PLOS COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

Tissue Forge: Interactive biological and biophysics simulation environment

T. J. Sego[1]*, **James P. Sluka**[2], **Herbert M. Sauro**[3], **James A. Glazier**[2]

1 Department of Medicine, University of Florida, Gainesville, Florida, United States of America, **2** Department of Intelligent Systems Engineering and Biocomplexity Institute, Indiana University, Bloomington, Indiana, United States of America, **3** Department of Bioengineering, University of Washington, Seattle, Washington, United States of America

* timothy.sego@medicine.ufl.edu

Check for updates

OPEN ACCESS

Citation: Sego TJ, Sluka JP, Sauro HM, Glazier JA (2023) Tissue Forge: Interactive biological and biophysics simulation environment. PLoS Comput Biol 19(10): e1010768. https://doi.org/10.1371/journal.pcbi.1010768

Editor: Melissa L. Kemp, Georgia Institute of Technology and Emory University, UNITED STATES

Received: November 28, 2022

Accepted: September 25, 2023

Published: October 23, 2023

Copyright: © 2023 Sego et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: The code for this project is available on GitHub (https://github.com/tissue-forge/tissue-forge). The code for all reported benchmarks is available in S3 File. Simulation code is available in the S2, S4, S5, S6 and S7 Files.

Funding: Funding for Tissue Forge is provided by NIBIB U24 EB028887 (HMS, JAG, TJS, JPS). TJS and JAG acknowledge funding from grants NSF 2120200, NSF 2000281, NSF 1720625, NIH R01 GM122424. JPS acknowledges additional funding from the EPA STAR RD840027 and NSF 2054061.

Abstract

Tissue Forge is an open-source interactive environment for particle-based physics, chemistry and biology modeling and simulation. Tissue Forge allows users to create, simulate and explore models and virtual experiments based on soft condensed matter physics at multiple scales, from the molecular to the multicellular, using a simple, consistent interface. While Tissue Forge is designed to simplify solving problems in complex subcellular, cellular and tissue biophysics, it supports applications ranging from classic molecular dynamics to agent-based multicellular systems with dynamic populations. Tissue Forge users can build and interact with models and simulations in real-time and change simulation details during execution, or execute simulations off-screen and/or remotely in high-performance computing environments. Tissue Forge provides a growing library of built-in model components along with support for user-specified models during the development and application of custom, agent-based models. Tissue Forge includes an extensive Python API for model and simulation specification via Python scripts, an IPython console and a Jupyter Notebook, as well as C and C++ APIs for integrated applications with other software tools. Tissue Forge supports installations on 64-bit Windows, Linux and MacOS systems and is available for local installation via conda.

Author summary

Tissue Forge is open-source software for particle-based modeling and simulation in physics, chemistry and biology problems. Tissue Forge users can build simulations using built-in model components and user-defined models, and execute their simulations interactively with real-time rendering or in high-performance computing environments. Simulations can dynamically create, modify and destroy particles during simulation through scripted or interactive commands, and can target a wide range of scales, from the molecular to the multicellular, using built-in features that support modeling atoms, molecules, cells, and solid and fluid materials. Tissue Forge allows users to inject procedural code into a simulation as user-specified functions, which supports custom simulation events and complex agent-based models. Tissue Forge provides user interfaces in the C and C++

This research was supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: No competing interests.

programming languages for building software applications, and in the Python programming language with integrated support for interactive execution in IPython and Jupyter Notebooks. Tissue Forge is publicly available, provides documentation on project philosophy, core concepts, working examples, and all language interfaces, and is maintained and distributed through an automated and transparent software development cycle.

Introduction

Computational modeling and simulation are key components of modern biological research. Simulations codify knowledge into computable representations that can challenge and validate our understanding of complex biological processes. A well defined model not only explains currently available data but also predicts the outcomes of future experiments. Biological computer simulations can address a wide range of length scales and employ numerous numerical and simulation technologies. Scales include that of the atomic bond length to model small molecules, proteins and other biological macromolecules, the macromolecular scale to model protein aggregates, the subcellular and cellular scales to model cells and aggregates of cells, the tissue scale to model long-range interaction between cell aggregates that give rise to organ-level behaviors, the whole-body scale where organs interact, and the population scale where individuals interact with each other and their environment. At various biological scales, models can represent biological objects as either discrete or as numerically aggregated populations, and so different mathematical and computational approaches are used to simulate behaviors at each scale. When spatiality is explicitly modeled, molecular dynamics (**MD**) simulations are often used at the atomic and macromolecular scales and spatial agent-based models are used at the higher scales. Often, discrete biological objects (*molecules, cells, cell aggregates*) are appropriately modeled as discrete objects at a particular scale, and then as numerically aggregated populations at higher scales using continuous dynamics like ordinary differential equations (**ODEs**) and partial differential equations (**PDEs**), which then describe the dynamics of a population of objects. For example, modeling at the multicellular scale can represent molecules of a given chemical species as densities or amounts, and at the molecular level as discrete molecules. While population models can have significant explanatory value, biology is intrinsically spatial. Emergent biological properties and behaviors arise in part because of the spatial relationships of their components. Population models sacrifice this aspect of biological organization.

In the subcellular, cellular and multicellular modeling domain, most spatiotemporal agent-based biological simulation tools only support one cellular dynamics simulation methodology, and focus on a particular problem domain with a particular length scale. For example, CompuCell3D (**CC3D**) [1] and Morpheus [2] implement cell model objects using the Cellular Potts model (**CPM**)/ Glazier-Graner-Hogeweg (**GGH**) formalism [3], and only support Eulerian, lattice-based models, while others like PhysiCell [4] and CHASTE [5] support modeling cells with Lagrangian, lattice-free, particle-based center models as simple, point-like cell particles. Lattice-free, particle-based methods can be extended to include subcellular detail using the Subcellular Element Model [6], which could support modeling the spatial complexity of cell shape, cytoskeleton and extracellular matrix. Extending the CPM/GGH to include cellular compartments [7] allows representation of subcellular components like the nucleus, critical molecular species or regions with specific properties but does not support specific representation of macromolecular machinery. Typically, modelers who are interested in subcellular and cellular detail must use and adapt general-purpose MD simulation tools like LAMMPS [8],

HOOMD-blue [9], NAMD [10] or GROMACS [11]. For example, Shafiee et al., customized LAMMPS to model cells as clusters of particles to simulate spheroid fusion during spheroid-dependent bioprinting [12].

Most MD simulation tools are designed to parse and execute models that are theoretically well defined and MD simulation specifications and engines tend to be well optimized for computational performance. Most assume a fixed numbers of objects within a model and do not support runtime object creation, destruction or modification. Many do not support real-time simulation visualization and user interactivity. In addition, extending these modeling environments with custom modeling and simulation features requires software development in C or C++ code. Results can be post-processed after execution, though this requires developing a pipeline of model development, simulation execution and data generation using a simulation tool, and data visualization and analysis using different visualization tools (*e.g.*, The Visualization Toolkit [13]) or a general purpose programming language like Python, which significantly increases user effort to produce useful results. To reduce user effort required to produce publishable simulation results and analysis, some simulation tools provide real-time simulation visualization and limited simulation interaction (*e.g.*, CC3D and Morpheus). Cell simulation tools with real-time visualization are often implemented as stand-alone programs, rather than as portable libraries that support integration with other modeling environments. This lack of software interoperability also complicates using simulation tools with other specialized software libraries (*e.g.*, optimization tools) in advanced computational workflows for solving difficult biological problems such as reverse-engineering model parameters, interrogation of mechanisms, or Bayesian modeling of populations.

This paper presents Tissue Forge, an open-source, real-time, modeling and simulation environment for interactive biological and biophysics modeling applications over a broad range of scales. Tissue Forge is designed to address many of the aforementioned issues and challenges. Tissue Forge enables agent-based, spatiotemporal computational modeling at scales from the molecular to the multicellular. It is designed for ease of use by modelers, research groups and collaborative scientific communities with expertise ranging from entry- to advanced-level programming proficiency. It supports all stages of model-supported research, from initial model development and validation to large-scale virtual experiments. Here we describe the philosophy, mathematical formalism and basic features of Tissue Forge. To demonstrate its usefulness across multiple disciplines in the physical and life sciences, we also present representative examples of advanced features at a variety of target scales.

Materials and methods

Tissue Forge seeks address some of the limitations of current modeling packages by providing a spatiotemporal modeling and simulation environment that supports multiple lattice-free, particle-based methods for agent-based modeling. It simplifies research by supporting representation of a wide range of scales encountered in biophysics, chemistry and biological applications. Tissue Forge supports the development, testing and deployment of models in large-scale, high-performance simulation, performed by users with a wide range of expertise and coding proficiency in multiple programming languages.

Problem domain

Simulation of complex systems, particularly in biological problems, is difficult for a number of reasons. Difficulties exist for both the domain knowledgeable modeler and the modeling tool developer. Problems in cell biology and biophysics applications often require representations of objects and processes at multiple scales, which resolve to spatiotemporal, agent-based

models with complex rules and decision making using embedded models of internal agent state dynamics (*e.g.*, chemical networks). Since such models are experimentally or empirically determined and highly diverse, their implementation requires flexible, robust model and simulation specification. Likewise, the spatial scale itself presents the challenge of choosing an appropriate mathematical framework for creating model objects and processes (*e.g.*, whether to model a cell with complex shape or simply as a sphere). Often, the modeler must learn a new software tool for each spatial scale they wish to model. In addition, the model features and computational performance of a particular software tool can be limited by the underlying mathematical framework, unpermissive or demanding object definitions, or the need for efficient use of computing resources.

Tissue Forge addresses these issues by providing an agent-based, spatiotemporal modeling and simulation framework built on a flexible, particle-based formalism. Particles, which are the fundamental agents of any Tissue Forge simulation, are suitable basic objects in model construction because they minimally constrain a model description. A Tissue Forge particle is an instance of a categorical descriptor called a "particle type," and is a discrete agent that has a unique identity, occupies a position at each moment in time and has velocity and mass or drag. Tissue Forge imposes no further restrictions on what physical or abstract object a particle represents. This framework has the theoretical and computational flexibility to enable agent-based, spatiotemporal computational models across a broad range of scales. An instance of a particle could represent an atom, or a cell, or a multicellular aggregate. Tissue Forge accommodates models with both pre- and user-defined particle behaviors and interactions, the creation and deletion of particles at runtime, and consistent object modeling at multiple scales.

Interactive and Batch Execution. Tissue Forge supports the efficient development agent-based models of complex systems. In general, the development of a computational model involving multiple interacting agents requires iterative cycles of model development, execution, analysis, and refinement. During model exploration, refinement and validation, modelers can benefit from a simulation environment that allow them to observe, interact with, and refine a simulation as it executes (*i.e.*, real-time simulation and visualization). However, computationally intensive investigations of developed models (*e.g.*, characterizing emergent mechanisms or the effects of system stochasticity, systems with large numbers of objects) require efficient high-performance computing utilization and batch execution. Tissue Forge supports both interactive and batch operation, providing both rapid and intuitive model development and high-performance simulation execution, so that modelers do not need to find and learn multiple software tools or settle for a tool that is either, but not both, feature rich or computationally efficient. Its interactive simulation mode is a stand-alone application with real-time visualization and user-specified events. Its batch mode leverages available resources in high-performance computing environments such as computing clusters, supercomputers, and cloud-based computing, and exports simulation data and high-resolution images. In batch mode, Tissue Forge can be included in workflows to carry out modeling task such as model fitting or simulation of replicates and populations.

Open Science Support. Development and dissemination of models that leverage interdisciplinary knowledge and previous modeling projects require robust support for scientific communication, collaboration, training and reuse. Tissue Forge provides a declarative model specification for many basic aspects of particle-based models and simulations (*e.g.*, particle type definitions, particle interactions and stochastic motion via generalized force and potential definitions) with robust support for procedural specification of complex, agent-based models particular to specific applications. Tissue Forge also supports model sharing and collaborative development by providing built-in support for exporting and importing simulations and model object states to and from human-readable string data (using JSON format). In support

of collaborative, community-driven and application-specific development of models, the Tissue Forge code base provides a designated space in which developers can implement features in customized Tissue Forge builds. Extending the Tissue Forge API with custom interfaces requires minimal effort in all supported software languages. Developers are also welcome to submit their custom features to the public Tissue Forge code repository for future public release as built-in features, or to design their software applications using Tissue Forge as an external software library. Along with executing scripted simulations specified in C, C++ and Python programming languages, Tissue Forge also supports collaboration, training and scientific communication through its Python API support for interactive simulations in Jupyter Notebooks. Tissue Forge simplifies robust model construction and simulation development through expressive model specification (*e.g.*, process arithmetic), a flexible event system for implementing model-specific rules (*e.g.*, agent rules) and simulation-specific runtime routines (*e.g*, importing and exporting data), and a simple, intuitive simulation control interface (*e.g.*, switching between interactive and off-screen execution).

Concepts

Tissue Forge updates the trajectory of a particle in time by calculating the net force acting on the particle. Forces determine the trajectory of a particle according to the dynamics of the particle type. Tissue Forge currently supports Newtonian and Langevin (overdamped) dynamics, which can be individually specified for each particle type of a simulation.

For Newtonian dynamics, the position $\boldsymbol{r}_i$ of the ith particle is updated according to its acceleration, which is proportional to its mass m_i and the total force $\boldsymbol{f}_i$ exerted on it,

$$\boldsymbol{f}_i = m_i \frac{d^2 \boldsymbol{r}_i}{dt^2}, \tag{1}$$

and for Langevin (overdamped) dynamics, m_i is the drag coefficient and the particle velocity is proportional to the total force,

$$\boldsymbol{f}_i = m_i \frac{d\boldsymbol{r}_i}{dt}. \tag{2}$$

Tissue Forge supports three broad classes of force-generating interaction,

$$\boldsymbol{f}_i = \sum_{j \neq i} \left(\boldsymbol{F}_{ij}^{impl} + \boldsymbol{F}_{ij}^{bond} \right) + \boldsymbol{F}_i^{expl}. \tag{3}$$

$\boldsymbol{F}_{ij}^{impl}$ is the force due to *implicit* interactions between the ith and jth particles, $\boldsymbol{F}_{ij}^{bond}$ is the force due to *bonded* interactions between the ith and jth particles, and $\boldsymbol{F}_i^{expl}$ is the explicit force acting on the ith particle. Implicit interactions result automatically from interaction potentials between pairs of particles of given types. Bonded interactions act between specific pairs of individual particles (Fig 1A). Explicit forces act on particles through explicitly-defined force descriptions and do not necessarily represent inter-particle interactions (*e.g.*, gravity, internal noise, system thermal equilibrium). Tissue Forge provides built-in force- and potential-based definitions, supports user-specified definitions for both, and permits applying an unlimited number of executable Tissue Forge force and potential objects to individual particles and particle types.

Implicit interactions are defined in Tissue Forge using potential functions and applied according to the types of two interacting particles. The force between the ith and jth interacting particles resulting from their implicit interactions is calculated as the sum of each kth potential

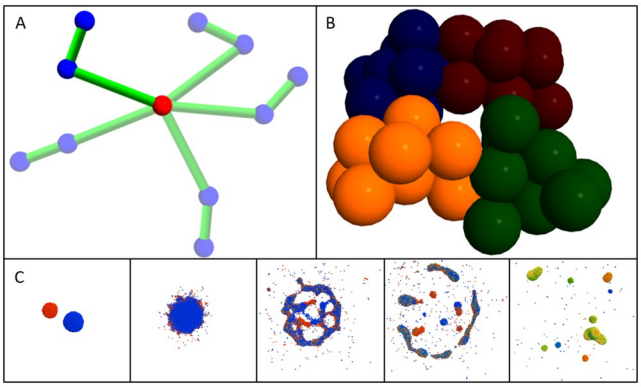

Fig 1. Examples of Tissue Forge modeling features. A: Five superimposed snapshots of a double pendulum implemented in Tissue Forge. Bonded interactions (represented as green cylinders) explicitly describe the interaction between a particular pair of particles, while a constant force acts on the blue particles in the downward direction. The red particle is fixed. B: Four Tissue Forge clusters representing biological cells, each consisting of ten particles whose color demonstrates cluster membership. Potentials describe particle interactions by whether they are in the same cluster (*i.e.*, intracellular) or different clusters *i.e.*, intercellular. C: Tissue Forge simulation of chemical flux during fluid droplet collision. Each particle represents a portion of fluid that carries an amount of a diffusive chemical, the amount of which varies from zero (blue) to one (red). When two droplets carrying different initial chemical amounts collide, resulting droplets tend towards homogeneous chemical distributions.

https://doi.org/10.1371/journal.pcbi.1010768.g001

U_{ijk}^{impl} that defines the implicit interaction,

$$\boldsymbol{F}_{ij}^{impl} = -\frac{\partial}{\partial \boldsymbol{r}_i}\sum_k U_{ijk}^{impl}. \tag{4}$$

Bonded interactions are defined in Tissue Forge using potential functions and are applied according to the identities of two interacting particles. The force between the *i*th and *j*th interacting particles resulting from their bonded interactions is calculated as the sum of each *k*th potential U_{ijk}^{bond} that defines the bonded interaction,

$$\boldsymbol{F}_{ij}^{bond} = -\frac{\partial}{\partial \boldsymbol{r}_i}\sum_k U_{ijk}^{bond}. \tag{5}$$

Explicit forces can be defined on the basis of particle type or on individual particles. The force on the *i*th particles resulting from external forces is calculated as the sum of each *k*th explicit force $\boldsymbol{F}_{ik}^{expl}$,

$$\boldsymbol{F}_{i}^{expl} = \sum_k \boldsymbol{F}_{ik}^{expl}. \tag{6}$$

Since Tissue Forge enables the implementation and execution of models at different length scales, particles in a simulation may represents objects with a wide variety of possible behaviors. A particle could be atomic and subject to energy-conserving, implicit interactions (*e.g.*, Coulomb, Morse or Lennard-Jones potentials) as in classic MD. Particles can also represent portions of material that constitute larger objects (*e.g.*, a portion of cytoplasm) and can carry quantities of materials within them (*e.g.*, convection of a solute chemical in a portion of a fluid, Fig 1C). Tissue Forge provides built-in features to enable particle-based modeling and simulation of fluid flow based on transport dissipative particle dynamics (*tDPD*) and smooth particle hydrodynamics, including a predefined tDPD potential U_{ij}^{tDPD} that can be applied when

describing the interactions of a simulation,

$$-\frac{\partial U_{ij}^{tDPD}}{\partial \boldsymbol{r}_i} = \boldsymbol{F}_{ij}^{C} + \boldsymbol{F}_{ij}^{D} + \boldsymbol{F}_{ij}^{R}, \tag{7}$$

where the interaction between the ith and jth fluid-like particles is a sum of a conservative force $\boldsymbol{F}_{ij}^{C}$, a dissipative force $\boldsymbol{F}_{ij}^{D}$ and a random force $\boldsymbol{F}_{ij}^{R}$ acting on the ith particle.

To support treating particles as constituents of larger objects, Tissue Forge provides a special type of particle, a *cluster*, whose elements can consist of constituent particles or other clusters. Clusters provide a convenient way to define implicit interactions that only occur between particles within the same cluster (*e.g.*, intracellular interactions), called *bound* interactions, and those that only occur between particles from different clusters (*e.g.*, intercellular interactions), called *unbound* interactions (Fig 1B).

To allow particles to carry embedded quantities, Tissue Forge supports attaching to each particle a vector of states that can evolve during a simulation. The values of the states can evolve according to laws defined between pairs of particle types for inter-particle transport (*e.g.*, diffusion), which Tissue Forge automatically applies during simulation, or according to local, intra-particle reactions. The time evolution of a state vector $\boldsymbol{C}_i$ attached to the ith particle is,

$$\frac{d\boldsymbol{C}_i}{dt} = \boldsymbol{Q}_i = \sum_{j \neq i} \boldsymbol{Q}_{ij}^{T} + \boldsymbol{Q}_i^{R}, \tag{8}$$

where the rate of change of the state vector attached to the ith particle is equal to the sum of the transport fluxes $\boldsymbol{Q}_{ij}^{T}$ between the ith and each nearby jth particle and the local reactions $\boldsymbol{Q}_i^{R}$. Among other models related to secretion and uptake, Tissue Forge provides a built-in transport flux model for Fickian diffusion. For the ith and jth particles with flux constant k_{ij} separated by distance r_{ij}, the Fickian diffusion flux $Q_{ij}^{T,diff}$ describing the rate of transport for species C_i is,

$$Q_{ij}^{T,diff} = \begin{cases} k_{ij}(C_j - C_i)\left(1 - \frac{r_{ij}}{r_{ij}^{cutoff}}\right)^2 & r_{ij} \leq r_{ij}^{cutoff} \\ 0 & r_{ij} > r_{ij}^{cutoff} \end{cases}. \tag{9}$$

Here r_{ij}^{cutoff} is the cutoff distance of the flux. A flux can be applied by species and pair of particle types, and each flux definition can prescribe its own cutoff distance and model parameter (s) (*e.g.*, flux constant). Note that, when particles are arranged in a regular grid, particle-based diffusion performs the same computations as those from solving the diffusion equation using the finite difference method with first-order central difference discretization of space. In such cases, the flux constant k_{ij} can be written in terms of a diffusion coefficient D,

$$k_{ij} = \frac{D}{r_{ij}^2\left(1 - \frac{r_{ij}}{r_{ij}^{cutoff}}\right)^2}. \tag{10}$$

Tissue Forge also supports integrating species transport in sub-intervals of time for a simulation step to handle numerical instabilities associated with fast diffusion, as available in CC3D and PhysiCell.

Basic features

Tissue Forge supports model and simulation specification using classes, objects and functions typical to object-oriented concepts in C, C++ and Python programming languages. In Python, custom Tissue Forge particle types can be defined by creating Python classes and specifying class attributes (Listing 1).

Listing 1. Importing the Tissue Forge library and declaring a particle type in Python. Comments are shown in green.

```
# Get the Tissue Forge Python library
import tissue_forge as tf
# Specify a particle type with a particular radius
class OscType(tf.ParticleTypeSpec):
    radius = 0.5
```

Tissue Forge allows specification of particle types without an initialized Tissue Forge runtime. However, initializing the Tissue Forge runtime, which in Python only requires a call to a single module-level function, permits retrieving template executable particle types that can be used to create particles (Listing 2). When a particle of a particular particle type is created, the particle inherits all attributes of its type (*e.g.*, mass), which can in turn be modified for the particular particle at any time during simulation. Initializing the Tissue Forge runtime requires no user-specified information, in which case a default configuration is provided, but explicit initialization provides a number of customization options to tailor a simulation to a particular problem (*e.g.*, domain size, interaction cutoff distance).

Listing 2. Initializing a Tissue Forge simulation, retrieving an executable particle type and creating particles in Python.

```
# Initialize with a 10x10x10 domain and cutoff distance of 3
tf.init(dim=[10, 10, 10], cutoff=3)
# Get the oscillator type and create two particles
osc_type = OscType.get()  # a particle type
osc_part1 = osc_type([4, 5, 5]) # particle 1: x,y,z coords
osc_part2 = osc_type([6, 5, 5]) # particle 2: x,y,z coords
# Change the radius of one of the particles
osc_part2.radius = 0.25
```

Users specify and apply interactions, whether using built-in or custom potential functions or explicit forces, by creating Tissue Forge objects that represent processes (*e.g.*, a force object), called *process objects*, and applying them categorically by predefined ways that processes can act on objects (*e.g.*, by type pairs for implicit interactions). Tissue Forge calls applying a process to model objects *binding*, which Tissue Forge applies automatically during simulation execution according to the model objects and process. For example, users can specify an implicit interaction between particles to two types by creating a potential object and binding it to the two particle types (Listing 3).

Listing 3. Creating a Tissue Forge potential and binding it to particles by type in Python.

```
# Create a harmonic potential object
pot = tf.Potential.harmonic(k=1, r0=1.5)
# Bind the harmonic potential to pairs of
#   particles of the oscillator type
tf.bind.types(pot, osc_type, osc_type)
```

Tissue Forge provides fine-grained simulation control, where each integration step can be explicitly executed, with other user-defined tasks accomplished between executing simulation steps (*e.g.*, exporting simulation data). For interactive execution, Tissue Forge simulations are

usually executed using a basic `run` function, which executes an event loop that (1) integrates the universe, (2) processes user input (*e.g.*, keyboard commands), (3) updates simulation visualization, and (4) executes an event system with user-defined events. The Tissue Forge event system allows users to insert instructions into the event loop via user-defined functions (Listing 4). Events can be executed at arbitrary frequencies, can automatically retrieve simulation data (*e.g.*, a randomly selected particle of a specific type), and can change qualities of individual particles (*e.g.*, change the radius of a particular particle based on its environment).

Listing 4. Creating a Tissue Forge event and running an interactive simulation in Python.

```
# Define an event that prints the time and particle x-coordinate
def my_event(e: tf.event.TimeEvent):
    print('Time:', tf.Universe.time)
    print('p1 x position:', osc_part1.position.x())
    print('p2 x position:', osc_part2.position.x())
# Register the event for execution at every simulation step
tf.event.on_time(period=tf.Universe.dt, invoke_method=my_event)
# Run the simulation
tf.run()
```

During simulation execution, including during execution of user-defined events, Tissue Forge objects are available for accessing and manipulating simulation, universe and system information. The Python code described in this section generates the Tissue Forge simulation depicted in Fig 2 (see S2 File), and also prints the current simulation time and x-coordinate of

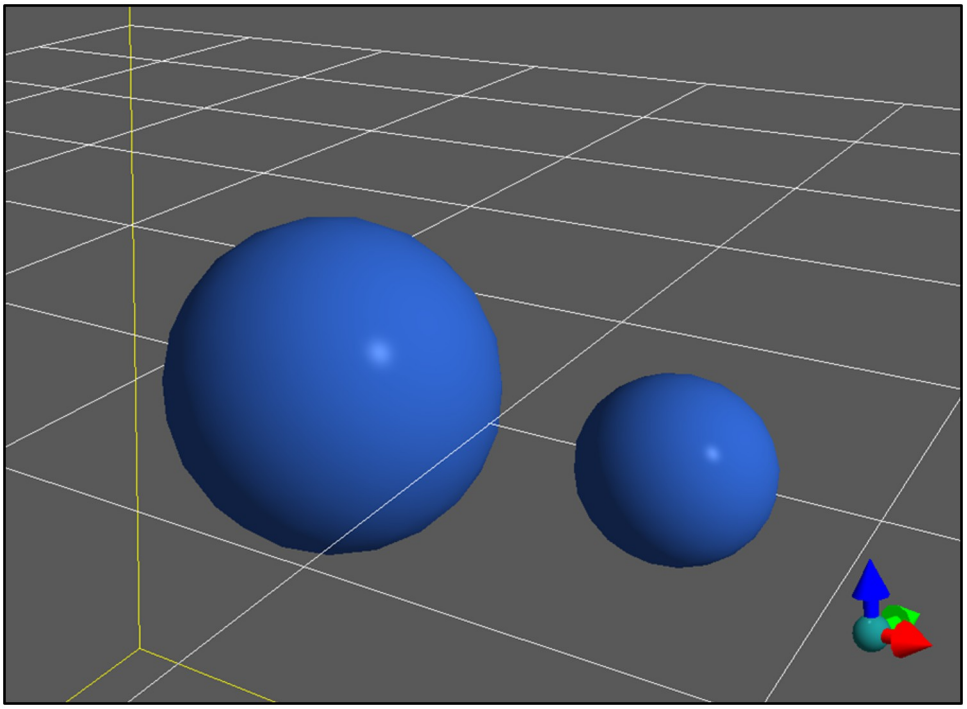

Fig 2. Tissue Forge simulation of a simple oscillator with two particles interacting via a harmonic potential. Tissue Forge helps to orient the user by drawing a yellow box around the simulation domain, a white grid along the xy plane at the center of the domain, and an orientation glyph at the bottom right to demonstrate the axes of the simulation domain with reference to the camera view, where red points in the x direction, green in the y direction and blue in the z direction.

https://doi.org/10.1371/journal.pcbi.1010768.g002

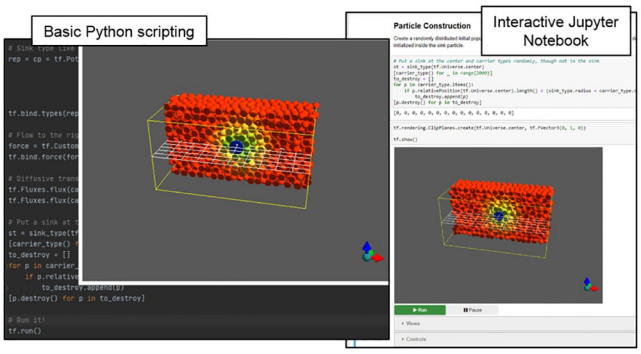

Fig 3. Tissue Forge Python deployment examples. Sample use of the Python API to specify an interactive simulation of convection of a species near a species sink in a Python script (left) and in an interactive Jupyter Notebook (right).

https://doi.org/10.1371/journal.pcbi.1010768.g003

both particles at every simulation step. This simulation can be executed as a Python script or in an IPython console.

In a Jupyter Notebook, this code executes the same simulation but generates an additional user interface, which provides widgets for interactive simulation controls, *e.g.*, for pausing and resuming the simulation, and choosing predefined camera views (Fig 3). When running Tissue Forge from a Python script or IPython console, the interface supports mouse control (*e.g.*, click and drag to rotate) and predefined and user-defined keyboard commands (*e.g.*, space bar to pause or resume the simulation). In interactive contexts like IPython and Jupyter Notebooks, the Tissue Forge event loop recognizes user commands issued *ad hoc* during simulation, allowing on-the-fly modification of the simulation state, which is especially useful during model development and interrogation (*e.g.*, when testing the effects of the timing of an event).

Implementation details

Tissue Forge treats space as a regular grid of connected subdomains, called *cells*. Data for each particle (*e.g.*, particle position and velocity) is stored in contiguous memory by the cell that contains the particle, which localizes inter-particle interactions in memory and establishes strong task parallelism by cell. As such, when the position of a particle changes from one cell to another, the data of the particle is also moved to the storage of the other cell (Fig 4). For each simulation step, Tissue Forge performs three stages of procedures to update the simulation state. In the first stage, called *Prep*, cached data from previous steps (*e.g.*, total force on a particle, total system energy) are reset. In the second stage, called *Force*, forces on particles and fluxes between them are calculated. In the third stage, called *Update*, particle states are updated, total system energy is calculated, and all registered events are executed.

Task-based parallelism is applied during the Force stage, where three types of tasks are performed for each cell. The first task, called *Sort*, builds ordered lists of particle indices according to proximity to neighboring cells. The Sort task is accomplished once for each cell. The second task, called *Force Pair*, calculates the interactions of particles in one cell with the particles of another cell. The Force Pair task is accomplished once for each pair of cells with interacting particles. The Force Pair task is not performed for a pair of cells until the Sort task has been performed for both cells. The third task, called *Force Self*, calculates the interactions of particles in the same cell. The Force Self task is accomplished once for each cell. Tasks are assembled into a queue and dedicated threads, called *Runners*, pull available tasks from the queue and perform them as they become available. Various routines in other stages, such as rendering,

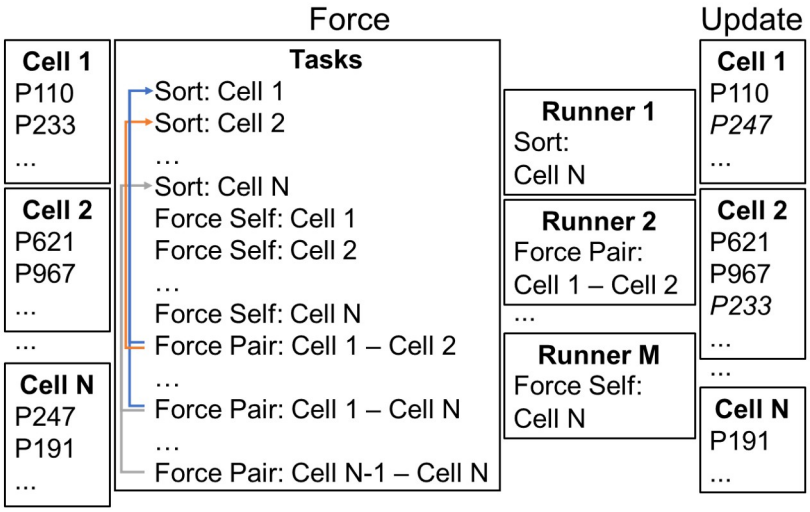

Fig 4. Conceptual diagram of space discretization and task-based parallelism. Space is discretized into subdomains called "cells" (shown here for N subdomains), and particles (listed as "PX") are stored in memory by which cell contains them. For each simulation step, a set of threads, called "runners" (shown here for M runners) perform a pre-defined set of tasks. The "Sort" task builds ordered lists of particle indices according to proximity to neighboring cells for efficient pruning of inter-cell interactions between particle pairs outside of the cutoff distance. The "Force Self" task calculates interactions between particles of the same cell. The "Force Pair" task calculates interactions between particles of different cells using results from the Sort task. Task scheduling enforces task dependency. After all forces are calculated, particle positions are updated and particles that move into a neighboring cell are appropriately moved in memory, as demonstrated for particles P247 (from cell N to cell 1) and P233 (from cell 1 to cell 2).

https://doi.org/10.1371/journal.pcbi.1010768.g004

data resetting and particle state updating, are trivially parallelized by maintaining an array of pointers to particle data memory. Particle trajectories and state vectors are updated using explicit first-order forward time integration. Architectural features provide support for implementing additional time integration schemes and real-time numerical stability analysis in future development.

While computational cost significantly varies based on simulation details like cutoff distance and number of particles (increasing values of which increases cost), the computational cost of a Tissue Forge simulation scales well with increasing particle number (Fig 5A). The Force stage contributes the most computational cost of a simulation step for simulations that include implicit inter-particle interactions (Fig 5C). As such, Tissue Forge supports offloading implicit interactions to available GPUs, both when running windowless and rendering in real time. Performance improvements from GPU acceleration also significantly vary due to both simulation details and computing hardware. For example, we tested simulating an implicit interaction for varying particle number but constant particle density (*i.e.*, with proportional increase in size of the simulation domain), varying cutoff distance and varying computing architecture (Intel i9-12900H CPU with and without acceleration on a NVIDIA A2000 GPU). We found that the computational cost per particle of simulation executed only on a CPU began to increase for particle numbers above 1M, while the cost remained approximately the same when using GPU acceleration. We also found that, in this upper range of particle number and all tested cutoff distances, GPU acceleration produced a speedup of around two. Source code for all benchmarks is available in S3 File.

Tissue Forge supports runtime-configurable GPU acceleration of a simulation. GPU acceleration can be configured, enabled, disabled and reconfigured at any time during a simulation. This modular, configurable approach allows fine-grain control of computations to achieve

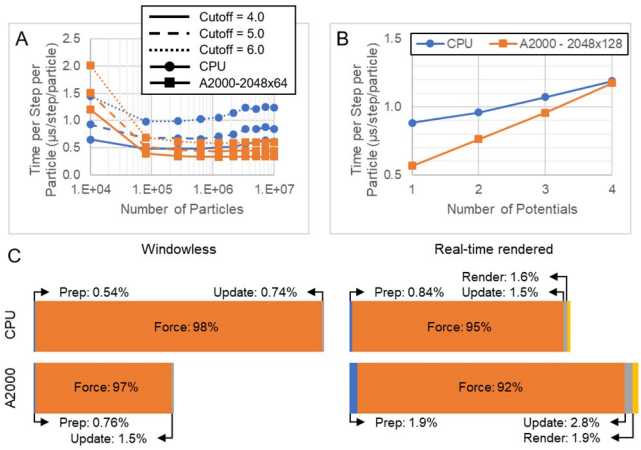

Fig 5. Tissue Forge performance metrics for windowless and real-time rendering modes. A: Computational cost per time step per particle for varying number of particles, varying cutoff distance and varying architecture when running windowless with fixed particle density and one implicit interaction. Computational cost generally increases with increasing cutoff distance. On a CPU, computational cost is lowest near 10k particles and then begins to increase. When offloading implicit interactions to a GPU, computational cost is generally less and tends towards a constant value. B. Computational cost for varying number of potentials defining implicit interactions with 10M particles and a cutoff of 5. Multiple potentials were implemented using potential arithmetic. Computational cost generally increases linearly with increasing number of potentials. C. Representative cost of solver stages when executing simulations from panel A with a cutoff of 5 in windowless (left) and real-time rendering (right) modes with implicit interactions calculated on a CPU (top) and GPU (bottom). Windowless mode simulated 10M particles. Real-time rendering mode simulated 10k particles. Size of bars for each mode represents relative cost of a simulation step. Bars are divided by solver stages that are ordered by execution order, and the area of each represents the portion of the total cost that the stage contributes. In both modes and architectures, force calculations make up the majority of the computational cost. As demonstrated in A, computing performance on a GPU is more efficient with increasing number of particles (Windowless), whereas computing performance on a CPU is more efficient for few particles (Real-time rendered).

https://doi.org/10.1371/journal.pcbi.1010768.g005

maximum performance for a given set of hardware, a particular simulation, and even a particular simulation state. For example, an event can periodically check whether implicit interactions should be performed on the CPU or offloaded to an available GPU based on the number of particles (Listing 5).

Listing 5. Selectively offloading computations to an available GPU when the number of particles exceeds a threshold.

```
# Define an event that governs where computations
#   for implicit interactions are performed
def maybe_to_gpu(e: tf.event.TimeEvent):
    eng_cuda = tf.Simulator.cuda_config.engine
    # If there are more than 1M particles,
    #   then probably GPU acceleration
    #   would improve performance
    if len(tf.Universe.particles) > 1E6:
        # Only send to GPU if not already on GPU
        if not eng_cuda.on_device():
            eng_cuda.set_threads(128)
            eng_cuda.set_blocks(1024)
            eng_cuda.to_device()
    else:
        # Only bring back from GPU if already on GPU
        if eng_cuda.on_device():
            eng_cuda.from_device()
```

The Tissue Forge API provides expressive syntax for combining potentials and forces as

process objects that can be bound to model objects. When a potential or force is defined as the sum of two potentials or forces, respectively, Tissue Forge creates a special *summation process object*. A summation process object forwards requests for energy or force calculations to their underlying constituent process objects and returns the sum of their results. For a potential summation process object U^{ab} defined as the sum of potentials U^a and U^b, U^{ab} is expressed as the form,

$$U^{ab}(r) = U^a(r) + U^b(r). \tag{11}$$

Summation process objects handle constituent process objects that are also summation process objects, as well as user-defined custom process objects (*i.e.*, custom forces and potentials). Energy and force calculations using summation process objects produce recursive requests for calculations through calls to constituent process objects. Summation process objects do not require special handling or storage by Tissue Forge. Hence, the computational cost of summation process objects is proportional to the cost of each constituent process object (Fig 5B). Summation process objects also eliminate the need to provide built-in potentials and forces for all possible combinations of individual process objects by allowing users to specify combinations through simple addition operations (Listing 6).

Listing 6. Creating and binding a complex potential using Tissue Forge potential arithmetic.

```
# Define two potentials
pot_a = tf.Potential.harmonic(k=1, r0=1)
pot_b = tf.Potential.coulomb(q=1)
# Combine the two potentials to create a sum potential
pot_ab = pot_a + pot_b
# Bind the potential to a pair of types
tf.bind(pot_ab ptype1, ptype2)
# Create another summation and bind to another pair of types
pot_c = tf.Potential.well(k=1, n=2, r0=1)
pot_abc = pot_ab + pot_c
tf.bind(pot_abc, ptype1, ptype3)
```

Results

Beyond the provided catalogue of built-in potentials, potential arithmetic (*e.g.*, a potential object as the sum of two potential objects) and support for user-specified custom potentials, Tissue Forge provides process objects for binding potential-based processes between specific particles (*i.e.*, a *bonded* interaction). Bonded interactions are a key component of MD modeling. Tissue Forge provides a number of bond-like processes to apply potentials for various types of bonded interactions. Each bonded interaction has a representative object that contains information about the bonded interaction (*e.g.*, which particles, what potential) that Tissue Forge uses to implement it during simulation. Currently, Tissue Forge provides the `Bond` for two-particle bonded interactions (where the potential is a function of the Euclidean distance between the particles, Fig 6, top left), the `Angle` for three-particle bonded interactions (where the potential is a function of the angle between the vector from the second to first particles and the vector from the second and third particles, Fig 6, top middle), and `Dihedral` (torsion angle) for four-particle bonded interactions (where the potential depends on the angle between the plane formed by the first, second and third particles and the plane formed by the second, third and fourth particles, Fig 6, top right). Like particles, all bonded interactions can be created and destroyed at any time during simulation, and bonded interactions can also be

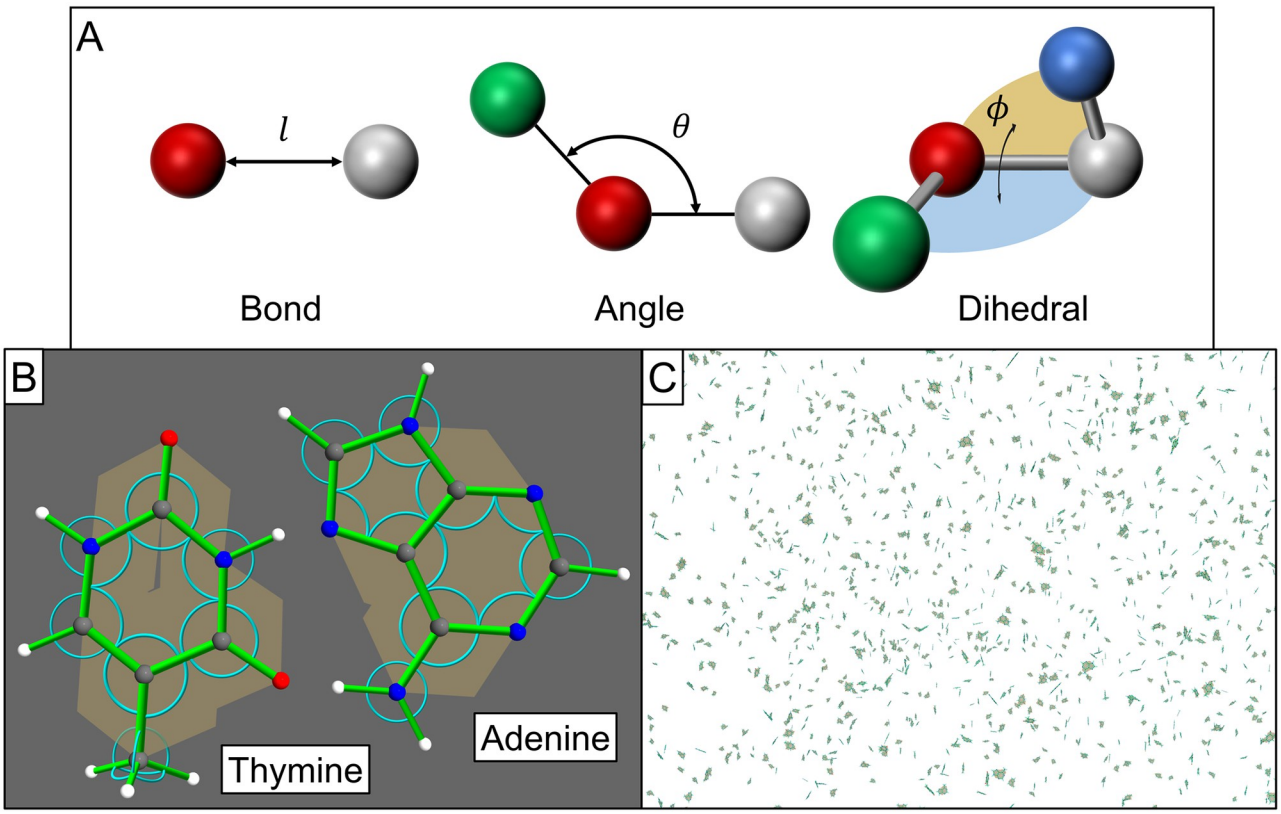

Fig 6. Molecular modeling and simulation with Tissue Forge. A: Classes of bonded interactions, where a measured property of the bond (length l for Bonds, angle θ for Angles, and planar angle ϕ for Dihedrals) is used as input to a potential function. B: Detailed view of thymine (left) and adenine (right) molecules constructed from Tissue Forge objects. Bonds shown as green cylinders, angles as blue arcs, and dihedrals as gold planes. C: Real-time simulation of a cloud of thymine and adenine molecules interacting via long-range potentials in a neutral medium.

https://doi.org/10.1371/journal.pcbi.1010768.g006

assigned a dissociation energy so that the bond is automatically destroyed when the potential energy of the bond exceeds its dissociation energy.

Tissue Forge supports combining aspects of object-oriented programming with primitive Tissue Forge objects to define complex model objects for use in simulations. When modeling the dynamics of biomolecules, each particle can represent an atom, the atomic properties of which are defined through the Tissue Forge particle type. Definitions of particular biomolecules, such as nucleobases like thymine and adenine (Fig 6B) can then be designed using generic Python (or other supported language) classes that construct an instance of a biomolecule by assembling Tissue Forge particles and bonded interactions according to experimental data. Tissue Forge facilitates the construction and deployment of software infrastructure to develop interactive simulations of biomolecular systems and processes (Fig 6C, see S4 File).

Particle-based methods are also useful for coarse-grained modeling of subcellular components, where the atoms of individual biomolecules, biomolecular complexes, or even organelles are omitted and instead represented by a single particle that incorporates the aggregate behavior of its constituents (*e.g.*, subcellular-element models). Tissue Forge supports coarse-grained subcellular modeling at various resolutions from the molecular to cellular scales, where a particle can represent a whole molecule, complex, or portion of an organelle or cytoplasm, to which coarse-grained properties (*e.g.*, net charge or phosphorylation state) and processes (*e.g.*, pumping of a solute, metabolism of a small molecule) can be applied.

For example, a particle can represent a portion of a lipid bilayer, in which case a sheet of such particle with appropriate binding and periodic boundary conditions can represent a section of a cell membrane. The Tissue Forge simulation domain can describe representative local spatial dynamics of the cell interface with its surrounding environment. Tissue Forge supports particle-based convection, providing a straightforward way to simulate a coarse-grained model of active transport at the cell membrane. Tissue Forge provides additional transport laws to model active pumping of species into or out of particles. To model transport at the cell membrane, these transport laws support implementing coarse-grain models of membrane-bound complexes like ion channels, which create discontinuities in concentrations of target species across the cell membrane (Fig 7, see S5 File).

At the coarsest scale of target applications, Tissue Forge provides support for particle-based modeling of multicellular dynamics. Tissue Forge provides a number of modeling features to support multicellular modeling at resolutions at or near the multicellular scale, where a particle can represent an individual cell, or a part of a cell. Overdamped dynamics describe the highly viscous, fluid-like collective motion of particle-based model cells, where short-range, implicit interactions can represent volume exclusion and contact-mediated intercellular interactions (*e.g.*, adhesion), long-range, implicit interactions can represent intercellular signaling via soluble signaling, and particle state vectors can describe the intracellular state.

For example, particle-based model descriptions have been previously used to describe cells as a set of particles (*e.g.*, a Tissue Forge cluster, Fig 1B) when modeling the process of spheroid fusion in tissue bioprinting, where cohesive cell shape is maintained by Lennard-Jones and harmonic potentials between particles of the same cell, and intercellular adhesion occurs by a Lennard-Jones potential between particles of different cells [12]. In a simpler model, representing each cell as a single particle and intercellular interactions with a single Morse potential can

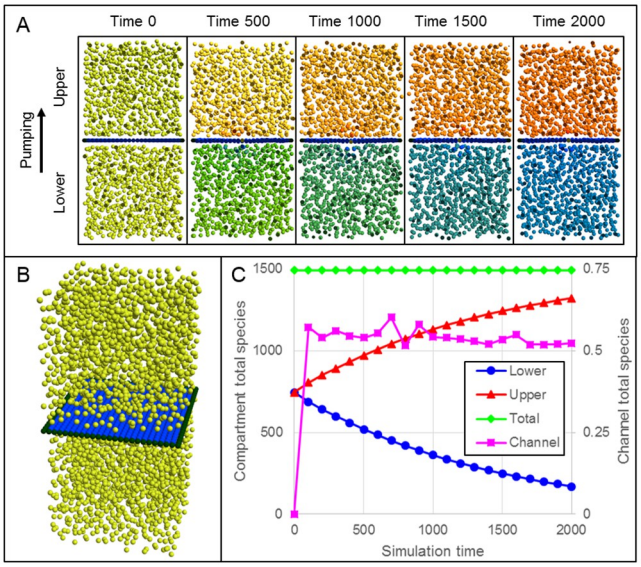

Fig 7. Active pumping of a diffusive species across a deformable membrane separating two fluid-filled compartments. A: Cut-plane views during simulation of two fluid-filled compartments separated by a deformable membrane, where each fluid is uniformly initialized with an initial concentration of a species. Particle color indicates species concentration with red as high, yellow and green as intermediate, and blue as low concentration. The membrane contains a particle that actively pumps the species from the lower to the upper compartment. B: Three-dimensional view of initial simulation state. C: Measurements of total species amounts in the lower (blue, circles), upper (red, triangles) and both (green, diamonds) compartments (left-hand vertical axis), and in the channel (magenta, squares, right-hand axis), during simulation.

https://doi.org/10.1371/journal.pcbi.1010768.g007

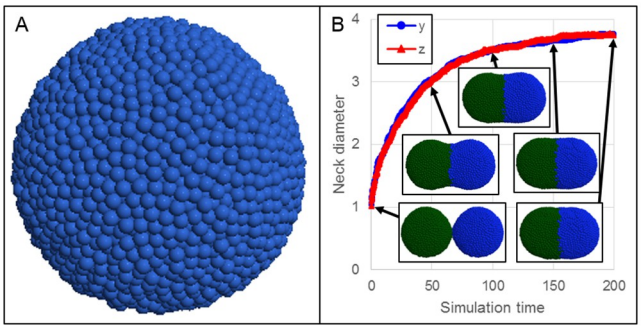

Fig 8. Simulating fusion of multicellular, homotypic spheroids. A: Spheroids of 12.5k cells each were individually pre-assembled, as in typical bioprinting practice. B: Two spheroids (green and blue) placed in close proximity fuse over time, as measured by the neck diameter along the *y* (blue circles) and *z* (red triangles) directions, which grows over time. The neck diameter along a direction is measured as the largest distance along the direction between any two particles at the mid-plane. Insets show the simulation at times 1, 50, 100, 150 and 200.

https://doi.org/10.1371/journal.pcbi.1010768.g008

also produce emergent fusion of spheroids like those used in bioprinting of mineralized bone (*i.e.*, about 12.5k cells per spheroid, Fig 8) [14]. When coupled with modeling diffusive transport and uptake like the scenario demonstrated in Fig 7, a Tissue Forge-based framework for the simulation of nutrient availability during spheroid-dependent biofabrication could support detailed modeling of spheroid viability in large tissue constructs [15].

Tissue Forge built-in features, APIs and the event system allow *ad hoc* construction of custom agent-based models (ABMs) of multicellular systems. The Tissue Forge event system allows the user to inject any custom model rule or algorithm into a simulation as defined by a user-specified function (or by several functions). When the function is called by the Tissue Forge event system, the function has access to the entire Tissue Forge API and hence can perform operations on individual cells (*e.g.*, differentiation, splitting, death) using global (*e.g.*, time), local (*e.g.*, nearby cells) and/or agent (*e.g.*, subcellular model) information.

For example, particle-based methods (and others) have been shown to support modeling cell proliferation, differentiation and migration in the intestinal crypt of intestinal epithelia [16]. The intestinal crypt consists of well-organized cellular dynamics, where cells within the crypt proliferate more rapidly compared to cells at the base of the crypt. A cellular dynamics ABM can describe such organization by assigning state dynamics that represent the cell cycle to each cell, where the cell cycle occurs only for cells located far enough within the crypt. In such a model, the geometry of the colonic crypt can be simplified to a two-dimensional configuration (as if unfolding the crypt) and each cell can be in one of four phases (*i.e.*, G1, S, G2, M) of the cell cycle, advances through each phase after a phase-specific period expires, divides after transitioning from the M phase to the G1 phase, and is removed upon reaching the base of the crypt. The ABM can introduce additional biophysical complexity like drawing the period of the G1 phase from a normal distribution for each cell or inhibiting advancement of the cell cycle by contact inhibition. This ABM has been shown to produce a cellular population that are all descendants of a single cell of the initial population over time by assigning a clone identification to each cell of the initial population and copying that identification to all progeny during cell division. Tissue Forge supports implementing this ABM with each cell represented as a particle and visualizing the clone identification of each cell by rendering each cellular particle according to is clone identification (*e.g.*, two cells with the same clone identification are rendered with the same color, Fig 9, see S6 File).

The Tissue Forge architecture also allows users to combine the simulation capability of Tissue Forge with other software packages that focus on different types of dynamical systems,

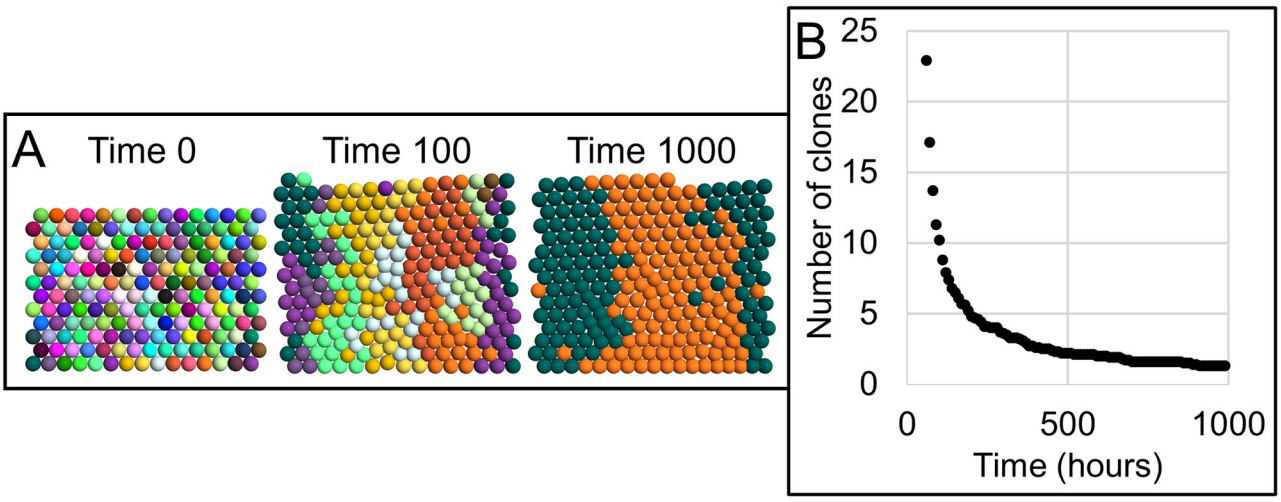

Fig 9. Two-dimensional agent-based model of cell proliferation and differential in the colonic crypt. A: Cells represented as particles are arranged in a sheet as if fixed on an unfolded cylindrical surface, where the upper boundary is treated as the base of the crypt and periodic boundary conditions are applied along the horizontal direction, as in [16]. Each cell is assigned a state dynamics model of the cell cycle and unique clonal identification (visualized as a unique particle color). The cell cycle model of each cell progresses through G1, S, G2 and M phases with deterministic periods except for G1, the period of which is randomly selected from a normal distribution for each cell. When the cell cycle model of a cell transitions from the M phase to the G1 phase, the cell divides and copies its clonal identification to its progeny. Cells are removed when they reach the base of the crypt. B: Number of clones during simulation time, measured as the mean of ten simulation replicates. Over time, the crypt tends toward monoclonality.

https://doi.org/10.1371/journal.pcbi.1010768.g009

such as libSBML [17] and libRoadRunner [18]. In the case of subcellular modeling coupled with multicellular ABMs, the Tissue Forge and libRoadRunner Python APIs make instantiating and managing executable SBML models and coupling their dynamics with those of individual cells straightforward (Listing 7).

Listing 7. Instantiating executable SBML models in libRoadRunner and associating them with Tissue Forge cellular particles. Comments are shown in green.

```
# Get the Tissue Forge and libRoadRunner Python libraries
import tissue_forge as tf
from roadrunner import RoadRunner
# Create global storage of SBML models by cell id
cell_models = dict()
# Define a function that creates an executable SBML model
#   from a given file path for a given cell, and
#   update the cell's species value accordingly
def attach_sbml_model(sbml_file: str, cell: tf.ParticleHandle):
    global cell_models
    this_cell_model = RoadRunner(sbml_file)
    cell_models[cell.id] = this_cell_model
    cell.species.D.value = this_cell_model['D']
# Define an event that integrates all SBML models and
#   updates associated cells
def integrate_sbml(t1: float, t2: float):
    for pid, rr in cell_models.items():
        rr.simulate(t1, t2)
        cell = tf.ParticleHandle(pid)
        cell.species.D.value = rr['D']
        cell.radius = 10 / (1 + rr['D'])
```

Tissue Forge support for multicellular ABMs couples well with built-in features for flux transport and application-specific models. For example, the provided species transport over particles arranged in a regular grid can produce the same solution for diffusive transport as when solved

using a discretized partial differential equation on a regular grid. In this case, the particles represent field measurements of local concentration, and those measurements can affect the processes of nearby cells in an ABM. Note that in such cases, particles representing field measurements are distinct from particles representing cells or cellular components (likely as different particle types), and the interactions between particles representing field measurements and particles representing cells or cellular components would almost certainly be chemical. Furthermore, as previously mentioned, Tissue Forge provides a dedicated space for developing user-custom and scale-specific features. This space, which in the Tissue Forge source code is referred to as *models*, is populated at the time of this writing with an implementation of a multicellular-level model of cell polarity that produces common epithelial tissue morphologies [19]. These combined features make agent-based, multiscale, multiphysics modeling of spatially heterogeneous tissues possible in Tissue Forge. For example, Delta-Notch signaling is well known to produce spatial patterns of alternating Delta or Notch expression, where cells expressing the Delta ligand perform contact-mediated lateral inhibition of Delta expression in neighboring cells through activation of the Notch receptor [20]. Delta-Notch signaling has been previously described by a simple system of ODEs [21], which has been integrated as an embedded subcellular model in multicellular simulation using CC3D [22] and CHASTE [16]. The same ODE model and contact-mediated intercellular interactions can be implemented in Tissue Forge (Fig 10A) and coupled with diffusion of soluble signals (Fig 10B) to simulate environmental activation of Delta-Notch signaling (Fig 10C, see S7 File).

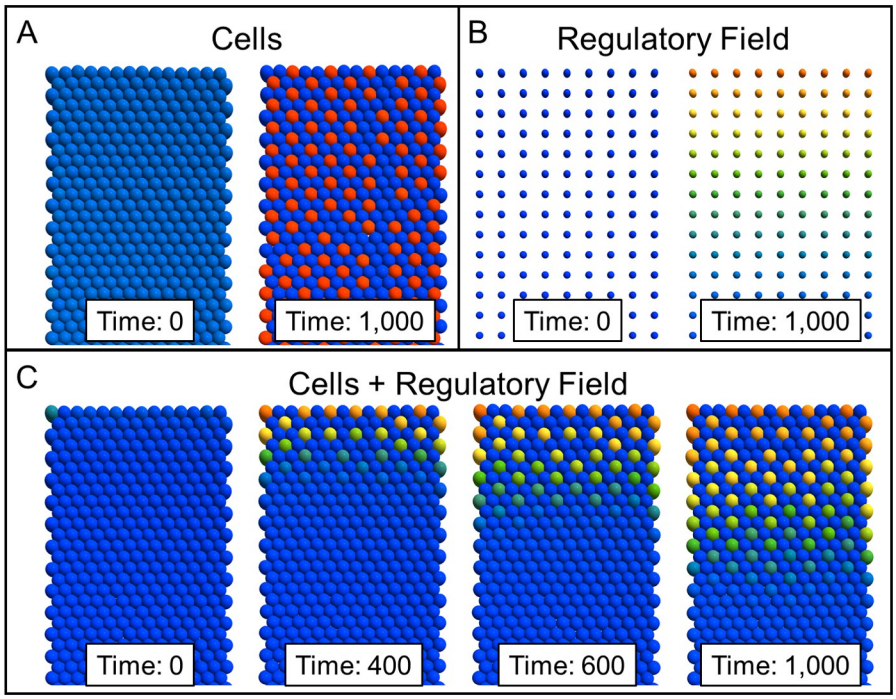

Fig 10. Delta-Notch signaling in a cellular monolayer subject to environmental control via a soluble signal. A: Lateral inhibition of Delta expression (cells with high Delta shown as red, cells with low Delta as blue) through contact-mediated Delta-Notch signaling without environmental regulation produces patterns in monolayers. Initially (left) Delta expression is approximately uniform but over time a pattern spontaneously emerges through contact-mediated signaling (right). B: Diffusion of a regulatory signal that induces Delta-Notch signaling along a uniform grid of particles representing point measurements of concentration. Initially (left) the cells are not exposed to the signal, but over time (right) the signal diffuses from the top boundary. C: When Delta-Notch signaling depends on induction via local concentration of the regulatory signal in B (not shown), pattern formation in the monolayer follows the propagation of the diffusive signal across the monolayer.

https://doi.org/10.1371/journal.pcbi.1010768.g010

Discussion

The Tissue Forge modeling and simulation framework allows users to interactively create, simulate and explore models at biologically relevant length scales. Accessible interactive simulation is key to increasing scientific productivity in biomodeling, just as simulation environments are fundamental to other fields of modern engineering. Tissue Forge supports both interactive runs with real-time visualization for model development, and headless execution for data generation and integrated applications. In addition, Tissue Forge supports user-specified model features (*e.g.*, custom particle types, forces and potentials) and scheduled and keyboard-driven simulation events, with intuitive user interfaces, in multiple programming languages and frameworks, supporting beginner- to expert-level programmers and beginner- to expert-level biomodelers. As demonstrated by our example of embedded ODE modeling, Tissue Forge also supports integration with other software packages that specialize in other types of models, methods, and applications.

Tissue Forge is open-source and freely available under the LGPL v3.0 license (https://github.com/tissue-forge/tissue-forge). Pre-built binaries are available in C, C++ and Python on 64-bit Windows, MacOS and Linux systems via conda (https://anaconda.org/tissue-forge/tissue-forge). Online documentation provides information on project philosophy, installation, walk-throughs, examples (in Jupyter Notebooks, https://github.com/tissue-forge/tissue-forge/tree/main/examples/py/notebooks) and API documentation for all supported languages. It has automated build updates to maintain synchronization between software versions and documented features (https://tissue-forge-documentation.readthedocs.io), including details on features not described in this paper (*e.g.*, species transport, boundary conditions). Tissue Forge's transparent development cycle, with automated continuous integration and continuous delivery, rapidly and reliably delivers the latest features to users (https://dev.azure.com/Tissue-Forge/tissue-forge). Instructions for installing Tissue Forge are available in S1 File.

Tissue Forge applies the abstraction of a particle to support modeling applications over a wide range of scales, ranging from sub-nanometer to hundreds of micrometers and beyond. It supports future development and integration of advanced numerical and computational methods for incorporating and/or generating biological information with increasingly greater detail. Tissue Forge provides a designated space for development of application-specific models and methods by both the development team and user community, and so is free to grow and evolve into other computational domains with significant relevance and impact to a number of scientific communities. To this end, we are preparing a followup manuscript that demonstrates advanced modeling and simulation features, detailed model construction in specific applications, and relevant features that are currently under development. Tissue Forge features under development include improvements to core Tissue Forge simulation capability (*e.g.*, multi-GPU support and libRoadRunner integration for network dynamics modeling), additional modeling features (*e.g.*, new built-in potentials, forces and multicellular ABMs, support for improper angles in MD modeling), enhanced user experience (*e.g.*, a graphical event interface), and additional modeling methodologies and solvers (*e.g.*, vertex and subcellular element models). Current work includes implementing general vertex model capability [23], which will provide support for mixed particle- and mesh-based modeling methodologies necessary to describe multicellular dynamics with explicit cell shapes [24]. Future work will also develop supporting features that facilitate integration with other software packages, such as coupling with advanced PDE solvers in Python (*e.g.*, FiPy [25]) and C++ (*e.g.*, FeNiCS [26], MFEM [27]), visualizing with different rendering schemes (*e.g.*, The Visualization Toolkit [13]), and supporting compositional modeling (*e.g.*, Vivarium [28]).

Supporting information

S1 File. Installation instructions. Instructions for installing pre-built Tissue Forge binaries.
(PDF)

S2 File. Oscillator example. Jupyter Notebook that simulates a simple oscillator with two particles.
(IPYNB)

S3 File. Benchmarks. Source code for benchmarks for variable cutoff distance, variable number of potentials, or algorithmic cost per simulation step.
(ZIP)

S4 File. DNA example. Python script that constructs adenine and thymine nucleobases on the basis of individual atoms using Tissue Forge particles.
(PY)

S5 File. Membrane transport example. Jupyter Notebook that simulates a neighborhood at a deformable membrane separating two fluids and active transport between them.
(IPYNB)

S6 File. Colonic crypt example. Jupyter Notebook that simulates a two-dimensional representation of cellular dynamics in the colonic crypt.
(IPYNB)

S7 File. Delta-Notch example. Jupyter Notebook that simulates Delta-Notch signaling in a monolayer with interactive control of environmental regulation.
(IPYNB)

S8 File. Source code. Tissue Forge version 0.1.1 source code.
(ZIP)

Author Contributions

Conceptualization: T. J. Sego, James P. Sluka, Herbert M. Sauro, James A. Glazier.

Data curation: T. J. Sego.

Formal analysis: T. J. Sego.

Funding acquisition: T. J. Sego, James P. Sluka, Herbert M. Sauro, James A. Glazier.

Investigation: T. J. Sego.

Methodology: T. J. Sego, James P. Sluka, Herbert M. Sauro, James A. Glazier.

Project administration: T. J. Sego, Herbert M. Sauro, James A. Glazier.

Resources: T. J. Sego, James P. Sluka, Herbert M. Sauro, James A. Glazier.

Software: T. J. Sego.

Supervision: T. J. Sego, Herbert M. Sauro, James A. Glazier.

Validation: T. J. Sego.

Visualization: T. J. Sego.

Writing – original draft: T. J. Sego, James P. Sluka, Herbert M. Sauro, James A. Glazier.

Writing – review & editing: T. J. Sego, James P. Sluka, Herbert M. Sauro, James A. Glazier.

References

1. Swat M., Thomas G., Belmonte J., Shirinifard A., Hmeljak D. & Glazier J. Multi-Scale Modeling of Tissues Using CompuCell3D. *Methods In Cell Biology*. 110 pp. 325–366 (2012), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3612985/ https://doi.org/10.1016/B978-0-12-388403-9.00013-8 PMID: 22482955
2. Starruß J., Back W., Brusch L. & Deutsch A. Morpheus: a user-friendly modeling environment for multiscale and multicellular systems biology. *Bioinformatics*. 30, 1331–1332 (2014,5)
3. Graner F. & Glazier J. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Physical Review Letters*. 69, 2013–2016 (1992,9), https://link.aps.org/doi/10.1103/PhysRevLett.69.2013, Publisher: American Physical Society
4. Ghaffarizadeh A., Heiland R., Friedman S., Mumenthaler S. & Macklin P. PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems. *PLOS Computational Biology*. 14, e1005991 (2018,2), https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005991, Publisher: Public Library of Science
5. Mirams G., Arthurs C., Bernabeu M., Bordas R., Cooper J., Corrias A., Davit Y., Dunn S., Fletcher A., Harvey D., Marsh M., Osborne J., Pathmanathan P., Pitt-Francis J., Southern J., Zemzemi N. & Gavaghan D. Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLOS Computational Biology*. 9, e1002970 (2013,3), https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002970, Publisher: Public Library of Science
6. Sandersius S. & Newman T. Modeling cell rheology with the Subcellular Element Model. *Physical Biology*. 5, 015002 (2008,4), https://iopscience.iop.org/article/10.1088/1478-3975/5/1/015002
7. Fortuna I., Perrone G., Krug M., Susin E., Belmonte J., Thomas G., Glazier J. & Almeida R. CompuCell3D Simulations Reproduce Mesenchymal Cell Migration on Flat Substrates. *Biophysical Journal*. 118, 2801–2815 (2020,6), https://www.sciencedirect.com/science/article/pii/S0006349520303490
8. Thompson A., Aktulga H., Berger R., Bolintineanu D., Brown W., Crozier P., Veld P., Kohlmeyer A., Moore S., Nguyen T., Shan R., Stevens M., Tranchida J., Trott C. & Plimpton S. LAMMPS—a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*. 271 pp. 108171 (2022,2), https://www.sciencedirect.com/science/article/pii/S0010465521002836
9. Anderson J., Glaser J. & Glotzer S. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Computational Materials Science*. 173 pp. 109363 (2020,2), https://www.sciencedirect.com/science/article/pii/S0927025619306627
10. Phillips J., Braun R., Wang W., Gumbart J., Tajkhorshid E., Villa E., Chipot C., Skeel R., Kalé L.. & Schulten K. Scalable molecular dynamics with NAMD. *Journal Of Computational Chemistry*. 26, 1781–1802 (2005), https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20289 https://doi.org/10.1002/jcc.20289 PMID: 16222654
11. Hess B., Kutzner C., Spoel D. & Lindahl E. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal Of Chemical Theory And Computation*. 4, 435–447 (2008,3), Publisher: American Chemical Society
12. Shafiee A., McCune M., Forgacs G. & Kosztin I. Post-deposition bioink self-assembly: a quantitative study. *Biofabrication*. 7, 045005 (2015,11), Publisher: IOP Publishing
13. Schroeder W., Avila L. & Hoffman W. Visualizing with VTK: a tutorial. *IEEE Computer Graphics And Applications*. 20, 20–27 (2000,9), Conference Name: IEEE Computer Graphics and Applications
14. Sego T., Prideaux M., Sterner J., McCarthy B., Li P., Bonewald L., Ekser B., Tovar A. & Jeshua Smith L. Computational fluid dynamic analysis of bioprinted self-supporting perfused tissue models. *Biotechnology And Bioengineering*. 117, 798–815 (2020), https://onlinelibrary.wiley.com/doi/abs/10.1002/bit.27238 https://doi.org/10.1002/bit.27238 PMID: 31788785
15. Sego T., Kasacheuski U., Hauersperger D., Tovar A. & Moldovan N. A heuristic computational model of basic cellular processes and oxygenation during spheroid-dependent biofabrication. *Biofabrication*. 9, 024104 (2017,6), Publisher: IOP Publishing
16. Osborne J., Fletcher A., Pitt-Francis J., Maini P. & Gavaghan D. Comparing individual-based approaches to modelling the self-organization of multicellular tissues. *PLOS Computational Biology*. 13, e1005387 (2017,2), https://dx.plos.org/10.1371/journal.pcbi.1005387
17. Bornstein B., Keating S., Jouraku A. & Hucka M. LibSBML: an API Library for SBML. *Bioinformatics*. 24, 880–881 (2008,3), https://academic.oup.com/bioinformatics/article/24/6/880/194657
18. Somogyi E., Bouteiller J., Glazier J., König M., Medley J., Swat M. & Sauro H. libRoadRunner: a high performance SBML simulation and analysis library. *Bioinformatics*. 31, 3315–3321 (2015,10)

19. Nielsen B., Nissen S., Sneppen K., Mathiesen J. & Trusina A. Model to Link Cell Shape and Polarity with Organogenesis. *IScience*. 23, 100830 (2020,2), https://www.sciencedirect.com/science/article/pii/S2589004220300134
20. Bocci F., Onuchic J. & Jolly M. Understanding the Principles of Pattern Formation Driven by Notch Signaling by Integrating Experiments and Theoretical Models. *Frontiers In Physiology*. 11 (2020), https://www.frontiersin.org/articles/10.3389/fphys.2020.00929 https://doi.org/10.3389/fphys.2020.00929 PMID: 32848867
21. Collier J., Monk N., Maini P. & Lewis J. Pattern Formation by Lateral Inhibition with Feedback: a Mathematical Model of Delta-Notch Intercellular Signalling. *Journal Of Theoretical Biology*. 183, 429–446 (1996,12), https://www.sciencedirect.com/science/article/pii/S0022519396902337
22. Chen K., Srinivasan T., Tung K., Belmonte J., Wang L., Murthy P., Choi J., Rakhilin N., King S., Varanko A., Witherspoon M., Nishimura N., Glazier J., Lipkin S., Bu P. & Shen X. A Notch positive feedback in the intestinal stem cell niche is essential for stem cell self-renewal. *Molecular Systems Biology*. 13, 927 (2017,4), https://onlinelibrary.wiley.com/doi/10.15252/msb.20167324
23. Sego T., Comlekoglu T., Peirce S., Desimone D. & Glazier J. General, Open-Source Vertex Modeling in Biological Applications Using Tissue Forge. *Research Square*. pp. rs.3.rs-2886960 (2023,5)
24. Van Liedekerke P., Neitsch J., Johann T., Warmt E., Gonzàlez-Valverde I., Hoehme S., Grosser S., Kaes J. & Drasdo D. A quantitative high-resolution computational mechanics cell model for growing and regenerating tissues. *Biomechanics And Modeling In Mechanobiology*. 19, 189–220 (2020,2), http://link.springer.com/10.1007/s10237-019-01204-7
25. Guyer J., Wheeler D. & Warren J. FiPy: Partial Differential Equations with Python. *Computing In Science & Engineering*. 11, 6–15 (2009,5), http://ieeexplore.ieee.org/document/4814978/
26. Scroggs M., Dokken J., Richardson C. & Wells G. Construction of Arbitrary Order Finite Element Degree-of-Freedom Maps on Polygonal and Polyhedral Cell Meshes. *ACM Transactions On Mathematical Software*. 48, 1–23 (2022,6), https://dl.acm.org/doi/10.1145/3524456
27. Kolev, T & Dobrev, V Modular Finite Element Methods (MFEM). (Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States),2010), https://www.osti.gov/doecode/biblio/35738
28. Agmon E., Spangler R., Skalnik C., Poole W., Peirce S., Morrison J. & Covert M. Vivarium: an interface and engine for integrative multiscale modeling in computational biology. *Bioinformatics*. 38, 1972–1979 (2022,3), https://academic.oup.com/bioinformatics/article/38/7/1972/6522109