functional modelling of planar cell polarity: an approach for identifying molecular function. BMC Dev. Biol. *13*, 20.

Hoehme, S., and Drasdo, D. (2010). A cell-based simulation software for multicellular systems. Bioinformatics *26*, 2641–2642.

Honda, H. (2017). The world of epithelial sheets. Dev. Growth Differ. *59*, 306–316.

Hopyan, S., Sharpe, J., and Yang, Y. (2011). Budding behaviors: growth of the limb as a model of morphogenesis. Dev. Dyn. *240*, 1054–1062.

Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. Comput. Sci. Eng. *9*, 90–95.

Kang, S., Kahan, S., McDermott, J., Flann, N., and Shmulevich, I. (2014). Biocellion: accelerating computer simulation of multicellular biological system models. Bioinformatics *30*, 3101–3108.

Kim, R., Green, J.B.A., and Klein, O.D. (2017). From snapshots to movies: understanding early tooth development in four dimensions. Dev. Dyn. *246*, 442–450.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., et al. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In Positioning and Power in Academic Publishing: Players, Agents and Agendas, F. Loizides and B. Scmidt, eds., pp. 87–90.

Mao, Y., Tournier, A.L., Hoppe, A., Kester, L., Thompson, B.J., and Tapon, N. (2013). Differential proliferation rates generate patterns of mechanical tension that orient tissue growth. EMBO J. *32*, 2790–2803.

Marin-Riera, M., Brun-Usan, M., Zimm, R., Välikangas, T., and Salazar-Ciudad, I. (2016). Computational modeling of development by epithelia, mesenchyme and their interactions: a unified model. Bioinformatics *32*, 219–225.

McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (SciPy 2010), S. van der Walt and J. Millman, eds., pp. 51–56.

Menshykau, D., Kraemer, C., and Iber, D. (2012). Branch mode selection during early lung development. PLoS Comp. Biol. *8*, e1002377.

Milde, F., Tauriello, G., Haberkern, H., and Koumoutsakos, P. (2014). SEM++: A particle model of cellular growth, signaling and migration. Comp. Part. Mech. *1*, 211–227.

Mirams, G.R., Arthurs, C.J., Bernabeu, M.O., Bordas, R., Cooper, J., Corrias, A., Davit, Y., Dunn, S.J., Fletcher, A.G., Harvey, D.G., et al. (2013). Chaste: an open source C++ Library for Computational Physiology and Biology. PLoS Comp. Biol. *9*, 1002970.

Mogilner, A., and Manhart, A. (2016). Agent-based modeling: case study in cleavage furrow models. Mol. Biol. Cell *27*, 3379–3384.

Nagarsheth, N., Wicha, M.S., and Zou, W. (2017). Chemokines in the cancer microenvironment and their relevance in cancer immunotherapy. Nat. Rev. Immunol. *17*, 559–572.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with Cuda. Queue *6*, 40–53.

Nobile, M.S., Cazzaniga, P., Tangherloni, A., and Besozzi, D. (2017). Graphics processing units in bioinformatics, computational biology and systems biology. Brief. Bioinform. *18*, 870–885.

Nyland, L., Harris, M., and Prins, J. (2007). Fast N-body simulation with Cuda. Simulation *3*, 677–696.

Okuda, S., Inoue, Y., Eiraku, M., Adachi, T., and Sasai, Y. (2015). Vertex dynamics simulations of viscosity-dependent deformation during tissue morphogenesis. Biomech. Model. Mechanobiol. *14*, 413–425.

Osborne, J.M., Fletcher, A.G., Pitt-Francis, J.M., Maini, P.K., and Gavaghan, D.J. (2017). Comparing individual-based approaches to modelling the self-organization of multicellular tissues. PLoS Comput. Biol. *13*, e1005387.

Palsson, E., and Othmer, H.G. (2000). A model for individual and collective cell movement in Dictyosteliumdiscoideum. Proc. Natl. Acad. Sci. USA *97*, 10448–10453.

Pathmanathan, P., Cooper, J., Fletcher, A., Mirams, G., Murray, P., Osborne, J., Pitt-Francis, J., Walter, A., and Chapman, S.J. (2009). A computational study of discrete mechanical tissue models. Phys. Biol. *6*, 036001.

Pokhilko, P., Epifanovsky, E., and Krylov, A.I. (2018). Double precision is not needed for many-body calculations: emergent conventional wisdom. J. Chem. Theory Comput. *14*, 4088–4096.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. (1992). Numerical Recipes in *C* Second Edition (Cambridge University Press).

Richmond, P., Walker, D., Coakley, S., and Romano, D. (2010). High performance cellular level agent-based simulation with FLAME for the GPU. Brief. Bioinform. *11*, 334–347.

Rudge, T.J., Steiner, P.J., Phillips, A., and Haseloff, J. (2012). Computational modeling of synthetic microbial biofilms. ACS Synth. Biol. *1*, 345–352.

Sharpe, J. (2017). Computer modeling in developmental biology: growing today, essential tomorrow. Development *144*, 4214–4225.

Somogyi, E., and Glazier, J.A. (2017). A modeling and simulation language for biological cells with coupled mechanical and chemical processes. arXiv, arXiv:1701.00317v5 https://arxiv.org/abs/1701.00317v5.

Song, Y., Yang, S., and Lei, J.Z. (2018). ParaCells: a GPU architecture for Cell-Centered models in computational biology. IEEE/ACM Trans. Comp. Biol. Bioinform. *5963*, 1.

Starruß, J., de Back, W., Brusch, L., and Deutsch, A. (2014). Morpheus: a user-friendly modeling environment for multiscale and multicellular systems biology. Bioinformatics *30*, 1331–1332.

Sussman, D.M. (2017). cellGPU: massively parallel simulations of dynamic vertex models. Comput. Phys. Commun. *219*, 400–406.

Sütterlin, T., Kolb, C., Dickhaus, H., Jäger, D., and Grabe, N. (2013). Bridging the scales: semantic integration of quantitative SBML in graphical multicellular models and simulations with EPISIM and COPASI. Bioinformatics *29*, 223–229.

Swat, M.H., Thomas, G.L., Belmonte, J.M., Shirinifard, A., Hmeljak, D., and Glazier, J.A. (2012). Multi-Scale Modeling of Tissues Using CompuCell3D. Methods Cell Biol. *110*, 325–366.

Tanaka, S. (2015). Simulation frameworks for morphogenetic problems. Computation *3*, 197–221.

Tapia, J.J., and D'Souza, R.M. (2011). Parallelizing the cellular Potts Model on graphics processing units. Comput. Phys. Commun. *182*, 857–865.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D.C., Augspurger, T., Halchenko, Y., Cole, J.B., Warmenhoven, J., et al. (2017). mwaskom/seaborn: v0, p. 8.1. https://doi.org/10.5281/zenodo.883859.

Wittwer, L.D., Croce, R., Aland, S., and Iber, D. (2016). Simulating organogenesis in COMSOL: phase- field based simulations of embryonic lung branching morphogenesis. In Excerpt from the Proceedings of the 2016 COMSOL Conference in Munich, https://www.comsol.ru/paper/download/441421/wittwer_paper.pdf.

Yang, Y., and Mlodzik, M. (2015). Wnt-frizzled/planar cell polarity signaling: cellular orientation by facing the wind (Wnt). Annu. Rev. Cell Dev. Biol. *31*, 623–646.

Yu, C., and Yang, B. (2014). Parallelizing the cellular Potts Model on GPU and Multi-core CPU: an OpenCL Cross-Plattform Study. In 11th International Joint Conference on Cumputer Science and Software Engineering (JCSSE), pp. 117–122.

Zeller, R., López-Ríos, J., and Zúñiga, A. (2009). Vertebrate limb bud development: moving towards integrative analysis of organogenesis. Nat. Rev. Genet. *10*, 845–858.

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Software and Algorithms		
ya||a	This paper	https://github.com/germannp/yalla
CUDA >=7	Nickolls et al. (2008)	https://developer.nvidia.com
Chaste	Mirams et al. (2013)	http://www.cs.ox.ac.uk/chaste
EmbryoMaker	Marin-Riera et al. (2016)	http://www.biocenter.helsinki.fi/salazar/software.html
ParaView	Ahrens et al. (2005)	https://www.paraview.org

CONTACT FOR REAGENT AND RESOURCE SHARING

Further information and requests for resources should be directed to and will be fulfilled by the Lead Contact, Philipp Germann (philipp.germann@embl.es).

METHOD DETAILS

Derivation of the Equations of Motion

Cells move with low inertia and high local friction, thus Newton's equations $m\ddot{\vec{x}} = \vec{F} - \gamma(\dot{\vec{x}} - \langle\dot{\vec{x}}\rangle_{(i,j)})$ become $\dot{\vec{x}} - \langle\dot{\vec{x}}\rangle_{(i,j)} \propto \vec{F}$ (Mao et al., 2013; Okuda et al., 2015). We approximate these equations of motion as $\dot{\vec{x}} = \vec{F} + \langle\dot{\vec{x}}(t - \Delta t)\rangle_{(i,j)}$ with either the center of mass or a specific cell kept fixed. We solve the resulting ODE system using Heun's method. Using a second order method allows taking approximately four times larger time steps without oscillations in the tested examples. The resulting time steps provide a suitable timescale for proliferation in our simulations, thus we do not use higher orders.

We usually use spherical forces like $\vec{F}_{ij} = [\max(0.7 - r,\ 0) \cdot 2 - \max(r - 0.8,\ 0)/2]\hat{r}_{ij}$ between cells $\vec{x}_i$ and $\vec{x}_j$ if $|\vec{x}_i - \vec{x}_j| < 1$, because they are simple to adapt. More complicated forces, like Lennard-Jones, can easily be implemented. However, they require smaller time steps and do not qualitatively change behavior over long time scales. We simulate diffusion of a signaling molecule w at each cell i as

$$\dot{w}_i = -D\sum_{(i,j)}(w_i - w_j),$$

where the sum is over all neighbors of i. For protrusions we typically use constant forces and for proliferation we simply duplicate random cells, more sophisticated models can be easily implemented.

We describe polarities $\hat{p}$ in spherical coordinates with $r = 1$, i.e.

$$\theta = \arccos p_z,$$

$$\varphi = \arctan(p_y/p_x),$$

and conversely

$$\begin{aligned} p_x &= \sin\theta\cos\varphi \\ p_y &= \sin\theta\sin\varphi \\ p_z &= \cos\theta, \end{aligned}$$

where $0 \le \theta \le \pi$ and $0 \le \varphi < 2\pi$. In these coordinates the scalar product is

$$\begin{aligned} \hat{p}_i \cdot \hat{p}_j &= \sin\theta_i\cos\varphi_i\sin\theta_j\cos\varphi_j + \sin\theta_i\sin\varphi_i\sin\theta_j\sin\varphi_j + \cos\theta_i\cos\theta_j \\ &= \sin\theta_i\sin\theta_j\cos(\varphi_i - \varphi_j) + \cos\theta_i\cos\theta_j, \end{aligned}$$

the gradient is

$$\nabla f = \frac{\partial f}{\partial\theta}\hat{\theta} + \frac{1}{\sin\theta}\frac{\partial f}{\partial\varphi}\hat{\varphi},$$

and the velocity is

$$\vec{v} = \dot{\theta}\hat{\theta} + \sin\theta\dot{\varphi}\hat{\varphi}.$$

To find the equations of motion $\vec{F} = -\nabla U$ for $U_{\mathrm{Pol}} = -\sum_{(i,j)}(\hat{p}_i \cdot \hat{p}_j)^2/2$ we therefore need to solve

$$\dot{\theta}_i \propto \sum_{(i,j)}(\hat{p}_i \cdot \hat{p}_j)\left[\cos\theta_i \sin\theta_j \cos(\varphi_i - \varphi_j) - \sin\theta_i \cos\theta_j\right]$$
$$\dot{\varphi}_i \propto -\sum_{(i,j)}(\hat{p}_i \cdot \hat{p}_j)\sin\theta_j \sin(\varphi_i - \varphi_j)/\sin\theta_i.$$

for $U_{\mathrm{WNT}} = -\sum_{(i,j)}H(w_j - w_i)\cdot w_j \cdot (\hat{p}_i \cdot \hat{r}_{ij})^2/2$ the second polarity $\hat{p}_j$ has to be replaced with $\hat{r}_{ij}$ and a strength like $H(w_j - w_i)\cdot w_j$ chosen.

Similarly, for $U_{\mathrm{Epi}} = \sum_{(i,j)}(\hat{p}_i \cdot \hat{r}_{ij})^2/2$ mainly the signs change, i.e.

$$\dot{\theta}_i \propto -\sum_{(i,j)}(\hat{p}_i \cdot \hat{r}_{ij})\left[\cos\theta_i \sin\theta_{ij} \cos(\varphi_i - \varphi_{ij}) - \sin\theta_i \cos\theta_{ij}\right]$$
$$\dot{\varphi}_i \propto \sum_{(i,j)}(\hat{p}_i \cdot \hat{r}_{ij})\sin\theta_{ij} \sin(\varphi_i - \varphi_{ij})/\sin\theta_i.$$

However, additionally contributions to $\dot{\vec{x}}$ arise, because movement affects the angle between $\hat{p}_i$ and $\hat{r}_{ij}$ too. The contribution to $\vec{F}_i$ from the angle between $\hat{p}_i$ and r_{ij} is

$$\begin{aligned}
\Delta\vec{F}_{i,ij} &= -\nabla(\hat{p}_i \cdot \hat{r}_{ij})^2/2 \\
&= -(\hat{p}_i \cdot \hat{r}_{ij})\nabla(\hat{p}_i \cdot \hat{r}_{ij}) \\
&= -\frac{\hat{p}_i \cdot \vec{r}_{ij}}{r}\nabla\frac{\hat{p}_i \cdot \vec{r}_{ij}}{r} \\
&= -\frac{\hat{p}_i \cdot \vec{r}_{ij}}{r}\left(\frac{\nabla(\hat{p}_i \cdot \vec{r}_{ij})}{r} + (\hat{p}_i \cdot \vec{r}_{ij})\nabla(1/r)\right) \\
&= -\frac{\hat{p}_i \cdot \vec{r}_{ij}}{r}\left(\frac{\nabla(\hat{p}_i \cdot (\vec{x}_i - \vec{x}_j))}{r} - \frac{\hat{p}_i \cdot \vec{r}_{ij}}{r^3}\vec{r}_{ij}\right) \\
&= -\frac{\hat{p}_i \cdot \vec{r}_{ij}}{r^2}\hat{p}_i + \frac{(\hat{p}_i \cdot \vec{r}_{ij})^2}{r^4}\vec{r}_{ij},
\end{aligned}$$

and similarly the contribution from the angle between $\hat{p}_j$ and r_{ji} is

$$\begin{aligned}
\Delta\vec{F}_{i,ji} &= -\nabla(\hat{p}_j \cdot \hat{r}_{ji})^2/2 \\
&= -\frac{\hat{p}_j \cdot \vec{r}_{ji}}{r}\left(\frac{\nabla(\hat{p}_j \cdot (\vec{x}_j - \vec{x}_i))}{r} - \frac{\hat{p}_j \cdot \vec{r}_{ji}}{r^3}\vec{r}_{ij}\right) \\
&= \frac{\hat{p}_j \cdot \vec{r}_{ji}}{r^2}\hat{p}_j + \frac{(\hat{p}_j \cdot \vec{r}_{ji})^2}{r^4}\vec{r}_{ij} \\
&= -\frac{\hat{p}_j \cdot \vec{r}_{ij}}{r^2}\hat{p}_j + \frac{(\hat{p}_j \cdot \vec{r}_{ij})^2}{r^4}\vec{r}_{ij}.
\end{aligned}$$

Implementation

The codebase of ya||a is structured in three folders, examples, include, and tests. examples contains currently sixteen examples, including the models used for Figures 1 and 2. Pieces from these examples can be mixed and matched to build new models. The folder include contains the header files with the functionality shared by the models. The first header, cudebug.cuh, contains macros for debugging CUDA code. dtypes.cuh contains the boilerplate code for a vector space over the datatype describing a cell's position and other integrated variables, like concentrations. inits.cuh provides functions to create initial shapes, like a hexagonal grid or a spherical distribution. links.cuh provides a container for the links between cells and functions to compute the forces the links exert. mesh.cuh contains functionality for image-based modeling. polarity.cuh contains functions to compute the forces for polarized cell behaviors, like the bending force in epithelia. property.cuh provides a container for cellular properties that are not integrated, like cell type. solvers.cuh implements Heun's method and is ya||a's core. In utils.cuh are pieces of code that are used in several headers and vtk.cuh contains a class for generating output. Finally, the tests folder contains automated testcases, e.g. comparing the two ways to compute the forces implemented in solvers.cuh.

ya||a takes the definition of the right hand side in three parts, Pairwise_interaction, Pairwise_friction, and Generic_forces. The former two functions are called for each pair of neighbors and have to return a contribution to $\vec{F}$ and a friction coefficient,

respectively. Pairwise_interaction typically also contains reaction-diffusion equations. The third function, Generic_forces, can be used to compute further interactions that are not necessarily among neighbors, like the forces exerted by protrusions. If boundary conditions or other constraints are required, e.g. for the reaction-diffusion equation in Figure 1B where we use a fixed source concentration in one cell, they also need to be implemented in these three functions, which define the right hand side of the equation.

Most of the computational time is spent calculating the pairwise force and friction terms. For this we implemented two methods: Tile_solver calculates the interactions between each pair (Nyland et al., 2007) and Grid_solver uses a grid to identify possibly interacting pairs (Green, 2013). Tile_solver has less overhead and is thus faster for small systems, but calculation times grow with the square of the number of spheroids, while Grid_solver scales linearly. Our implementation is for Linux and macOS in CUDA/C++ (Nickolls et al., 2008) and does not depend on additional libraries. We visualize the resulting Vtk files using ParaView (Ahrens et al., 2005).

Since the order of atomic operations is not deterministic, some of the included examples produce different results in each run, due to accumulating numerical errors. For instance in the examples with proliferation we use atomic operations to compute where to store new cells, resulting in different orders in each run and therefore different orders of interactions. The errors due to float arithmetic not being commutative then accumulate. We could avoid atomic operations for proliferation by first marking proliferating cells and then creating the new cells in a deterministic order. However, this makes the code more complicated and thus we decided to embrace variation in the examples by seeding our random number generators for each run (Figure S1), because we believe that biologically relevant models must be robust to this kind of noise. Of course this can be readily changed if required.

We only use single (float) precision in our models. Single precision has little effect on the accuracy of many-body calculations (Pokhilko et al., 2018) and single precision calculations run efficiently on the cheap consumer NVidia Geforce GPUs. Such GPUs can be installed in most computers, given enough power supply. Using various GPUs and profiling with nvvp we found that our calculations are memory bound, i.e. the execution speed scales with the memory bandwidth of the GPU.

While our early CUDA implementations were already comparably fast, they were held back by poor design. Initially, following *Numerical recipes in C* (Press et al., 1992), we used function pointers to pass the different interactions to our solvers. Using macros, and later templates, drastically improved performance. A second significant improvement was accumulating the forces in the local register, instead of writing each contribution directly into the GPU's global RAM. The calculations are usually faster than writing the output, thus we use threads to generate output, while the GPU computes several steps.

We used Blender (Blender Online Community, 2017) to create a closed mesh for the teapot in Figure 1F.

Limitations

We have built ya||a for limb bud morphogenesis and we have not studied unrelated morphogenetic processes, like apical constriction or planar cell polarity. However, ya||a is very flexible and developing corresponding models would be straight forward.

Similarly, the solvers we require are based on the overlapping spheres model, i.e. all cells within a certain radius are potentially interacting neighbors. Such a model is not ideal for processes with cells deforming drastically before changing neighbors (Pathmanathan et al., 2009; Osborne et al., 2017). While alternative interaction models can be implemented on top of the provided solvers, the required code might be more complex and the resulting simulations might be slower than dedicated solvers.

Running ya||a requires a graphics card from NVidia, since it is implemented for NVidia's parallel computing platform CUDA. CUDA supports many generic programming features and includes valuable resources like a visual profiler, making CUDA easier to program than more portable platforms. This platform choice affects mainly laptops as other computers can be upgraded at low cost.

QUANTIFICATION AND STATISTICAL ANALYSIS

We used Python packages for scientific computing (Waskom et al., 2017; McKinney, 2010; Hunter, 2007; Kluyver et al., 2016) to create the plot in Figure 2C.

DATA AND SOFTWARE AVAILABILITY

ya||a is available as Data S1 mmc2.zip and at github.com/germannp/yalla under the MIT license.