Simuscale: A Modular Framework for Multiscale Single-Cell Modelling

Samuel Bernard, Fabien Crauste, Olivier Gandrillon, Carole Knibbe, David Parsons

Simuscale: A Modular Framework for Multiscale Single-Cell Modelling

Samuel Bernard, Fabien Crauste, Olivier Gandrillon, Carole Knibbe, David Parsons

TECHNICAL REPORT
N° 0520
January 18, 2024
Project-Teams Dracula

ISSN 0249-0803 ISRN INRIA/RT--0520--FR+ENG

Simuscale: A Modular Framework for Multiscale Single-Cell Modelling

Samuel Bernard*, Fabien Crauste†, Olivier Gandrillon‡, Carole Knibbe§, David Parsons¶

Project-Teams Dracula

Technical Report n° 0520 — January 18, 2024 — 15 pages

Abstract: Simuscale is a multiscale, individual-based modelling platform for performing numerical simulations of heterogeneous populations of individual cells evolving in time and interacting physically and biochemically. Models are described at two levels: cellular level and population level. The cellular level describes the dynamics of single cells, as defined by the modeller. Cells have an internal state that includes default properties such as cell size and position, and may also include any other cell-specific state, such as gene or protein expression. The population level describes the mechanical constraints and biochemical interactions between cells. Cells evolve in bounded 3D domain, and can divide or die. Simuscale implements the physical simulator that manages the simulations at the population level. It delegates the details of cellular dynamics to each cell. This makes Simuscale modular, as it can accommodate any number of cell models with the same simulation, including models with different modelling formalisms. Biochemical interactions occur between cells that are in contact with each other, through intercellular signals. Intercellular signals can be known to all or to a subset of the cells only. Simuscale expects an input file describing the initial cell population and numerical options, it runs a simulation over a specified time interval, updating the cell population at given time steps, and generates an output file containing the state of each cell at each time step, and the tree of cell divisions and deaths.

Key-words: modelling, multiscale, individual-based model, population dynamics, cell, cell-cell interaction

* Université Lyon 1, Institut Camille Jordan CNRS UMR5208, Inria, Villeurbanne F-69100, France
† Université Paris Cité, CNRS, MAP5, F-75006 Paris, France
‡ ENS Lyon, LBMC CNRS UMR5239, Inria, Villeurbanne F-69100 France
§ INSA Lyon, CarMeN INSERM U1060, Université Lyon 1, INRAe U1397, Inria, Villeurbanne F-69100, France
¶ Inria, Villeurbanne F-69100, France

RESEARCH CENTRE
LYON – RHÔNE-ALPES

Campus La Doua
56 boulevard Niels Bohr CS 52132
69603 Villeurbanne Cedex

Simuscale: un outil pour la simulation multi-échelles de populations cellulaires

Résumé : Simuscale est une plateforme de modélisation multi-échelles et individu-centrée pour réaliser des simulations numériques de populations hétérogènes de cellules individuelles évoluant en temps et interagissant physiquement et biochimiquement. Les modèles sont décrits à deux niveaux : cellulaire et populationnel. Le niveau cellulaire décrit la dynamique de la cellule individuelle. Les cellules possèdent un état interne qui comprend notamment des propriétés par défaut comme la taille de la cellule et sa position, et peut aussi inclure d'autres états spécifiques à la cellule, comme l'expression de gènes ou de protéines. Le niveau populationnel décrit les contraintes mécaniques et les interactions biochimiques entre les cellules. Les cellules évoluent dans un domaine 3D borné, et peuvent se diviser ou mourir. Simuscale implémente le simulateur physique qui gère les simulations au niveau populationnel. Elle délègue les détails de la dynamique cellulaire à chacune des cellules. Cette délégation des tâches permet de rendre Simuscale modulaire, en ce qu'elle peut prendre en charge différents formalismes de modélisation. Les interactions biochimiques se produisent entre cellules en contact entre elles, au travers de signaux inter-cellulaires. Les signaux inter-cellulaires peuvent être reconnus en tout ou en partie par un sous-groupe de cellules donné. Simuscale prend en entrée un fichier spécifiant les populations cellulaires initiales et des options de simulation numérique. La simulation met à jour les populations cellulaires à pas de temps discrets sur un intervalle de temps fini, et génère en sortie les trajectoires cellulaires de chaque cellule, ainsi que l'arbre des divisions cellulaire et des morts.

Mots-clés : modélisation, multi-échelles, modèle, individu-centré, dynamique des populations, cellule, interaction cellule-cellule

Contents

1 Introduction

Systems biology is an approach that uses statistical and modelling tools to integrate biological knowledge at lower levels, gene, molecule, cells, with the aim to understand the "big picture" at higher levels: cell, tissues, organisms ([1] and Figure 1**a**). This raises two challenges. The first challenge is the mismatch between the available biological data or knowledge and the process of interest: there are often missing links that prevent the bridging of scales. Multiscale models are computational models that provide an explicit description of the links between the different scales, often with interactions that are bi-directional [2]. Many problems concern the determination of cell fate (differentiation, survival, death, proliferation) from biomolecular data, and the nature of cell decision mechanisms. The second challenge is heterogeneity, when outliers cannot be taken into account by averaged models, but are able to tilt the big picture. The main problem we face in computational implementation of multiscale models is linking or bridging the scales. Assuming that a cell can be described (observed) by a vector of low level markers X in a set Ω, what can be said about the fate of the cell, or of the tissue to which it belongs? Even with a detailed model of the molecular processes going on inside the cell, it is difficult to infer the fate of the cell unambiguously.

In this report we present the simulation platform Simuscale for multiscale and integrative modelling in systems biology. The development of Simuscale[1] addresses the two challenges mentioned above by providing a simplified single-cell-based simulation environment (Figure 1**b**). Simuscale is a versatile multiscale modelling plateform for efficient numerical simulation of large, heterogeneous populations of interacting cells. Cells are represented by oriented visco-elastic balls that evolve under mechanical constraints in a three-dimensional domain $W \subset \mathbb{R}^3$. Each cell possesses a volume, a position and an orientation. Additionally, each cell can be endowed with user-defined intrinsic intracellular dynamics. The intracellular states of each cell can be coupled to other cells in contact via biochemical signals, as if expressed on the surface of the cells. Macroscopic cell fate (division, death, growth, differentiation, or movement) is entirely dictated by the intracellular state of the cell. Physical and biochemical interactions evolve in a 3D environment. Simuscale takes the form of a *core* numerical simulation engine on top of which biological model plugins can be added by the user. Plugins are meant to be interoperable, so that different plugins (say, one for tumor cells and one for immune cells) can be used together in the same simulation. This feature relies on the fact that all cells share common properties: position, size, orientation, mechanical properties, and biochemical signals. The intracellular dynamics is encapsulated in each cells, so different types of formalism can co-exist: Boolean

[1] Simuscale is hosted at `https://gitlab.inria.fr/bernard1/simuscale`.

networks, ordinary or stochastic differential equations, piecewise-deterministic Markov processes. This reduces the need for re-implementing existing models. There are several existing modelling frameworks, such as `CompuCell3D` [3], `PhysiCell` [4], `Chaste` (Cancer, Heart and Soft Tissue Environment), `TiSim/CellSys` INRIA/IfADo [5], and `Tissue Forge` [6]. By contrast, Simuscale is not focused on mechanical interactions. It rather focus on intracellular dynamics and cell-cell interaction, by providing a modular and plug-and-play interface to run simulations with several interacting cell models.

2 Software architecture

Simuscale numerically simulates the time evolution of a heterogeneous population of interacting cells within a bounded spatial domain. Simulations are performed iteratively in time with discrete time steps. At each time step, intercellular signalling and mechanical interactions are first computed for each cell. The internal state of each cell is then updated, including cell size and position, and cell death and cell division processed. The update is performed synchronously.

Simuscale is structured around a simulation core that performs model initialization, time iteration, and generates the simulation output. Iterations are synchronous: mechanical, biochemical interactions and cell fate (death or division) are first computed, then cells are updated based on computed interactions. Cellular updating is delegated to user-defined plugins that implement the details on how cells should evolve.

Because the focus is not on the mechanical behaviour, the simulation core includes predefined cell movement behaviours. The user can easily implement three types of movement: non-moving cells (immobile), cells that can move due to mechanical constraints, but have no motility of their own (mobile), and cells that can move on their own (motile). Movement rules for motile cells is user-defined.

Biochemical signals are mediated by a signalling interface common to all plugins. The signalling interface consists in a list of signals that can be expressed and detected by each cell. During the core update, each cell scans the subset of signals expressed by neighbouring cells and integrates the total signal strength. There is no limit to how many different signals a cell can express or detect. Each plugin defines the subset of signals they can use, and there is not need for the subsets of different plugins to intersect.

The numerical iterations are performed over a predefined time interval $t \in [0, T]$. During iterations, a partial set of the state of each cell is recorded in a text file called `trajectory.txt`, in a tall format, i.e. each row corresponds to a single cell at a given time. Cell state includes: time, a unique cell identifier, spatial position, and radius. Optionally, the orientation and the expression of any subset of signals (expressed or not by the cell) can be recorded. The text file format was chosen so that it can easily be parsed into tables and data frames for post-processing by standard packages in `Python` or `R`, such as `panda` or `dyplr`. In addition to the trajectory file, a cell lineage tree is recorded in Newick format [7, 8] in the file `newick.tree`, and in a graph format (TGF) [9] in the file `treelist.tgf`. Numerical simulation algorithms are detailed below.

3 Plugins

Cellular models are implemented in plugins. A plugin is a class derived from the class `Cell` that implements the internal dynamics, the movement behaviour, intercellular signalling, growth, and cell division and death (Figure 2).

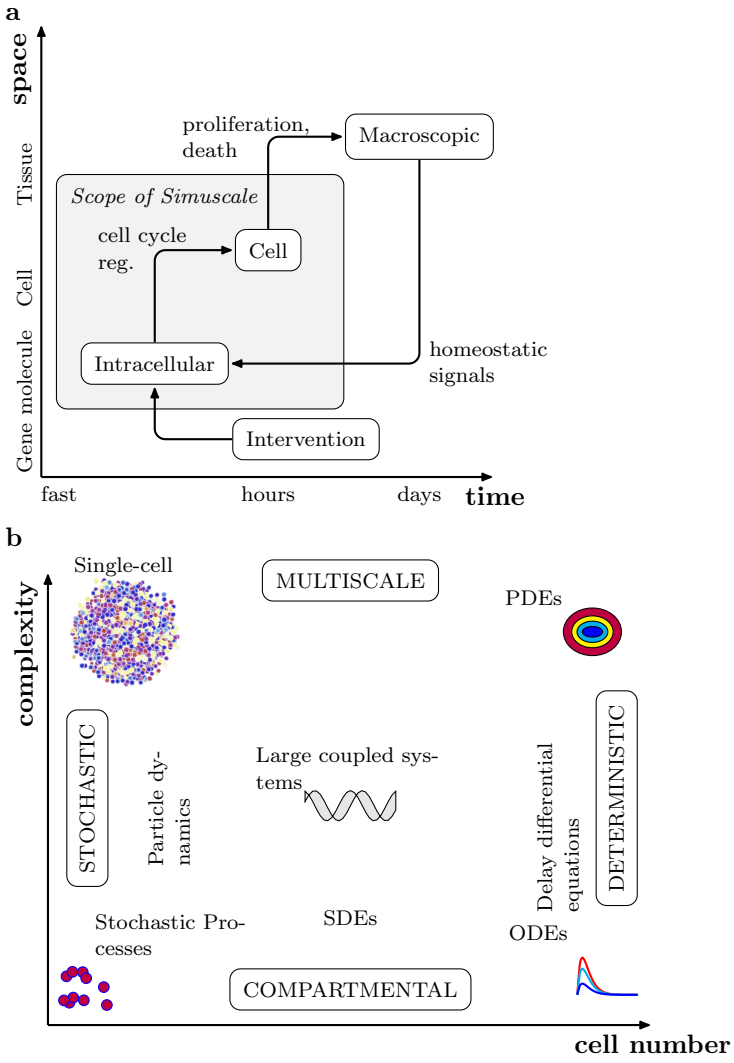

Figure 1: Scales (**a**) and modelling approaches (**b**) in systems biology. Simuscale is centrered around the cell (*grey area* in **a)** and *upper-left corner* in **b)**).

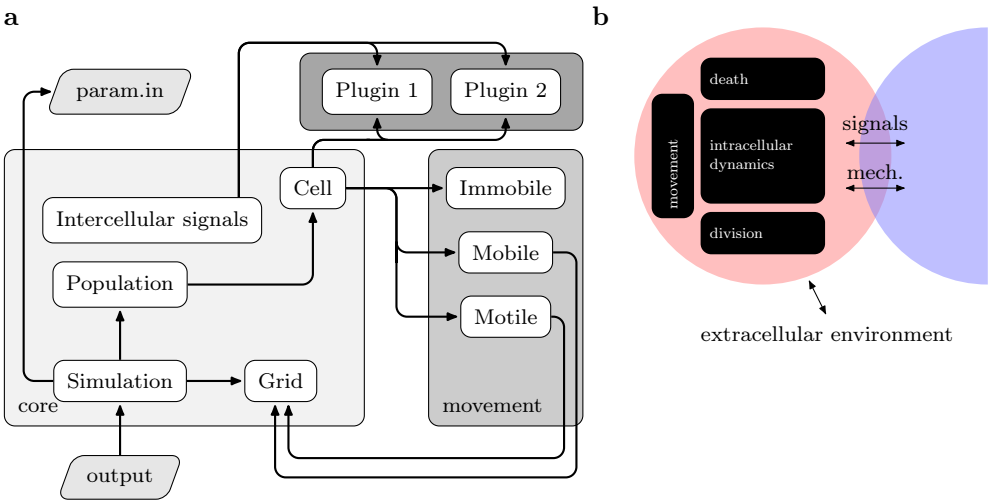

Figure 2: **a)** Architecture of Simuscale. Plugins are independent user-implemented cell models. They interact biochemically via common intercelular signals and mechanically through the movement module. **b)** Cell-cell interaction. Mechanical interactions provide the basic cellular spatial organisation. Biochemical interaction is based on a common set of intercellular signals.

Movement behaviour

Each cell is modelled as a visco-elastic ball with a soft shell and an incompressible core. The soft shell is the intersection of two concentric balls with radii r and s, $0 < r < s$. The core is the ball of radius r. The radius r is called the internal radius, and the radius s is call the external radius. The volume of a cell can change with time, but we assumed that the ratio of the internal to the external radius is a user-defined constant parameter R.

Cells have one of the three movement behaviours: immobile, mobile, or motile. Cell with immobile behaviour do not move. This behaviour is suited for cells that play a static role, and do not divide. Cells with mobile behaviour can move when forces are applied on them, but do not generate a force on their own. Cells with motile behaviour can move when forces are applied on them, and can also generate a force on their own, as defined by the model.

To model the mechanical interaction between cells, we use a capped Lennard-Jones type potential in a high-viscosity environment, i.e. cell displacement is proportional to the force applied to it. The force exerted between two cells from the Lennard-Jones potential is slightly repulsive (positive) when cells are close, but not in contact. The force becomes attractive, i.e. takes a negative value, when cells get in contact with each other, and become repulsive again when the distance between the centres of the two cells become less than the sum of the internal radii. Let r_1 and r_2 be the internal radii of the two cells, and δ the distance between the two cells. The force due to physical cell contact is

$$F = -24\epsilon\Big(2\frac{\sigma^{12}}{\delta^{13}} - \frac{\sigma^6}{\delta^7}\Big),$$

with parameters $\epsilon = 0.002$, $\sigma = (r_1 + r_2)/L$, and $L \sim 1.12$. When the ratio of the internal to the external radii is around 0.9, cells tend to stick together once they become in contact. When the distance between the centres of the two cells is less than the sum of the internal radii, the incompressible cores overlap, and the force quickly becomes repulsive. When δ is small, the

force F can become large. To avoid unnecessary numerical instabilities when solving equations of movement, we capped the force F by a constant value $F_{\max} = 0.5$,

$$F_c = \mathrm{cap}(F, F_{\max}),$$

where the capping function is defined by

$$\mathrm{cap}(x, M) = \begin{cases} x, & |x| \leq M, \\ \mathrm{sign}(x)M, & |x| > M. \end{cases}$$

The mechanical movement equations $\dot{x} = F$, $x \in \mathbb{R}^3$, are solved with an explicit updating scheme.

Intercellular signals

Cells interact by expressing intercellular signals to neighbouring cells. The intercellular signals expressed by a cell is a subset of a common set of intercellular signals. Neighbouring cells can pick up these intercellular signals. There is no limit to the number of intercellular signals. The full list of intercellular signals is maintained in a master file `InterCellSignal.values.in`. Signals can be defined in the list as needed. A signal name must be a valid `C++` identifier.

Intracellular dynamics

The intracellular dynamics is implemented as a blackbox, and is left to the user to define. The advantage of this approach if that in principle any computable model can be implemented. The drawback is the time cost of implementation.

Cell fate

Cells can divide and die. Rules for death and division are implemented in the plugin. At division, a daughter cell is created from the mother cell. The daughter cell shares the same class as the mother cell. The initial state of the daughter cell depends on the mother cell only. The state of the mother cell can be reset at division to allow the partitioning of biological material, for example. At death, the cell is instantaneously deleted from the simulation.

Cell type

The single-cell approach used in Simuscale means that cells do not have to have pre-defined cell types. However it can be useful to tag cells into a discrete classes, for computational or analytic purposes. The file `CellType.values.in` contains a list of cell types that can be used. As with the intercellular signal, any number of unique cell types can be added. Cell types must be valid `C++` identifiers.

Extracellular environment

Currently, simulations are exclusively cell-based, and the extracellular environment is empty. However, non-cell extracellular particles (such as macro-molecules) can be implemented as a specific plugin.

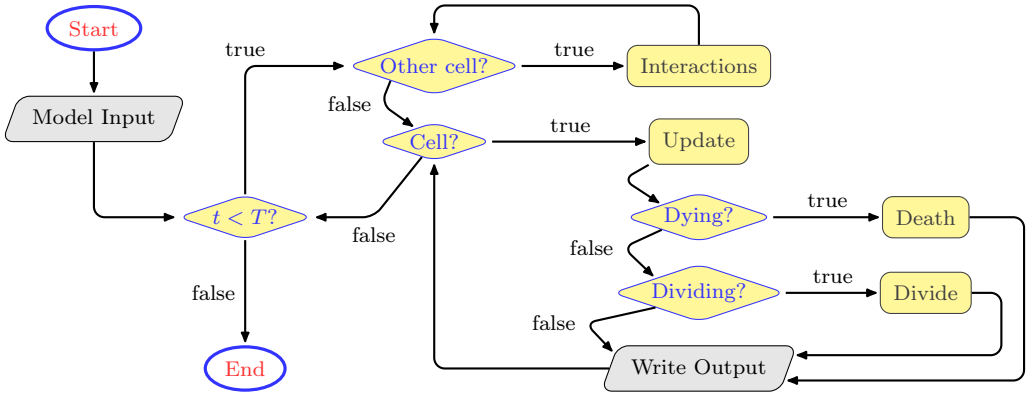

Figure 3: Flowchart of the numerical simulation in Simuscale .

4 Numerical simulations

Launching. Plugins are located in the directory `src/plugins`. As plugins are written directly as source code, a new compilation is necessary after changes in plugins. Simulation parameters are contained in a file named `param.in`, in which initial population sizes and types can be set. Simulation is launched with the executable `build/bin/simuscale`. Numerical simulations are performed iteratively with a discrete time step parameter `DT` hours, according to the flowchart shown in Figure 3.

Start. At the beginning of the simulation, cell populations defined in param.in are created. At cell creation, the constructor defined in the plugin is called. By default, spatial initial positions are chosen randomly. User-defined initial positions can be set in two ways: 1) by overriding the initial position in the constructor, and 2) by providing the positions in a text file, detailed below.

Initial cell populations are specified by the keyword `ADD_POPULATION`, followed by six mandatory parameters: number of cells, cell type, plugin, movement behaviour, doubling time, minimal volume. Optionally, the name of a file containing the initial positions and radii of each initial cell can optionally be provided. The initial position file is a space separated file with six columns:

```
L(int) X(float) Y(float) Z(float) R(float) C(int)
```

Only columns 2 to 5 (initial position (x, y, z) and the initial external radius c) are used at the moment. Cells are initialized in row order. The total number of rows must be at least equal to the number of cells in the population being added.

Iteration. Iterations are synchronous: mechanical forces and intercellular signal from neighbouring cells are are first computed. Cells are then updated independently: movement, orientation, internal status including the decision for cell death and cell division, followed by cell death and cell division. Cell death has priority on cell division. If a cell is marked for death and division, division will not occur.

End. Simulation ends when $t \geq T$. The complete cell lineage tree is written in two formats, newick (`newick.tree`) and TGF (`treelist.tgf`). The trajectory file can be visualised with the companion tool `build/bin/view`.

Parameter file

The parameter file `param.in` is necessary to run the simulations. The general syntax is

```
KEYWORD VALUE1 VALUE2 ...
```

Lines starting with # and empty lines are ignored. Keywords are listed in Table 1. Any number of lines with the keywords `ADD_POPULATION` and `SIGNAL` can be present in the parameter file.

Output files

The main output file is the text file `trajectory.txt`. It contains the trajectory of each indiviudal cell over the interval $t \in [0, T]$. Lineage trees are stored in `newick.tree` and `treelist.tgf`. The file `normalization.txt` is a space-separated value file that contains the extremal values of each signal present in the trajectory file the entire time interval: each line has the form `signal max min`. The normalization file is intended to be used with the companion script `view`, which can display a 4D rendering (3 spatial dimensions plus time) of the trajectory file. Each cell is identified by a unique cell ID, starting at ID=1, that remains the same across its life span. At division, a new cell is created with a unique ID.

The trajectory file is text file beginning with a header, and followed by a space-separated value data section. The header is formed by the lines starting with one of the characters `!`, `#` or `$`. The data section starts with the first line starting with a floating-point number. The line starting with a `!` specifies the world size in the x, y, z dimensions. The lines starting with a `#` or `$` describe each column in the data section. Column description has the form `column number : description`. Lines starting with the character `#` describe columns that have a fixed meanings: time, number of cells in the population, cell ID, x position, y position, z position, x orientation, y orientation, z orientation, external radius. Orientation are written to the trajectory file only if `WRITEORIENTATION` is set to 1. Lines starting with `$` describe signal column. Only signal specified in the parameter files are included in the trajectory file, even though any signal can be used in the simulation. The data section has a fixed number of columns as specified in the header. Each row records the state of a single cell at a given time (it is a tall table format with respect to time).

The newick file is a text-based file encoding a binary tree that records the whole cell lineage. Unlike many graph formats, the newick format provides a way to specify the length of each edge with the syntax `node:length`. Inner nodes (non-leaf, non-root nodes) represent cell division, with one branch for the mother cell and one branch for the daughter cell. Leaf nodes represent the end of existence of a cell (either death or end of simulation). A cell division is recorded as `(descendants)cellID>motherID:ageOfMother` where `descendant` is the tree with a root at `cellID`. Cell deaths are recorced as tree leaves `cellID:ageAtDeath`. Cells that are alive at the end of the simulation are recorded the same way, with `cellID:ageAtT`. A tree is a comma separated list of trees enclosed by parentheses and followed by `cellID>motherID:ageOfMother`. Because cells initially present at $t = 0$ do not have ancestors, a "root" cell with ID 0 is added to the tree to make it rooted. Therefore the newick tree always ends with

```
...>root:0.00)2>root:0.00)1>root:0.00;
```

The annotation is made compatible with the phylogenetic plot method from the `ape` package in `R` [10].

The trivial graph format is a text-based adjacency list format, i.e, the graph is represented as a list of edges. The file is divided in two parts: node definition and edge definition. Node definitions and edge definitions are separated by a line containing a single `#`. In the first part, each line defines a node: `node(uint32) label(string without space)`. The node is the cell ID, and the label is `<cell_type>-ID`. In the second part, each line defines a directed edge as a pair of nodes. The TFG format does not provide a native way to specify edge lengths, but

Keyword	Parameters	Description
`PRNG_SEED`	(integer)n or `AUTO`	pseudo-random generator seed n or automatically generated (`AUTO`)
`MAXTIME`	(float)T	maximal time T for the simulations
`DT`	(float)dt	core simulator time step
`BACKUP_DT`	(float)bdt	Backup time step
`ADD_POPULATION`	(int)N ...	cell number
	(CellType)*type* ...	*type* in `CellType.values.in`
	(CellFormalism)p ...	plugin name (`classKW_` in plugin file)
	(MoveBehaviour)m ...	m in {`IMMOBILE`, `MOBILE`, `MOTILE`}
	(float)d ...	volume doubling time, $d > 0$
	(float)v ...	minimal initial volume
	[(string)*init.file*]	initial positions file
`R_RATIO`	(float)r	ratio between internal and external radii
`SIGNAL`	(InterCellSignal)s	record signal s to trajectory file
`WORLDSIZE`	(float)x (float)y (float)z	size of the computational domain (default: 40, 40, 40)
`WORLDMARGIN`	(float)x (float)y (float)z	size of the computational domain (defaults to 2, 2, 2)
`USECONTACTAREA`	0 or 1	cell signals weighed by the cell-cell contact area (1) or not (0)
`WRITEORIENTATION`	0 or 1	record orientation coordinates to trajectory file

Table 1: Description of keywords in the parameter file `param.in`.

arbitrary labels can be specified after the edge definition. The time of birth and death of the daughter are recorded as edge label:

```
edge[mother(node) daughter(node)] label[birth(float) death(float)]
```

5 Test case: circadian clock-regulated tumour growth

As a test case, we re-implemented a model for the control of the cell cycle by the circadian clock [11]. In mammals and other organisms, cell division is known to occur at preferential times of the day. The circadian clock is an autonomous biological rhythm that has been characterized across all life domains. The circadian clock is generated and maintained by gene regulatory loops that modulate the activity of transcription factors and their downstream targets. The cell cycle genes *p21* (controlling cell growth) and *Wee1* (controlling mitosis) are under direct control of the circadian clock [12]. The circadian clock and the cell cycle models, implemented as coupled systems of ODEs, were adapted from previously published models [13, 11]. The circadian clock module is described by seven variables (Figure 4, *circadian clock* module), and the cell cycle by two variables (Figure 4, *cell cycle* module). The circadian clock transcription factor BMAL1 acts directly on the cell cycle variable X, which has a role similar to *Wee1*. BMAL1 activates X, which in turn blocks the transition to mitosis and cell division. As the activity of BMAL1 decrease during the day, X becomes less inhibited, and cell division can proceed. As an additional mechanism, the circadian clock synchronizes neighbouring cells through activation of the *Per2* and *Cry* circadian clock genes via a `Clock` signal. As a result, cells tend to divide at preferential times. Cell death occurs when local cell density rise, limiting the total population (Figure 4, *cell*

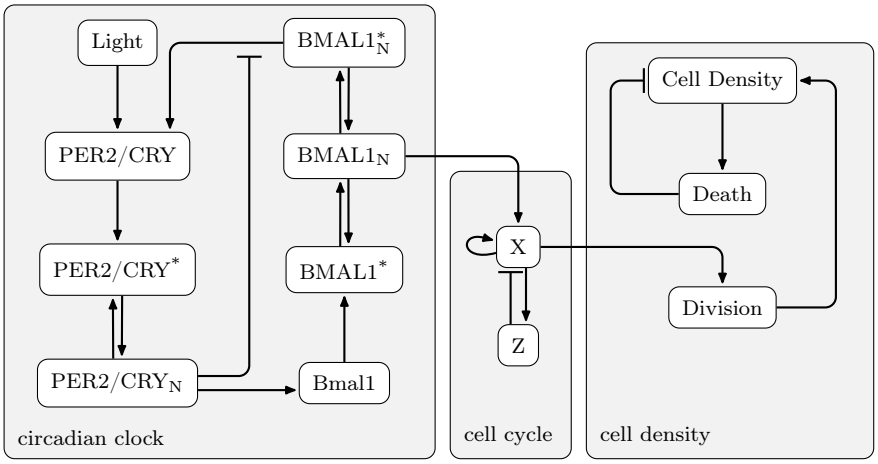

Figure 4: Circadian clock-driven cell cycle model. Adapted from reference [11].

File	Description
`currentdir/param.in`	Parameter configuration file
`src/plugins/Cell_SyncClock.cpp`	clock cell implementation
`src/plugins/Cell_SyncClock.h`	header file for clock cells
`src/plugins/Cell_Killer.cpp`	killer cell implementation
`src/plugins/Cell_Killer.h`	header file for killer cells

Table 2: Simulation files for the test case.

density module). For the purpose of the test case, the cells endowed with this model are called *cancer cells*. The file `param.in` is

```
PRNG_SEED          1421746927
MAXTIME            480.0
DT                 0.1
ADD_POPULATION     17 CANCER SYNC_CLOCK MOTILE 40 1.0 init_clock_4.txt
ADD_POPULATION     1 KILLER KILLER MOTILE 8 1.0 init_killer.txt
USECONTACTAREA     1
WRITEORIENTATION   1
R_RATIO            0.90
SIGNAL  CLOCK
SIGNAL  CYCLE
SIGNAL  DEATH
SIGNAL  KILL
```

The initial configuration consisted in 17 cancer cells located near the center of the computational domain (the `World`). An 18th cell, a killer cell, was added. Killer cells move randomly. Upon contact with cancer cells, the killer cell reduces its motility, increase the probability of clock cells to die, and starts proliferating. `Clock` and killer cells are modelled by distinct plugins, but they shared the same signal `Kill`: killer cells express the signal `Kill`, and clock cell can sense the signal. Cancer cells express two signals in addition to the `Clock` signal: `Cycle` and `Death`. High `Cycle` expression indicates a higher probability to divide, and a `Death` signal above 1.0 triggers cell death.

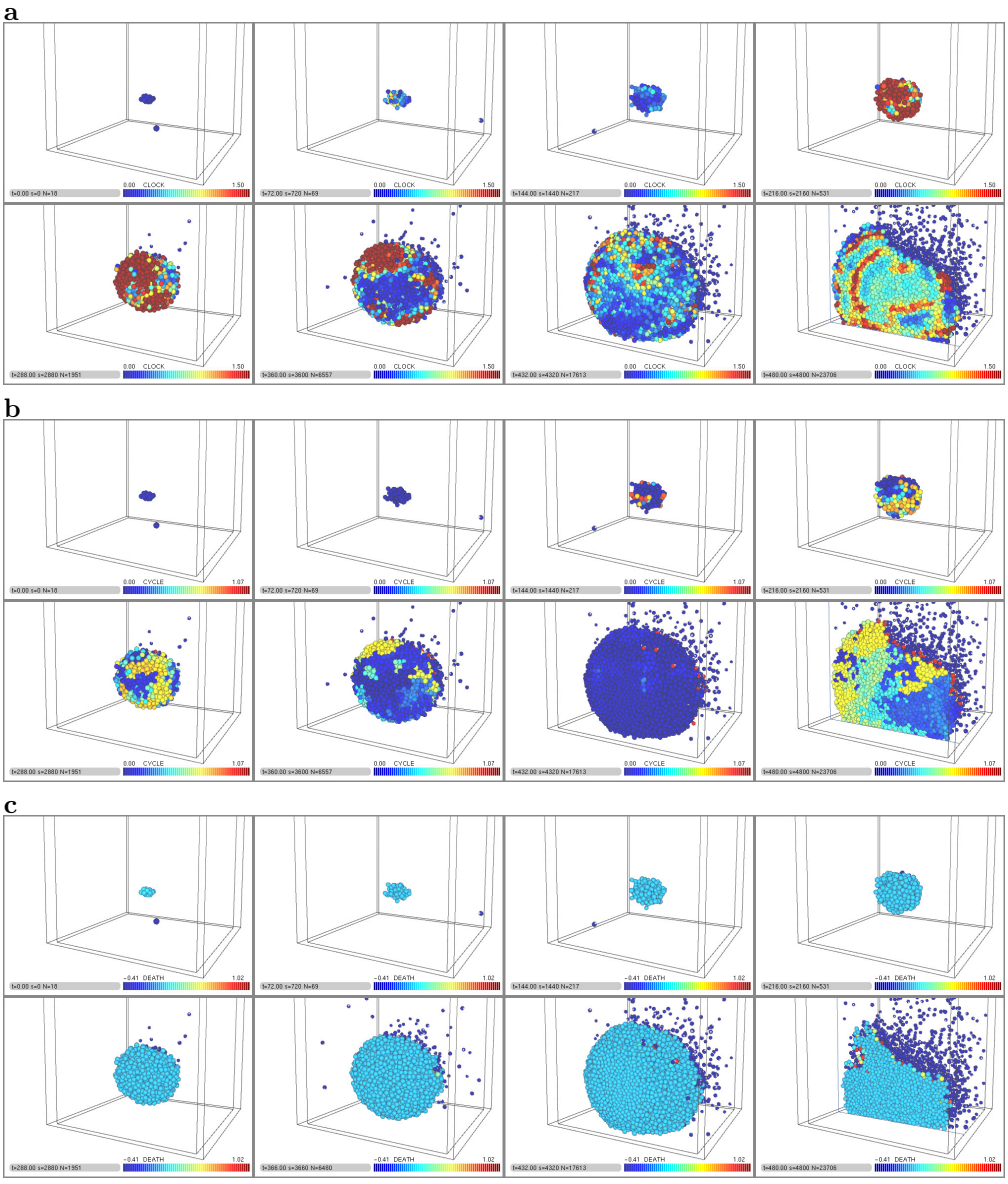

Figure 5: Tumour growth under a circadian clock control. **a)** `Clock` signal. The `Clock` signal synchronizes cells together, and drive the cell cycle. Clock synchronization decreases with tumour growth. **b)** `Cycle` signal. High levels indicates that the cells are more likely to divide. **c)** `Death` signal. Cell death occurs when the `Death` signal is above 1.0. States at time points 0, 72, 144, 216, 288, 360, 432 and 480 h. Cell types (Cancer and Killer) are best distinguished in panel **c)**, where cancer killer cells are in dark blue and cancer cells have colours corresponding to positive `Death` expression, from light blue to red.

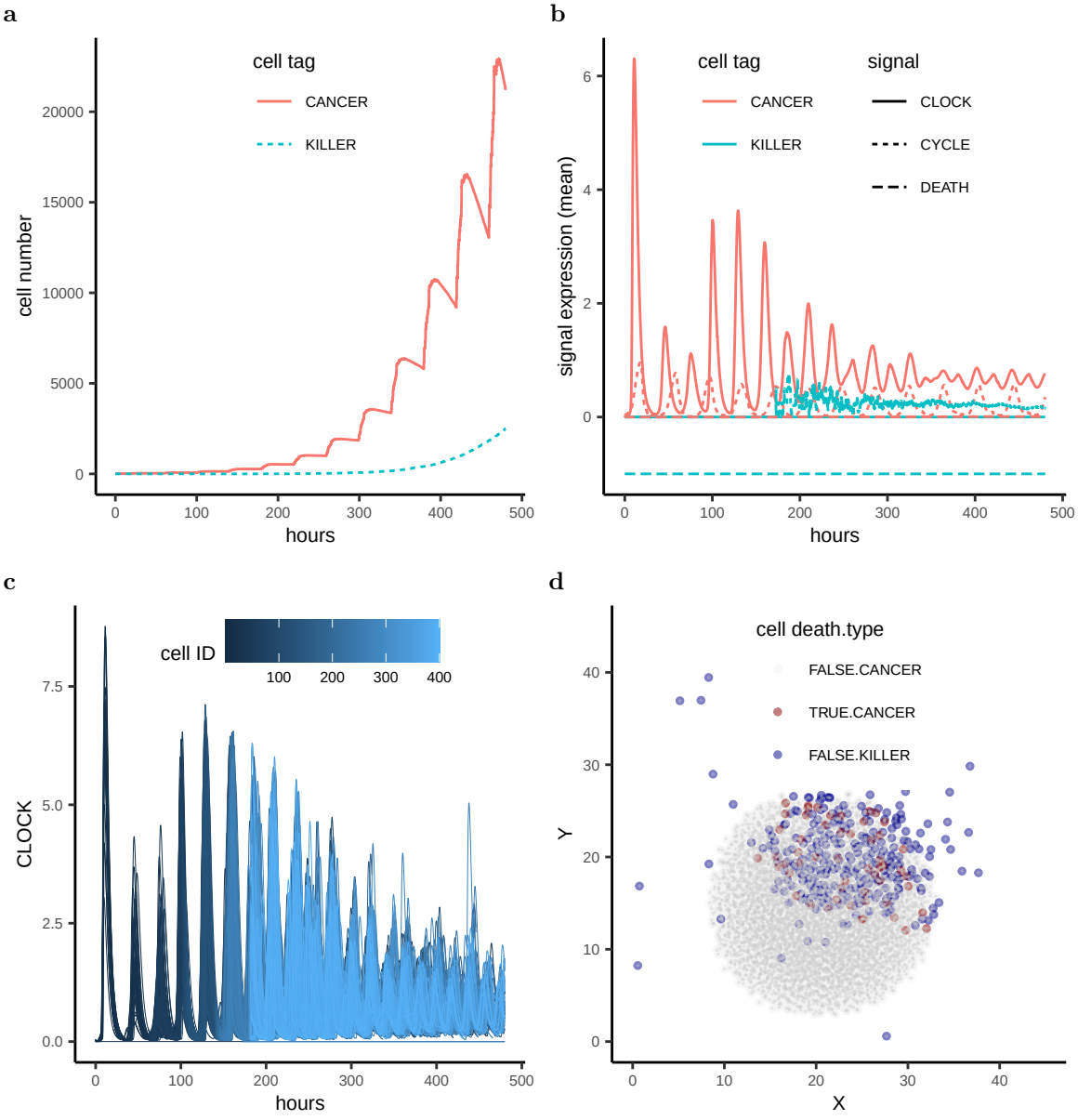

Figure 6: Total cell number **a)**, mean signal expression levels **b**, Single-cell `Clock` signal expression levels for the initial 400 cells, and the spatial distribution of the cell cycling status **d**. In (**d**, dying cancer cell (*red*) co-localize with the killer cells (*blue*).

The numerical simulation ran for t in $[0, 480]$ hours (Table 2). At $t = 480$h, the total cell number was 23,706 (cancer cells: 21,202; killer cells: 2504). The `Clock` and `Cycle` signals maintained a moderate amount of coherence during the first six days (144 hours), but `Clock` synchrony (and consequently `Cycle` synchrony) disappeared once the killer cell made contact with cancer cells and started proliferating (Figure 5**a,b** and Figure 6**a,b,c**). The `Death` signal was restricted to the zone in contact with the killer cells (Figure 6**d**).

6 Conclusion

Preliminary tests show that Simuscale is viable for modelling cell population in realistic settings. While the current release is functional, there is a number of planned features to be added in future releases. First, there is a need for a better user-interface, with a clearer separation between the core simulator and the plugins. Moreover, an integrated initial configuration tool could be helpful. Currently, modifying the initial state of the population requires changes in several files. Second, although there are many advantages to using a single-cell modelling approach, the lack of an extracellular environment can limit certain applications. For example, long-range intercellular interaction is not possible, because only cells in contact with each other can communicate. Adding a Gaussian field to cells could mimick the diffusion of molecular signals at longer ranges, at a light computational cost, thank to fast Gauss transform algorithms [14]. Numerical optimisation could take several form. Mixed-precision numerical schemes exploit reduced precision arithmetics to speed up computation without reduction in numerical accuracy[2]. This could be particularly relevant in the context of parallelisation on GPUs, where large amount of data transfer can severely limit speed [15]. Working with low precision arithmetics, such as half precision, could have the advantage of benefiting from the hardware acceleration on modern GPUs developed for machine learning. Finally, developing concurrent or post-mortem (once the simulation ended) data analysis tools will be useful for model or parameter inference and optimisation. Tools for macroscopic/continuum approximation could also be valuable to gain analytical insight, or to reduce the numerical complexity of the models [16].

References

[1] Mark Alber, Adrian Buganza Tepole, William R Cannon, Suvranu De, Salvador Dura-Bernal, Krishna Garikipati, George Karniadakis, William W Lytton, Paris Perdikaris, Linda Petzold, et al. Integrating machine learning and multiscale modeling—perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences. *NPJ Digital Medicine*, 2(1):115, 2019.

[2] S Bernard. How to Build a Multiscale Model in Biology. *Acta Biotheor*, 61:291–303, 2013.

[3] François Graner and James A Glazier. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Phys Rev Lett*, 69(13):2013, 1992.

[4] Ahmadreza Ghaffarizadeh, Randy Heiland, Samuel H Friedman, Shannon M Mumenthaler, and Paul Macklin. PhysiCell: an open source physics-based cell simulator for 3-D multicellular systems. *PLoS Comput Biol*, 14(2):e1005991, 2018.

[5] Stefan Hoehme and Dirk Drasdo. A cell-based simulation software for multi-cellular systems. *Bioinformatics*, 26(20):2641–2642, 2010.

[2] Inria exploratory project ExOde

[6] T. J. Sego, James P. Sluka, Herbert M. Sauro, and James A. Glazier. Tissue forge: Interactive biological and biophysics simulation environment. *PLoS Comput Biol*, 19(10):1–22, 10 2023.

[7] Joseph Felsenstein. *PHYLIP (phylogeny inference package), version 3.5 c.* Joseph Felsenstein, 1993.

[8] Newick format. `https://en.wikipedia.org/wiki/Newick_format`. Accessed: 05-12-2023.

[9] Trivial graph format. `https://en.wikipedia.org/wiki/Trivial_Graph_Format`. Accessed: 14-12-2023.

[10] ape: Analyses of phylogenetics and evolution. `https://cran.r-project.org/web/packages/ape/index.html`. Accessed: 08-12-2023.

[11] R El Cheikh, S Bernard, and N El Khatib. A multiscale modelling approach for the regulation of the cell cycle by the circadian clock. *J Theor Biol*, 426:117–125, 2017.

[12] Elham Farshadi, Gijsbertus TJ van Der Horst, and Inês Chaves. Molecular links between the circadian clock and the cell cycle. *J Mol Biol*, 432(12):3515–3524, 2020.

[13] R El Cheikh, S Bernard, and N El Khatib. Modeling circadian clock-cell cycle interaction effects on cell population growth rates. *J Theor Biol*, 363:318–331, 2014.

[14] Leslie Greengard and John Strain. The fast Gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[15] Chris Gregg and Kim Hazelwood. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pages 134–144. IEEE, 2011.

[16] Matthew J Simpson, Ruth E Baker, Pascal R Buenzli, Ruanui Nicholson, and Oliver J Maclaren. Reliable and efficient parameter estimation using approximate continuum limit descriptions of stochastic models. *J Theor Biol*, 549:111201, 2022.

RESEARCH CENTRE
LYON – RHÔNE-ALPES

Campus La Doua
56 boulevard Niels Bohr CS 52132
69603 Villeurbanne Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803