
Universiteit
Leiden
The Netherlands

The Ithildin library for efficient numerical solution of anisotropic reaction-diffusion problems in excitable media

Kabus, D.; Cloet, M.; Zemlin, C.; Bernus, O.; Dierckx, H.

Citation
Kabus, D., Cloet, M., Zemlin, C., Bernus, O., & Dierckx, H. (2024). The Ithildin library for efficient numerical solution of anisotropic reaction-diffusion problems in excitable media. *Plos One*, *19*(9). doi:10.1371/journal.pone.0303674

Version:	Publisher's Version
License:	Creative Commons CC BY 4.0 license
Downloaded from:	https://hdl.handle.net/1887/4246785

Note: To cite this publication please use the final published version (if applicable).

PLOS ONE

RESEARCH ARTICLE

The Ithildin library for efficient numerical solution of anisotropic reaction-diffusion problems in excitable media

Desmond Kabus[1,2], **Marie Cloet**[1], **Christian Zemlin**[3], **Olivier Bernus**[4], **Hans Dierckx**[1]*

1 Department of Mathematics, KU Leuven Campus Kortrijk (KULAK), Kortrijk, Belgium, **2** Laboratory of Experimental Cardiology, Leiden University Medical Center (LUMC), Leiden, The Netherlands, **3** Division of Cardiothoracic Surgery, Department of Surgery, University of Washington School of Medicine, St Louis, MO, United States of America, **4** Univ. Bordeaux, Inserm, Centre de Recherche Cardio-Thoracique de Bordeaux U1045, IHU Liryc, Hôpital Xavier Arnozan, Pessac, France

* h.dierckx@kuleuven.be

Check for updates

OPEN ACCESS

Citation: Kabus D, Cloet M, Zemlin C, Bernus O, Dierckx H (2024) The Ithildin library for efficient numerical solution of anisotropic reaction-diffusion problems in excitable media. PLoS ONE 19(9): e0303674. https://doi.org/10.1371/journal.pone.0303674

Editor: Rafael Sachetto Oliveira, Universidade Federal de Sao Joao del-Rei, BRAZIL

Received: April 29, 2024

Accepted: September 3, 2024

Published: September 19, 2024

Copyright: © 2024 Kabus 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 source code of version 3.5.1 of the Ithildin software implemented for this paper is publicly available at https://gitlab.com/heartkor/ithildin and has been archived on Zenodo (DOI: 10.5281/zenodo.12799245). This archive also contains the data generated by the simulations used throughout this paper. Documentation of the code is publicly available at https://heartkor.gitlab.io/ithildin/.

Funding: DK is supported by KU Leuven grant GPUL/20/012. MC is supported by KU Leuven

Abstract

Ithildin is an open-source library and framework for efficient parallelized simulations of excitable media, written in the C++ programming language. It uses parallelization on multiple CPU processors via the message passing interface (MPI). We demonstrate the library's versatility through a series of simulations in the context of the monodomain description of cardiac electrophysiology, including the S1S2 protocol, spiral break-up, and spiral waves in ventricular geometry. Our work demonstrates the power of Ithildin as a tool for studying complex wave patterns in cardiac tissue and its potential to inform future experimental and theoretical studies. We publish our full code with this paper in the name of open science.

1 Introduction

With the Ithildin framework, we want to open up new gateways in the numerical simulation of reaction-diffusion systems, such as the electrical activation patterns in the heart. (In this way, it is similar to its namesake in the *Lord of the Rings*, where Ithildin is an Elven substance that reveals a hidden gateway to another realm after a spell is cast [1]).

Our motivation to write a reaction-diffusion solver comes from the numerical study of electrical patterns inside the heart [2]. These patterns, which are incompletely understood, are a main cause of death and even as a chronic disease, they complicate people's lives. In the past decades, computer models of arrhythmia have allowed mechanistic insight in the origin and control of arrhythmias [3]. On the longer term, it is thought that digitized versions of patients' hearts could help offer better diagnostics and planning of procedures; such personalized heart models are called *cardiac digital twins* [4–7]. In view of open science, we have decided to share the code that has been steadily developed in our group since 2007 with the scientific community.

A flowchart outlining the functionality of Ithildin can be found in Fig 1. Ithildin is designed to comply with the 2011 version of the ISO-C++ standard [8], but it compiles with all newer versions, including the current 2023 ISO-C++ standard [9–12]. The software facilitates

grant STG/19/007 and FWO-Flanders fellowship, grant 11PMS24N. HD is supported by KU Leuven grant STG/19/007. OB is supported by Agence Nationale de la Recherche grant ANR-IHUA-04.

Competing interests: The authors have declared that no competing interests exist.

forward Euler and Runge-Kutta finite-difference solutions for reaction-diffusion systems in *N*-dimensional space, such as the monodomain equation for cardiac electrophysiology, with specified boundary conditions [2].

The framework offers quick computation through CPU parallelization using OpenMPI [13]. It also boasts decent documentation of available features, made accessible through Doxygen [14]. Ithildin writes easy-to-parse YAML log files to document simulation setups [15]. Additionally, it allows convenient output of frames of recorded variables at regular intervals in the form of NumPy NPY files [16]. The software supports easy and powerful post-processing with the Python module for Ithildin [17], including integration with Scientific Python [18], 2D visualization with Matplotlib [19], and 3D visualization with ParaView [20].

Ithildin also allows the recording of pseudo-electrograms (EGMs) and state variables at full numerical time resolution, as well as the tracking of filaments, which represent the instantaneous rotation axes of rotors. The software features a flexible setup for in-silico experiments, also called simulations, through a simple class-based C++ interface.

Various types of geometries are implemented, ranging from a simple 1D cable and spirals in 2D tissue to whole-heart geometry and even 4D hyperspace. The space can be partitioned to use multiple cell models in the same experiment via `Model_multi`. Realistic stimulation protocols can be added as `Stimulus` objects and may be started by a `Trigger`. Ithildin also includes a logging system with minimal impact on computation speed and various levels of verbosity.

In this paper, we provide an overview of this framework, guiding the reader through its components. Results from several in-silico experiments are presented as the main components

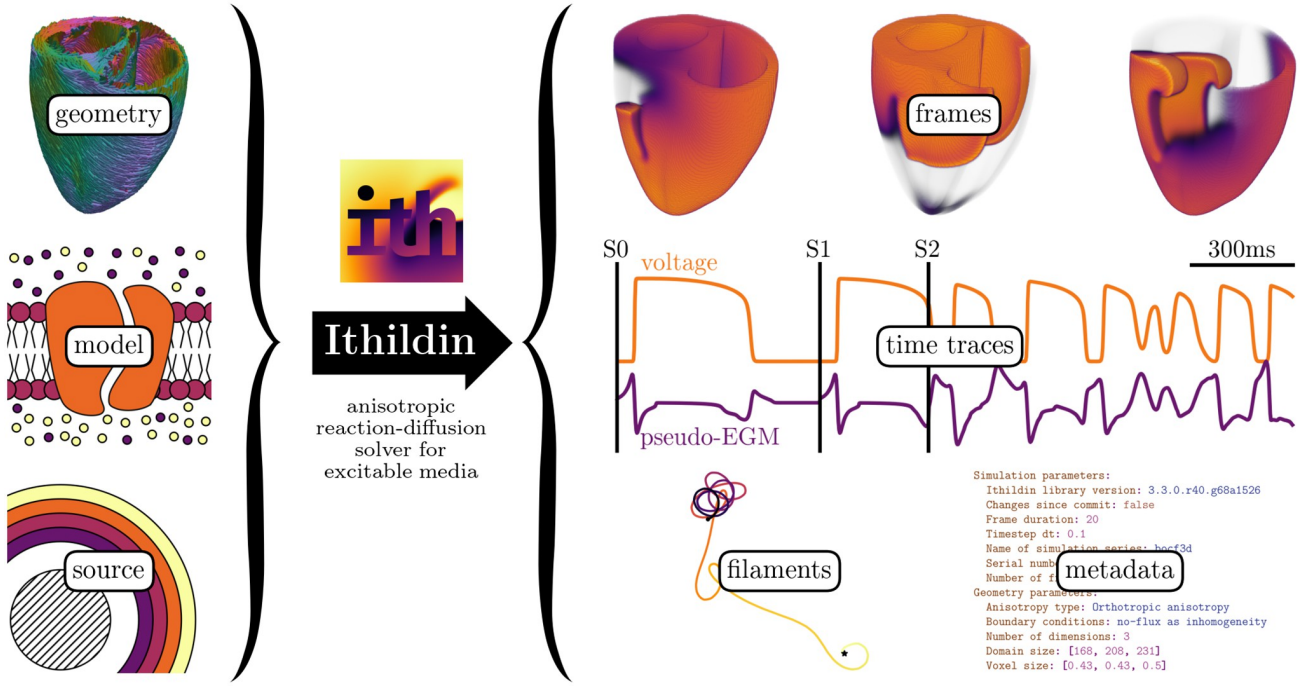

Fig 1. Ithildin can be used to solve reaction-diffusion problems in excitable media. Required inputs for the software are: the diffusion tensor and geometry of a medium—such as the heart muscle, a reaction term—the so-called model, and source terms—typically a stimulation protocol. Ithildin can then calculate the evolution of the model variables in the medium over time. During calculation, Ithildin records relevant spatio-temporal data and metadata, as well as detecting rotor cores as filaments. The data visualized here are taken from several simulations which will be discussed below.

https://doi.org/10.1371/journal.pone.0303674.g001

of Ithildin are introduced. The details of these so-called simulations are outlined towards the end of this paper in section 5, along with a tabular overview in Table 4.

2 Essential numerical methods

2.1 Reaction-diffusion system

The diffusion of electrical signals in cardiac tissue can be modelled as a reaction-diffusion system, where the diffusion tensor $\boldsymbol{D}$ represents the anisotropic properties of the medium. This tensor encapsulates the spatial orientation of fibers in the medium and the effects of inhomogeneities on signal propagation. With the local unit vectors along the fibers $\boldsymbol{e}_\mathrm{f}$, normal to fibers in the sheet plane $\boldsymbol{e}_\mathrm{s}$, and normal to both of these $\boldsymbol{e}_\times = \boldsymbol{e}_\mathrm{f} \times \boldsymbol{e}_\mathrm{s}$, forming a orthonormal basis, the fiber orientation is encoded using diffusivities $D_{\mathrm{f,s,}\times}(\boldsymbol{x})$ in each of these directions [2]:

$$\boldsymbol{D} = D_\mathrm{f}\boldsymbol{e}_\mathrm{f}\boldsymbol{e}_\mathrm{f}^{\mathrm{T}} + D_\mathrm{s}\boldsymbol{e}_\mathrm{s}\boldsymbol{e}_\mathrm{s}^{\mathrm{T}} + D_\times\boldsymbol{e}_\times\boldsymbol{e}_\times^{\mathrm{T}} \tag{1}$$

The core equation governing the evolution of the state variable vector $\underline{u}$ is the reaction-diffusion equation:

$$\partial_t \underline{u} = \underline{P}\nabla \cdot \boldsymbol{D}\nabla\underline{u} + \underline{r}(\underline{u}) \tag{2}$$

or in index notation:

$$\partial_t u_m(t,\boldsymbol{x}) = \sum_{m'} P_{mm'} \sum_n \partial_n \sum_{n'} D_{nn'}(\boldsymbol{x})\ \partial_{n'} u_{m'}(t,\boldsymbol{x}) + r_m(\underline{u};\boldsymbol{x}) \tag{3}$$

for $m, m' \in \{1, \ldots, M\}$ with the number of state variables M, and $n, n' \in \{1, \ldots, N\}$ with the number of spatial dimensions N, using the notation $\partial_n = \partial_{x_n}$ for the spatial partial derivatives.

Here, $\underline{u}$ is the state variable vector and $\underline{r}(\underline{u})$ accounts for the reaction term and is called the *model*. We refer to the first component of $\underline{u}$ as u, which for electrophysiological models is the transmembrane voltage V_m or a rescaled version of it, see also section 4.2. For two-variable models, the second component of $\underline{u}$ is often referred to as the restitution or recovery variable v. In the term representing diffusion, $\boldsymbol{D}$ is determined by the geometry of the medium and the presence of inhomogeneities, see section 4.1. The projection matrix $\underline{P}$ is typically a diagonal matrix describing whether or not a variable is diffused. For instance, only the first variable of the AP96 model [21] is diffused, such that $\underline{P} = \mathrm{diag}(1, 0)$.

Different notation is used to distinguish between vectors $\boldsymbol{x}$ and matrices $\boldsymbol{D}$ in physical space in bold font, and underlined vectors $\underline{u}$ and matrices $\underline{P}$ with respect to state variables. We use lowercase letters for vectors and uppercase for matrices. An overview of the most relevant quantities in Ithildin is given in Table 1.

Table 1. Quantities in the reaction-diffusion problem.

symbol	dimension	unit	name
N	$\in \mathbb{N}$	1	number of spatial dimensions
M	$\in \mathbb{N}$	1	number of state variables
t	$\in \mathbb{R}_+$	ms	time
$\boldsymbol{x}$	$\in \Omega \subset \mathbb{R}^N$	mm	space
$\underline{u}(t,\boldsymbol{x})$	$\in \mathbb{R}^M$	$\star$ (various units)	state variables
$\underline{r}(\boldsymbol{x},\underline{u})$	$\in \mathbb{R}^M$	$\star$/ms	reaction term, model function
$\boldsymbol{D}(\boldsymbol{x})$	$\in \mathbb{R}^{N\times N}$	mm^2/ms	diffusion tensor
$\underline{P}$	$\in \mathbb{R}^{M\times M}$	1	projectionmatrix

https://doi.org/10.1371/journal.pone.0303674.t001

Ithildin obtains approximate solutions of the reaction-diffusion equation (Eq 2) via a finite-differences approach: Time t and space $\boldsymbol{x}$ are discretized on a grid and values $\underline{u}(t, \boldsymbol{x})$ are associated with the vertices of this grid. We choose a fixed temporal resolution, the time step Δt, and constant spatial grid spacing $\Delta \boldsymbol{x}$. The values $\underline{u}(t + \Delta t, \boldsymbol{x})$ at a subsequent time-step are computed based on the previous ones, according to discretized versions of the governing equations, i.e., the reaction-diffusion equation (Eq 2), together with boundary and initial conditions.

2.2 Time integration

Starting from an initial state, the state variable vector $\underline{u}$ is integrated over time using a so-called time stepping scheme leading to an approximate solution of the reaction-diffusion system using finite differences. Ithildin implements two main stepping schemes to choose from: forward Euler and the classic Runge-Kutta method (RK4) [22, 23].

Defining $\underline{f}$ as the right hand side of the reaction-diffusion equation (Eq 2), the forward Euler method takes the form [23]:

$$\underline{u}(t + \Delta t, \boldsymbol{x}) = \underline{u}(t, \boldsymbol{x}) + \Delta t \underline{f}(t, \boldsymbol{x}; \underline{u}) + O(\Delta t^2) \tag{4}$$

This method is the default time integration scheme in Ithildin. Despite its numerical error being of order $O(\Delta t^2)$, with a sufficiently small time step, the accuracy of the Euler method is adequate for our use cases.

Due to the Courant-Friedrichs-Lewy condition (CFL), a stability criterion for the integration of the reaction-diffusion equation, Δt needs to be chosen sufficiently small [24]. Ithildin automatically chooses an appropriate time step based on the CFL condition for the different supported geometries, cf. section 4.1. For example, for the most simple implemented geometry contained in Ithildin, i.e., isotropic diffusion (see `Geometry_Iso` in section 4.1 and Table 2), the CFL condition is enforced by setting [25]:

$$\Delta t < \left[2 \max \underline{P} \sum_{n=1}^{N} \frac{D_{nn}}{\Delta x_n^2} \right]^{-1} \tag{5}$$

where D_{nn} are the diagonal components of $\boldsymbol{D}$ and $\max \underline{P}$ is the maximum value of $\underline{P}$.

Table 2. Overview of tissue geometries supported by Ithildin.

class	description	dim N	points in stencil
`Geometry_Iso`	isotropic diffusion	1..3	$2N + 1$
↳ `Geometry_ND`	isotropic diffusion in higher dimensions [31]	$N \in \mathbb{N}$	$2N + 1$
`Geometry_OrtAniso`	orthotropic diffusion	2..3	$\begin{cases} 9 & N = 2 \\ 19 & N = 3 \end{cases}$
`Surface`	2D domain with extrinsic curvature	2	9
`QuadricSurface`	quadric surface with rotated parallel fibers	2	9
↳ `Ellipsoid`	ellipsoidalsurface with rotated parallel fibers	2	9
↳ `Hyperboloid`	hyperboloidalsurface with rotated parallel fibers	2	9
↳ `Paraboloid`	paraboloidalsurface with rotated parallel fibers [32]	2	9

https://doi.org/10.1371/journal.pone.0303674.t002

Higher accuracy at the cost of more computations per time step, can be achieved with the RK4 method [23]:

$$\underline{u}(t+\Delta t,\boldsymbol{x}) = \underline{u}(t,\boldsymbol{x}) + \frac{1}{6}\Delta\underline{u}_1 + \frac{1}{3}\Delta\underline{u}_2 + \frac{1}{3}\Delta\underline{u}_3 + \frac{1}{6}\Delta\underline{u}_4 + O(\Delta t^5) \tag{6}$$

$$\Delta\underline{u}_1 := \Delta t\underline{f}(t,\boldsymbol{x};\underline{u}) \tag{7}$$

$$\Delta\underline{u}_2 := \Delta t\underline{f}\left(t+\frac{1}{2}\Delta t,\boldsymbol{x};\underline{u}+\frac{1}{2}\Delta\underline{u}_1\right) \tag{8}$$

$$\Delta\underline{u}_3 := \Delta t\underline{f}\left(t+\frac{1}{2}\Delta t,\boldsymbol{x};\underline{u}+\frac{1}{2}\Delta\underline{u}_2\right) \tag{9}$$

$$\Delta\underline{u}_4 := \Delta t\underline{f}(t+\Delta t,\boldsymbol{x};\underline{u}+\Delta\underline{u}_3) \tag{10}$$

Note that $\underline{f}$ needs to be evaluated four times for the RK4 method and only once for the Euler method. While RK4 is still an explicit method subject to instability at too large Δt, a larger Δt value than for the Euler method can typically be used.

The stepping scheme to be used can be chosen in Ithildin on a per-variable level via `Model::steppings`.

2.3 Numerical spatial derivatives

For the numerical solution of the reaction-diffusion equation (Eq 2), the spatial derivative in its right hand side must be computed, i.e., the diffusion operator $\underline{P}\nabla\cdot\boldsymbol{D}\nabla\underline{u}$. This is implemented as weighted sums of the value of $\underline{u}$ at neighboring vertices on the grid of the discretized domain. The weights for the calculation of the stencil depend on the chosen type of diffusion (section 4.1). The two main types in this software are a first order stencil, including only the nearest neighbors, and a second order stencil, including also the next to nearest neighbors.

In the simplest case (see `Geometry_Iso` in section 4.1 and Table 2), we consider isotropic and homogeneous diffusivity. The diffusion operator can then be computed via the Laplacian operator ∇^2. This is done with a 5-point stencil for the 2D case, a 7-point stencil for the 3D case, etc., see also Fig 2, panel (a). Consequently, the weights are calculated as:

$$w_n = \frac{1}{\Delta x_n} \quad \text{for } 1 \le n \le 2N \tag{11}$$

$$w_0 = -\sum_{n=1}^{2N} w_n \tag{12}$$

where N is the number of dimensions and Δx_n is the grid resolution in the direction of the neighbor corresponding to weight w_i, e.g., $\Delta x_5 = \Delta z$. Note that the indices correspond to those displayed in Fig 2.

For the more general orthotropic diffusion, the stencil for numerical differentiation includes the nearest and diagonal neighbors of a grid point, see Fig 2, panel (b). The weights for orthotropic diffusion are obtained by a combination of central differences approximations to derivatives and linear interpolation of values between two grid points. For more details, the reader is referred to the documentation of `Geometry_OrtAniso`.

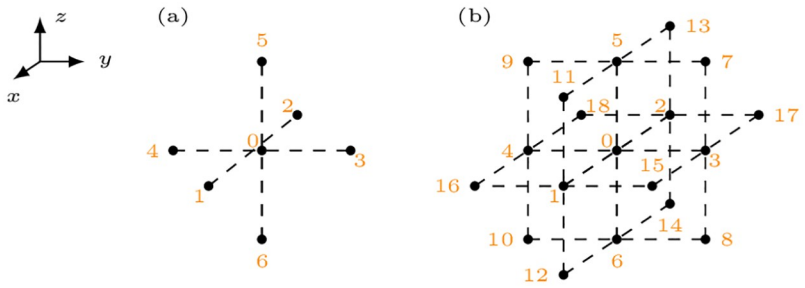

Fig 2. Stencils for isotropic diffusion (a) and orthotropic diffusion (b) with numbering of the involved grid points.

https://doi.org/10.1371/journal.pone.0303674.g002

3 Implementation

The source code of Ithildin is written in the C++ programming language, following the 2011 version of the ISO-C++ standard [8]. This was chosen to facilitate programming at a level relatively close to the hardware, but also using some of the useful data-structures that are contained in the standard template library (STL).

Ithildin is designed to run in parallel on multiple CPU cores using MPI, specifically Open-MPI [13]. Using the `mpirun` command, multiple instances, also known as processes, of the same compiled executable are started that run across different processor cores. The processes are then coordinated such that there is one so-called *manager* process, that manages a bunch of so-called *worker* processes. The manager does an equal share of the computational work, just like a worker. The only difference is that the manager distributes and directs information to be exchanged from one process to another. In Ithildin, the memory is not shared across processes, instead each process works on its share of the computational domain. We split the domain in the x-direction, such that each process is responsible for computations on a roughly equal share of vertices inside the to-be-simulated medium. Even when some parts of the domain are classified as exterior points, i.e., points on which no calculations need to be performed, they are taken into account when splitting the domain between processes. This splitting is possible because the reaction-diffusion systems to be studied with Ithildin are local, meaning that the temporal evolution at each time t and each point $\boldsymbol{x}$ in space depends only on the current state vector $\underline{u}(t, \boldsymbol{x})$ at that position and its spatial derivatives (Eq 2). Internally, a layer of so-called *ghost points* is added around each process' part of the domain such that the spatial derivatives can still be calculated in the same way as for any other point in the domain (section 2.3). The values $\underline{u}$ on these ghost points are exchanged with the neighboring processes, as coordinated by the manager process. Additional ghost points are also used to enforce Neumann boundary conditions, which is done by setting the weights for the calculation of the numerical spatial derivatives accordingly.

To run a simulation in Ithildin, the user needs to define a `main()` function that is to be called by all subprocesses. This is usually done using a C++ file calling the required components of the Ithildin library defining the `main()` function, a so-called *main file*. Ithildin can be installed as a shared library on Unix-based systems that is required by executables obtained from compiling main files. Alternatively, it is also possible to statically compile the Ithildin library with a main file into a stand-alone executable.

Besides the necessary preparations for using MPI, running a simulation using Ithildin's `Sim` class requires three main components, which are instances of three classes:

1. `Model`: the reaction term $\underline{r}(\underline{u})$, typically a cell model,

2. `Geometry`: the discretized diffusion term $\underline{P}\nabla \cdot \boldsymbol{D}\nabla\underline{u}$ for a chosen geometry of the medium, and

3. `Source`: the stimulus protocol to use as well as inhomogeneities in the medium.

In the following section 4, an overview is given for each of these classes and their derived classes. Combining all of the components, an illustrative, minimal main file can be obtained, in which a planar wave crosses the medium in positive x-direction:

```
#include "ithildin.h"

int main(int argc, char** argv){
  // 0. initialize MPI
  Mpiclass mpi(argc, argv);

  // 1. select reaction term
  Model_SmooKa model;

  // 2. select diffusion term
  vector<int> size{30, 30, 1};
  vector<float> dx{1., 1., 1.};
  Geometry_Iso geom{&mpi, size, dx, &model};

  // (set up simulation)
  Sim sim{1., 10, &mpi, &model, &geom, "example"};

  // 3. define stimulus protocol
  Source sour{&model, &geom, &mpi, &sim};
  sour.stimulate({{0}, {1.}, Shape::Rect({0, 0}, {5, 0})});

  // (run simulation)
  return sim.run(&sour);
}
```

More detailed examples for main files can be found in the S1 Appendix, where we set up the numerical examples used throughout this paper.

We consider the geometry, the model and the source to be the inputs of Ithildin, see also Fig 1. Upon running the simulation, Ithildin produces a variety of outputs, as files in easy-to-parse standardized data formats. The names of these files all begin with the so-called *stem*, consisting of a descriptive series name and a serial number, which defaults to a time-stamp. In the following, an overview of the usual output files of Ithildin by suffix appended to the stem is provided:

- `_log.yaml`: The log file contains metadata describing the setup of the simulation, as well as metadata about the conditions under which the simulation was run. This file is always output by Ithildin and is considered the central file of the results, as it points to the relevant other files that are only conditionally written during the simulation. While earlier versions of Ithildin used a non-standard format for the log files, in current versions, the YAML format is used, making it easy to parse by both: humans and machines [15].

- `_main.cpp`: A copy of the C++ code in the main file may be included in the results for reproducibility.

- `_git.diff`: If run in a Git-repository with changes since the last commit, a patch file of these changes will be included in the results.

- `_*.txyz.npy`: For each of the state variables $\underline{u}$, a so-called var file in the NumPy NPY format will be written [16]. These files contain the N+ 1-dimensional floating-point number array of the evolution of a state variable u in the whole computational grid over time. Note that the order of indices is (t, x, y, z), meaning that time is the slowest varying index and the x-axis the second slowest varying index. This order was chosen because the domain is split across processes along the x-axis, such that the processors can open and write to these files sequentially. For 2-dimensional simulations, the third spatial dimension, the z-axis, is one vertex thick, such that each var file still has four dimensions. For higher-dimensional simulations, more axes are added, leading to more dimensions in the var files.
- `_inhom.txyz.npy`: The `inhom` field, see details in section 4.3, is also stored in the NPY format, but with integer values, and only for the initial time-step. This file hence has the shape $(1, N_x, N_y, N_z)$ with N_n denoting the number of vertices in each of the spatial dimensions.
- `_hist*.csv`: Comma-separated value (CSV) files describing the temporal evolution of $\underline{u}(t, \boldsymbol{x}_s)$ at a chosen sensor position $\boldsymbol{x}_s \in \Omega \subset \mathbb{R}^N$, see details in section 4.5.
- `_egm*.csv`: CSV files containing the recorded pseudo-EGM $\Phi(t, \boldsymbol{x}_e)$ at a chosen electrode location $\boldsymbol{x}_e \in \mathbb{R}^N$, see details in section 4.6.
- `_tipdata.yaml`: This YAML file is written if filament-tracking is turned on, see also section 4.7, and contains the detected phase singularities in regular time intervals.

While we have selected these file formats to be easy to read using a wide variety of software, we have also developed the Python module for Ithildin to facilitate interacting with the results of an Ithildin simulation and converting them to a variety of file formats, for instance writing files in the extensible data model and format (XDMF) that can be used to view simulation results in ParaView [17, 20, 26, 27]. The Python module also offers post-processing and analysis methods, for instance the computation of action potential duration (APD), conduction velocity (CV), various phases, phase defect detection, functions acting on filaments and filament trajectories, and several plotting functions [17, 26].

4 Application-focused numerical methods

4.1 Diffusion term

The diffusion term in the reaction-diffusion equation (Eq 2) is stated as $\underline{P}\nabla \cdot \boldsymbol{D}\nabla\underline{u}$. The conduction in cardiac tissue and hence the diffusion is stronger along the fiber direction than normal to the fibers [2]. This is encoded in the diffusion matrix $\boldsymbol{D}$.

As an example, in Fig 3, the fiber direction is drawn on the surface of the ventricular geometry used in Sim 4. The fibers are additionally colored by their fiber helix angle [28]. The voxel-based representation of this geometry was obtained by cutting a human heart into 1 mm-thin slices, digitizing and stacking them [29, 30].

While the projection matrix $\underline{P}$ is managed by the `Model` class, see section 4.2, the diffusion matrix $\boldsymbol{D}$ and the handling of the spatial derivatives is implemented in the `Geometry` class. The most important things the `Geometry` class takes care of, are:

- Initialization of the computational grid, along with the strides and pointers, which are important for efficient computing;
- Computation of the entries of the diffusion tensor, based on the main directions of diffusion and the respective diffusion values;

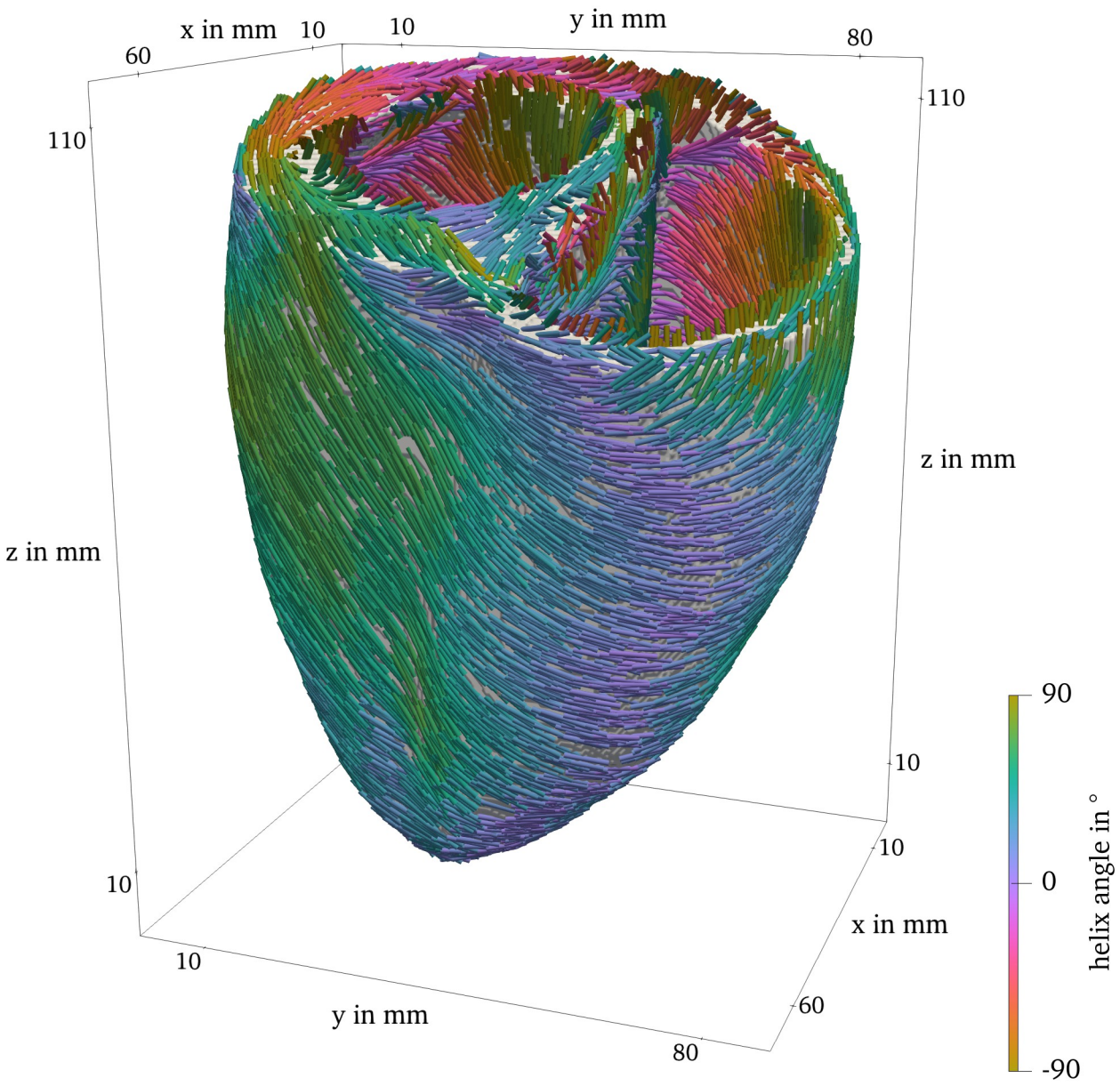

Fig 3. Ventricle geometry with fiber direction, colored by thefiber helix angle.

https://doi.org/10.1371/journal.pone.0303674.g003

- Computation and storage of the weights for the stencils of the numerical spatial derivatives (section 2.3) respecting the Neumann boundary conditions; and
- Handling the upper bound for the time step due to the CFL condition (section 2.2).

The `Geometry` class is a base class and should not be used directly to construct a computational domain. Instead, there are several subclasses, each representing a different type of diffusion or extrinsic shape. An overview of the `Geometry` subclasses is given in Table 2.

The `Geometry_ND` class is a subclass of `Geometry_Iso` and the `Ellipsoid`, `Hyperboloid`, and `Paraboloid` classes are subclasses of `QuadricSurface`.

The details on how the extrinsic and intrinsic curvature affect the diffusion tensor and hence the weights for the discretized differential operator, are discussed in the documentation of the code. There, the different ways to initialize the available domain types are given as well.

Note that some additional features are implemented in the `Geometry` class, that do not have a direct link with the diffusion term, such as inhomogeneities (section 4.3) and filament detection (section 4.7).

4.2 Reaction term

The `Model` class serves as the base class for all cardiac electrophysiology models in the Ithildin framework. It encapsulates common functions and variables used across different models. Key features of the `Model` class include:

- Implementation of the reaction term $\underline{r}(\underline{u})$ in the reaction-diffusion equation.
- Storage of model-specific metadata such as relevant citations.
- Handling of variable-related information, including their names, indices, and resting values.
- Management of the values of the projection matrix $\underline{P}$.

Derived classes extend the `Model` class to implement specific cardiac electrophysiology models in the `reactionterm` function.

An overview of the cell models that are currently included in the source code of this project is provided in Table 3. More models may be added as additional classes.

The Ithildin framework also introduces several `ModelWrapper` classes that provide additional functionality and allow for the combination of models. These wrappers enable the recording of diffusion terms, reaction terms, local activation times (LAT), and local deactivation times (LDT) as additional state variables. This is done by adding code to the reaction term calculations of the underlying model. The wrappers inherit from the base `ModelWrapper` class, which is a wrapper that leaves the model unchanged. The primary `ModelWrapper` classes are:

- `ModelWrapper_RecordDiffusion` records diffusion terms of selected variables as additional variables.

Table 3. Overview of cell models included in Ithildin.

class	description	#vars	references
`Model_1VarPoly`	one-variable polynomial model	1	
`Model_AP`	Aliev-Panfilov, continuous epsilon	2	[21]
`Model_AP2`	Aliev-Panfilov, discontinuous epsilon	2	[21]
`Model_AP3`	Aliev-Panfilov, discontinuous epsilon, simple recovery	2	[21, 33, 34]
`Model_AP4`	Aliev-Panfilov, smoothened epsilon, simple recovery	2	[21, 33, 34]
`Model_BO`	Bueno-Orovio 2008 4 var	4	[35]
`Model_Ba`	Barkley	2	[36]
`Model_FHNa`	FitzHugh-Nagumo (a), 2 var	2	[37, 38]
`Model_FHNb`	FitzHugh-Nagumo (b), 2 var	2	[37, 38]
`Model_FHNc`	FitzHugh-Nagumo (c), 2 var	2	[37, 38]
`Model_FK`	Fenton-Karma 3 var	3	[3]
`Model_Kaz`	Kazantsev PRE 2003 spiking neuron	2	[39]
`Model_LRI`	Luo-Rudy Phase I	8	[40]
`Model_MS`	Mitchell Schaeffer	2	[41]
`Model_SmooKa`	Smooth Karma by Marcotte 2017	2	[42–45]
`Model_TP06`	Ten Tusscher & Panfilov 2006	19	[4, 46]

https://doi.org/10.1371/journal.pone.0303674.t003

- `ModelWrapper_RecordReaction` records reaction terms of selected variables as additional variables.
- `ModelWrapper_RecordActivationTime` records LAT for selected variables.
- `ModelWrapper_RecordDeactivationTime` records LDT for selected variables.
- `ModelWrapper_RescaleTimeSpace` linearly rescales the wrapped model in time and space.
- `ModelWrapper_RescaleVars` linearly rescales selected state variables of the wrapped model.

The `Model_multi` class enables the combination of multiple submodels into a single model. The behavior of the combined model may vary depending on the location $\boldsymbol{x}$ and is determined by one of its submodels. Which model is to be used depends on the integer value of the `inhom` field, which is used to describe spatial inhomogeneities, such as obstacles. Inhomogeneities will be further explained in the following, cf. section 4.3. This class is particularly useful for simulating scenarios where different regions of cardiac tissue exhibit distinct behaviors.

The Ithildin C++ framework provides a structured and modular approach to modeling cardiac electrophysiology. The `Model` class serves as the base for different models, while various `ModelWrapper` classes and the `Model_multi` class offer extended functionalities for recording and combining different model aspects.

4.3 Inhomogeneities

In simulations of cardiac electrophysiology, accurately modeling the spatial properties of the cardiac tissue is essential. Inhomogeneities represent variations in the tissue's characteristics, such as its electrical conductivity or cellular properties, that influence the propagation of electrical signals.

An inhomogeneity in the Ithildin framework is a distinct region within the simulation domain with different properties compared to its surroundings. In the context of cardiac electrophysiology, these properties could correspond to variations in the electrical conductivities of cells, cellular properties, or even the absence of excitable cells altogether. Inhomogeneities are defined by an integer field called `inhom` associated with each point in the domain. Grid points with a non-zero `inhom` value are considered *interior points*, indicating that the reaction-diffusion equation needs to be solved on these points. If the selected reaction term is a `Model_multi` (see also section 4.2), for an `inhom` value of n, the nth submodel will be used for this point. For example, in Fig 4, the `inhom` field for Sim 1 is visualized. Two different cell models are used for the values 1 and 2. The points where `inhom` has the value 0, are considered *exterior points*. The set of all interior points is the physical domain Ω. At the boundary of the physical domain, Neumann boundary conditions are applied.

To incorporate inhomogeneities into the simulation, the framework provides methods to add them to the domain. These methods allow specifying the shape, location, and properties of each inhomogeneity. For example, a rectangular inhomogeneity could be added by specifying its width, height and location.

4.4 Stimulation protocols

Ithildin provides a flexible way to define stimulation protocols using the `Stimulus` and `Trigger` classes, as well as the scheduling functionality of the `Source` class.

The `Stimulus` class enables the specification of temporal and spatial characteristics of voltage-based or current-based stimuli. It allows defining which state variables should be

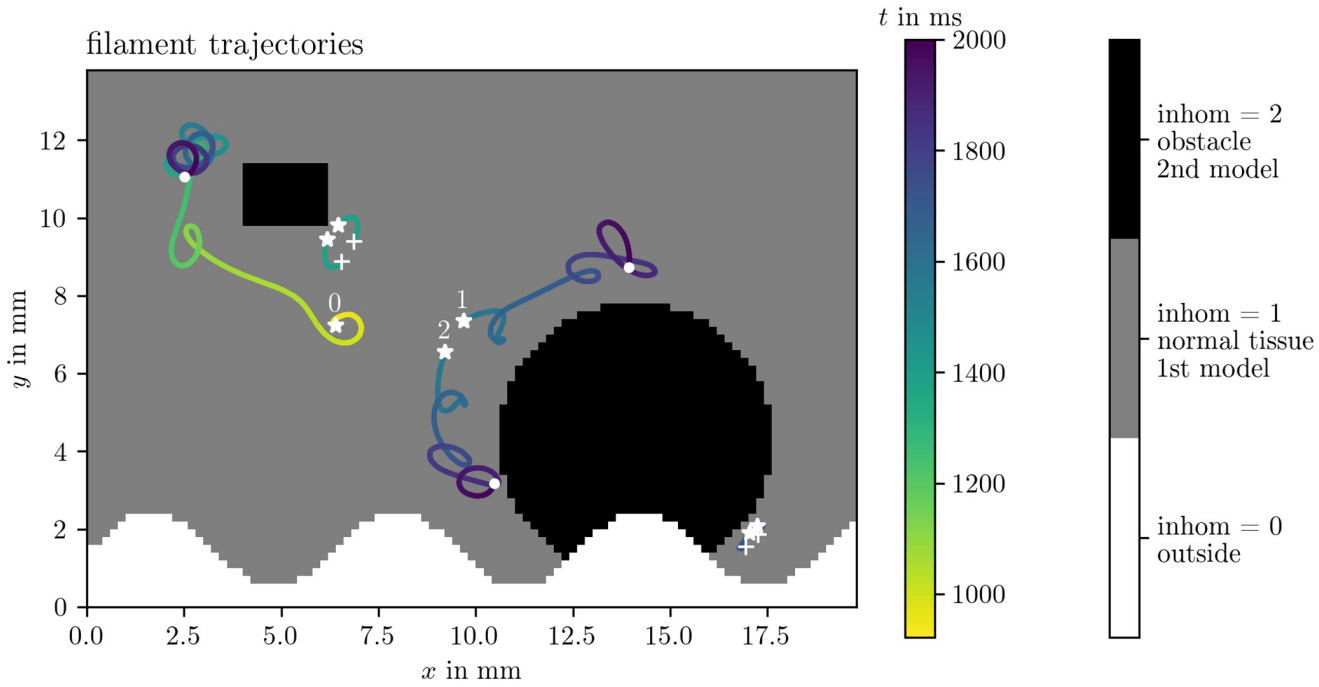

Fig 4. Inhomogeneities in Sim 1. The field `inhom` describes where unexcitable obstacles or exterior points are located with the value `inhom = 0` and which cell model is to be used inside if `inhom > 0`. This figure also visualizes the filament trajectories colored by time, cf. section 4.7. Birth of a tip is denoted by stars, their deaths by crosses, and tips that are still around at the end of the simulation by points. The three trajectories with the longest lifetime are numbered by indices 0 through 2.

https://doi.org/10.1371/journal.pone.0303674.g004

affected directly, the associated values, and whether the stimulus sets the variable directly or whether it is additive and hence behaving like a current source. Temporal modulation of stimulus strength is achieved through the `amplitude` function, while spatial constraints are managed by the `shape` function, an instance of the `Shape` class.

The `Shape` class describes a geometric shape via its characteristic function which can be used to define regions of interest in a simulation domain. The core principle is that the function evaluates a given position vector and returns a value: 1 if the position is inside the defined shape, and 0 if it is outside. However, values in the range [0, 1] may also be used to create a smooth transition. A smoothly varying characteristic function may be useful to create more-realistic stimuli that deposit current in a smooth profile. This simple yet powerful concept forms the basis for constructing intricate spatial configurations.

The `Shape` class provides several pre-defined shape functions, though additional shapes can easily be added by defining a characteristic function:

- **Ellipsoid**: Defined by radii, a center and optionally the Euler angles, this shape represents a general three-dimensional ellipsoid with the specified orientation.
- **Sphere**: A special case of an ellipsoid where all radii are equal, forming a three-dimensional sphere.
- **Ellipse in *xy*-plane**: This two-dimensional shape resembles an ellipse lying on the *xy*-plane, defined by radii and a center.
- **Cylinder along *z*-axis**: Representing a three-dimensional cylinder centered along the *z*-axis, this shape is defined by a radius and a center. In 2D, it defines a disk.

- **Rectangular cuboid**: Defining a three-dimensional region, this shape is specified by two opposing corner points, creating a cuboid. In 2D, it defines a rectangle.
- **Half plane**: A plane defined by an origin and an outward normal vector splits the three-dimensional space into a half-space. In 2D, a straight line splits the plane in a similar way.

Characteristic functions can also be loaded from files in the NPY format [16]. This feature facilitates the incorporation of custom shapes derived from external data sources.

The scheduling functionality via `Source::schedule` offers the ability to execute functions at specified points in time during the simulations. This feature greatly enhances experimental flexibility by allowing the execution of arbitrary code snippets at chosen moments during simulations. The scheduler is especially useful for introducing dynamic changes to the simulation environment, such as modifying stimuli or conditions mid-simulation.

The `Trigger` class provides a means to orchestrate actions based on specific conditions. Triggers encapsulate the decision-making process of when to execute a particular action, influenced by condition checks and coordination modes. Different coordination modes allow for the synchronization of trigger actions across multiple processes, facilitating complex simulations of activation waves. These modes are to trigger on each process individually once the condition is met, once the condition is met in a specific process, once the condition is met in any process, or once it is true in all processes.

Within the framework of cardiac electrophysiology, triggers are essential for defining stimulation protocols, e.g. for the S1S2 protocol, which is illustrated in Fig 5: After a first excitation wave passes a sensor position, a second wave is triggered behind a part of the first waveback to stimulate spiral waves.

4.5 Recording temporal evolution of variables at sensor positions

Besides the sensors for triggering stimuli, sensors in the context of this simulation framework are components that monitor and record the state variables of the simulated system at chosen positions during the simulation. We call this the *history* at a given sensor position.

These sensors are used to gather data about the behavior of the system at particular time intervals, regulated by the `sensorlag` parameter. This parameter controls the frequency at which sensor data is collected, allowing for flexibility in recording intervals. Notably, the recording frequency set by `sensorlag` need not align with the simulation's time step or the duration between frames. The data will be recorded at the first time step after the specified `sensorlag` duration. This high-resolution temporal data can be used to study individual points in the medium in detail.

The recorded data are then written to designated comma separated value (CSV) output files associated with each sensor. These files are used to store the collected data over the course of the simulation.

The first four panels of Fig 6 contain time traces of the transmembrane voltage u, the restitution variable v, the recorded value of the diffusion term, and the LAT at a given sensor position for Sim 1. The times of the three stimuli are indicated by the black vertical lines. The other panels will be explained in the subsequent sections.

4.6 Pseudo-EGMs

The EGM is a measurement of the potential generated by the charge distribution in cardiac tissue over time at a point in space, outside the tissue. In theory, this is a measurement of the extracellular potential Φ_e. However, since Ithildin is a monodomain solver, the extracellular

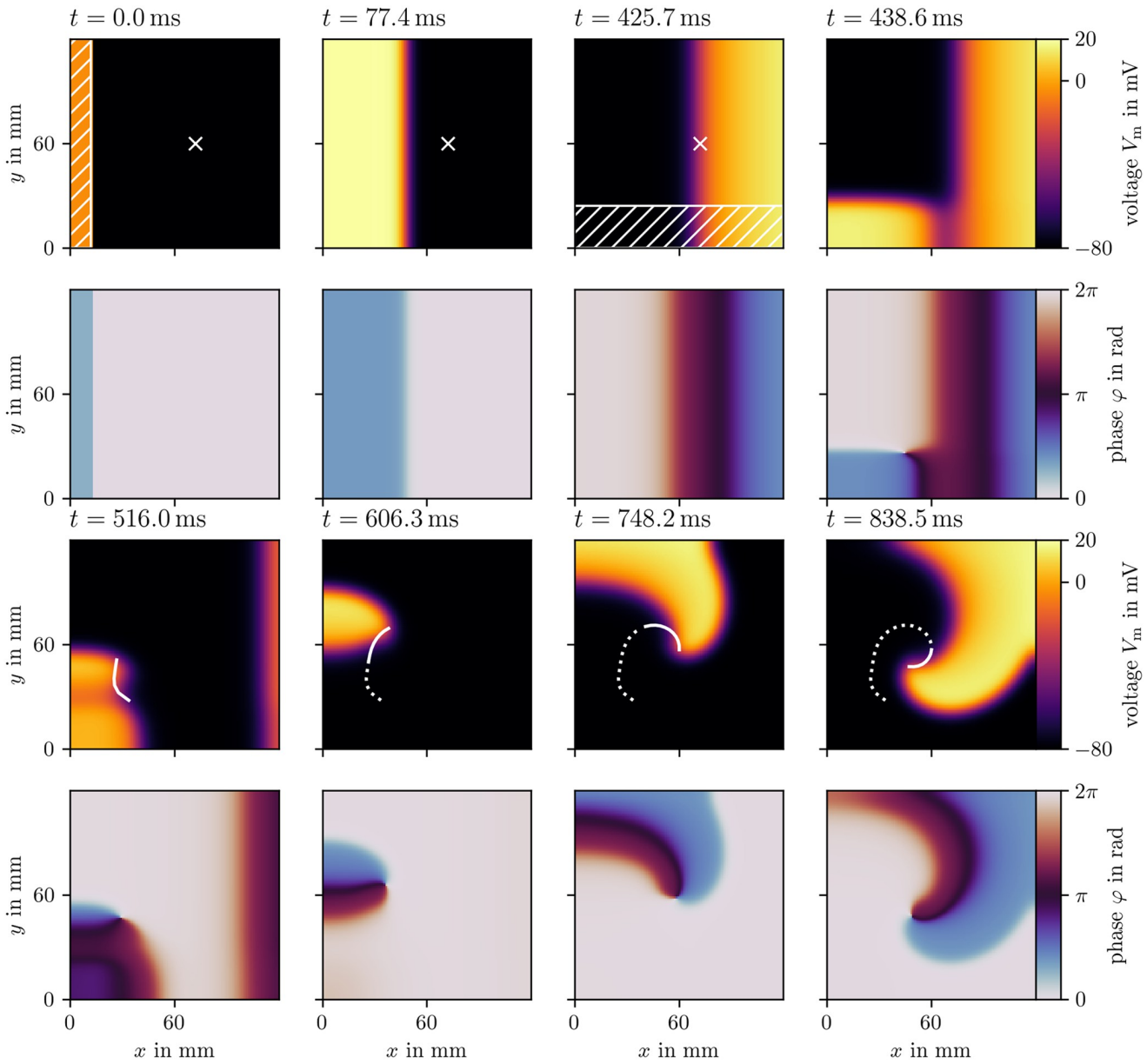

Fig 5. The S1S2 protocol illustrated for Sim 2. The first and third row display the transmembrane voltage u at selected frames in time, and the second and fourth row the state space phase [17, 47]. The first stimulus is applied in the first frame in the hatched region at the left edge. The sensor location is marked with a cross. Right after the third frame, the sensor triggers the second stimulus in the hatched region at the bottom edge. A spiral wave forms. The phase singularity at the center of the spiral is tracked as the white curve, which is dotted for the entire trajectory of the phase singularity, and solid for its trajectory since the previous frame.

https://doi.org/10.1371/journal.pone.0303674.g005

potential is not part of the model equations [2]. The code therefore calculates an approximation of the extracellular potential at points outside the mesh using the `Egm` class. To obtain this approximation, the pseudo-bidomain theory is used [48]. The approximation, referred to as a pseudo-EGM, uses the simplifications that the intracellular and extracellular conductivities are proportional, such that there is an explicit formula to calculate the extracellular potential, and that the bath conductivity is homogeneous.

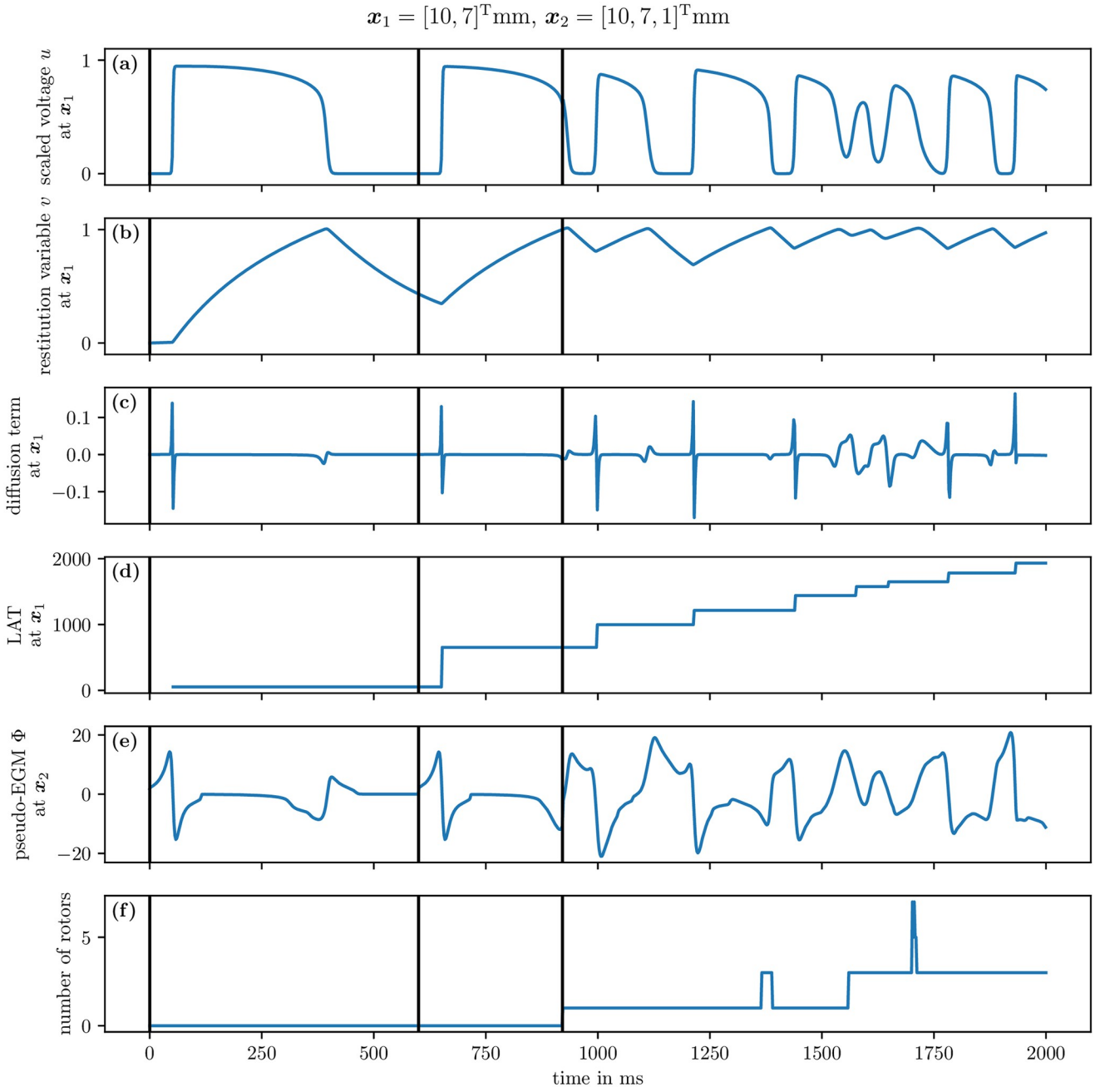

Fig 6. Extracted time traces of Sim 1. The first two panels (a, b) contain the model variables u and v, followed by data computed by model wrappers, namely the diffusion term $\nabla \cdot \boldsymbol{D}\nabla u$ (c) and the LAT (d), at an interior point, cf. section 4.5. Panel (e) contains the pseudo-EGM at an exterior point, next to the 2D domain, cf. section 4.6. The last panel (f) contains the number of rotors detected via phase singularities, cf. section 4.7. The black vertical lines indicate times at which a stimulus was applied.

https://doi.org/10.1371/journal.pone.0303674.g006

The extracellular potential Φ_e at position $\boldsymbol{x}_e$ over time is then calculated as:

$$\Phi_e(\boldsymbol{x}_e, t) = \frac{\tau_e}{4\pi} \int_\Omega \mathrm{d}\boldsymbol{x}\ \frac{\nabla \cdot \boldsymbol{D}\nabla u(\boldsymbol{x}, t)}{\|\boldsymbol{x}_e - \boldsymbol{x}\|} \tag{13}$$

where Ω denotes the computational domain, i.e., the simulated heart muscle tissue, u is a

state variable of the model representing the transmembrane potential, which is usually encoded as the first variable in the state vector $\underline{u}$. Furthermore, $\boldsymbol{D}$ is the diffusion matrix from the reaction-diffusion equation (Eq 2), and τ_e is a proportionality factor with units ms.

Essentially, the used approximation for the EGM is a convolution of the diffusion term $\nabla \cdot \boldsymbol{D}\nabla u$ for the first variable with a kernel $\|\boldsymbol{x}_e - \boldsymbol{x}\|^{-1}$. Depending on the choice of the diffusion tensor, made by the user in the `Geometry` class (section 4.1), a different prefactor of the integral is required. Hence, the prefactor $\frac{\tau_e}{4\pi}$ is user-defined and can be changed using the function `set_prefactor`.

The integral in Eq 13 is discretized and its calculation is implemented such that the additional amount of storage and number of calculations is limited as much as possible. For instance, as the required diffusion term $\nabla \cdot \boldsymbol{D}\nabla V_m$ is already computed during the forward stepping of the reaction-diffusion equation, it can be stored as an additional state variable using a `ModelWrapper_RecordDiffusion` and subsequently used in the pseudo-EGM calculation. This model wrapper is essential for the functioning of the `Egm` class and hence is a requirement when setting up a simulation with pseudo-EGM calculation.

The result is a CSV file with the pseudo-EGM data for each electrode that is defined by the user. The computed pseudo-EGM at a given position for Sim 1 is displayed in the fifth panel of Fig 6.

4.7 Filaments

Formally, filaments can be understood as a line of wave break, i.e., a line where an activation and recovery surface come together [49, 50]. The activation surface can be seen as the wavefront, while the recovery surface can be seen as the waveback. When considering an excitable system in 2D, filaments become tips, being the point of intersection between the activation and recovery curve. Since a point cannot be excited and recovering at the same time, points on a filament are also called *phase singularities*.

Using this definition of filament points, detection algorithms have been designed. Our code relies on the one described by Fenton *et al.* [3]. Additionally, this algorithm has been extended to grids in any dimension, where the generalization of filaments are called *superfilaments* [31].

The filament point detection algorithm is included in the `Geometry` class. Its goal is to compute the points where the wavefront and waveback meet. While looping over all coordinate planes and grid points, it is checked whether there is an intersection of isolines in the adjacent voxel faces of a grid point. The location of the filament point is then estimated by bilinear interpolation. An illustrative sketch of this method is given in Fig 7.

The last panel in Fig 6 displays the number of rotors over time for Sim 1, which are found via filament detection. It can be seen that at the S2 stimulus, a single rotor is formed, and subsequently pairs of rotors as figure-of-eight spiral pairs.

More details of this process can be seen in Fig 4 which shows the trajectories of these filaments in Sim 1 tracked using the Python module for Ithildin. The trajectories are colored by time and the formation of a new tip is denoted by stars and their decay by crosses. Tips that still persist at the final frame of the simulation are indicated by points. The tip trajectory denoted with index 0 is formed by the S1S2 protocol and meanders around the medium. It persists until the end of the simulation. The figure-of-eight spiral wave pair denoted by indices 1 and 2 is formed by a conduction block breaking up. The spiral tip with index 2 runs into the boundary to disintegrate there.

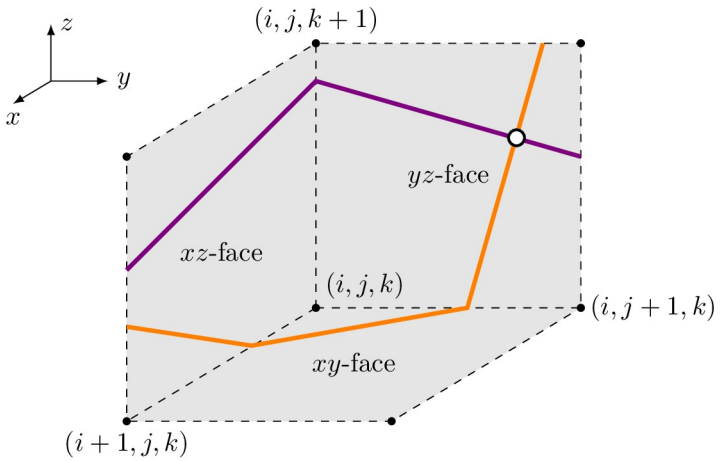

Fig 7. Illustration of the filament point tracking algorithm. Each coordinate plane adjacent to a grid point (i, j, k) is checked for an intersection of two surfaces (purple and orange lines) which are usually isosurfaces of state variables in $\underline{u}$. In this example, a filament point (white dot) will be found in the yz-face.

https://doi.org/10.1371/journal.pone.0303674.g007

4.8 Phase defects

While a phase singularity can be seen as points where all phases meet—in mathematics called a *pole*, a phase defect is a point at which there is a discrete jump from one phase value to another in an otherwise continuously varying phase. While phase defects are well known in physics, they were only recently identified in excitable media, within linear-core rotors and conduction block regions [47, 51]. Phase defects are lines in 2D and surfaces in 3D.

In Ithildin, the phase defect detection is done with its Python module. For instance, during simulation, Ithildin can record the local activation times (LAT) which can then be used to compute the activation time phase in post-processing [17, 47]. A variety of methods exist to then localize the phase defect [17, 51].

Two examples for phase defect detection in Sim 1 are given in Figs 8 and 9. Both display four frames over time of the transmembrane voltage V_m, the activation time phase φ, and the phase defect ϱ computed via the cosine method [17, 51]. In Fig 8, the phase defect of a single spiral wave is tracked. It can be seen that the phase defect extends due to conduction block such that the spiral wave moves across the domain. In Fig 9, the break-up of a conduction block line into a figure-of-eight spiral wave pair can be seen. The conduction block line is a phase defect of zero topological charge [52–54] which, in this case, reaches a critical length breaking apart into two oppositely charged spiral waves with much shorter phase defect lines.

In Fig 10, we present the final frame of Sim 4 in ventricular geometry, visualized with ParaView [20]. It is colored by the normalized transmembrane voltage. Both, the classical tip and the phase defect surface are visualized. The spiral waves revolve around the phase defect surfaces.

4.9 Further documentation

More complete documentation of Ithildin can be generated with Doxygen [14] and can be also found online, see the data availability statement for details. The documentation contains detailed information on how to get started installing and working with Ithildin.

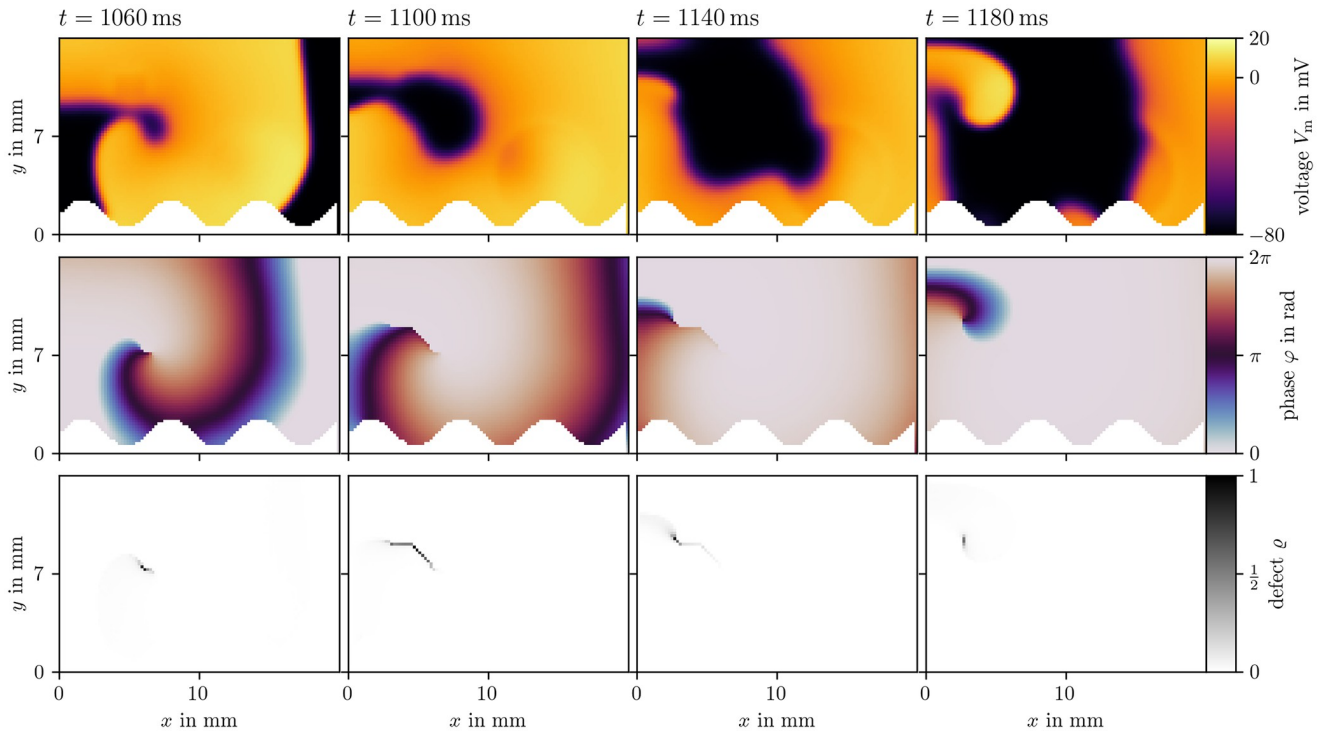

Fig 8. Phase defect detection for a single meandering spiral in Sim 1. By recording the activation times of the transmembrane voltage V_m (top row), the activation time phase φ [17, 51] can be computed (middle row). Where this phase is discontinuous, a phase defect is localized. This is visualized as the phase defect density ϱ (bottom row). In these four frames, a single rotor is tracked shortly after its formation. Due to conduction block, which can be seen as an extended phase defect line, the spiral moves through the medium. Afterwards, the spiral remains mostly stationary, leading to a shorter phase defect line.

https://doi.org/10.1371/journal.pone.0303674.g008

5 Results

Ithildin is used for numerical experiments in various use cases. For instance, it was used to study the structure of the core of rotors emerging in in-silico cardiac-electrophysiology cell models, leading to the description of phase defect lines [17, 47, 54, 55]. Also, higher-dimensional rotors waves were simulated and the emerging super-filaments were detected using Ithildin [31]. In the creation of novel data-driven cell models using state space expansion, Ithildin was used to generate synthetic training data sets [26].

Five simulations were conducted for this paper to illustrate the features of Ithildin. An overview of the simulations is given in Table 4, along with references to the figures that were generated with these data sets while details on their simulation setup can be found in their C++ code in the S1 Appendix.

5.1 Cardiac electrophysiology benchmark

Niederer *et al.* proposed a benchmark problem that is now used by the in-silico modelling community to validate and compare cardiac electrophysiology solvers [56, 57]. Ithildin passes the benchmark as implemented in Sim 5. In the benchmark problem an excitation wave travels through a cuboid-shaped medium following the cell model by Ten Tusscher and Panfilov (2006) [46]. In Fig 11, we present the results from the benchmark in a similar way as in the original publication introducing the benchmark [56]: Point $P_1 = [0, 0, 0]^T$ is the corner of the cuboid where the stimulus is applied and $P_8 = [20, 7, 3]^T$ mm the furthest-away opposite

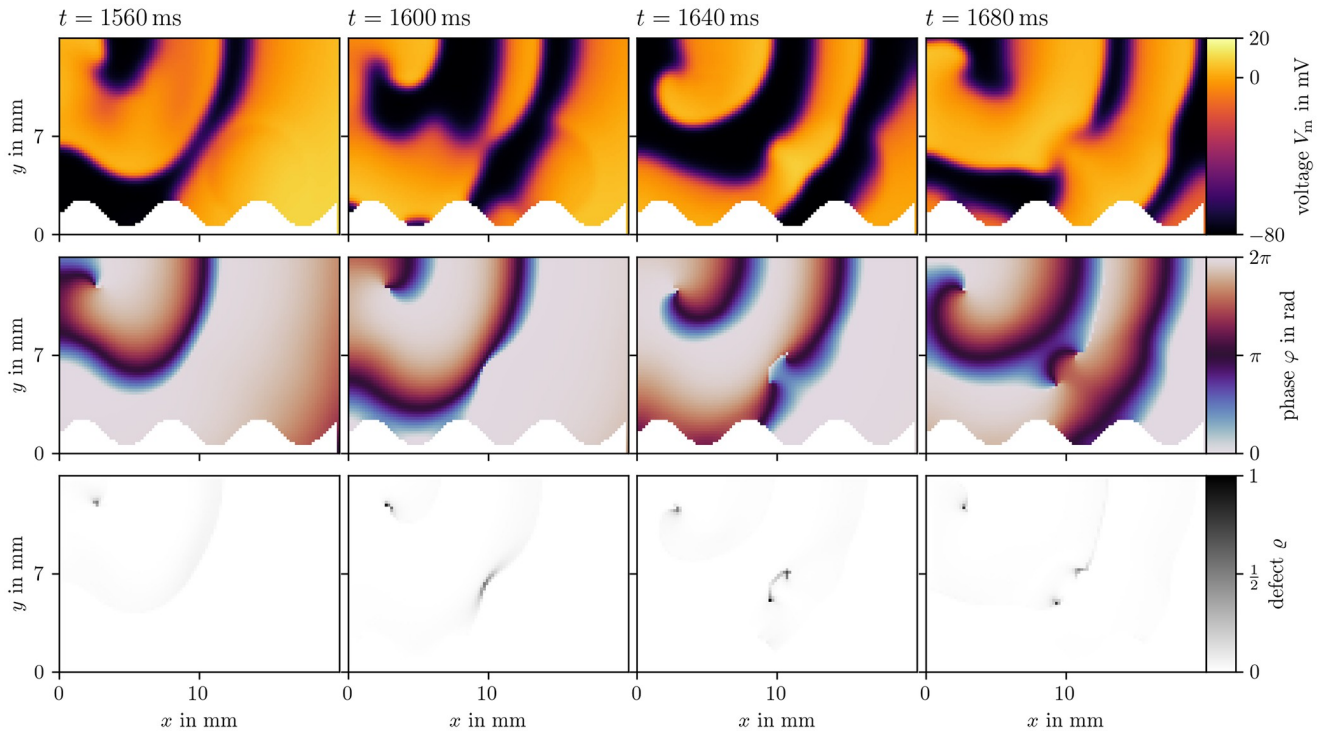

Fig 9. Phase defect detection during figure-of-eight spiral wave pair creation in Sim 1. Visualization in the same style as Fig 8. A long conduction block line breaks apart into two rotors.

https://doi.org/10.1371/journal.pone.0303674.g009

corner. Consider the plane going through P_1 and P_8 as well as $P_{10} = [0, 7, 1.5]^T$ mm. We call the distance along the long axis of the plane slicing through the cubioid ξ_1 and the short axis ξ_2. In the panels (a), the LAT on this plane is shown for the benchmark simulation at high and low resolution in space, $\Delta x = 0.1$ mm and $\Delta x = 0.5$ mm respectively, at the same temporal resolution $\Delta t = 0.005$ ms. Panel (b) also shows the LAT on the line from P_1 to P_8 for the different spatial resolutions $\Delta x \in \{0.1\text{ mm}, 0.2\text{ mm}, 0.5\text{ mm}\}$ at the same temporal resolution. In panel (c), the LAT value at P_8 is compared across all the combinations of spatial and temporal resolution. Just like for most other solvers which are compared in the benchmark paper, it can be seen that at the coarsest spatial resolution, the excitation wave is slowed down significantly in the transversal direction [56]. Similarly to the other finite-difference solvers, the simulation fails at $\Delta t = 0.05$ mm and $\Delta x = 0.1$ mm [56], due to numerical instability. When a check of the CFL condition is enabled (Eq 5), Ithildin suggests to lower Δt to 0.03 ms given this spatial resolution, leading to a simulation at which no instability is observed.

Panel (d) of Fig 11 shows the speed-up in computation time from parallelization by comparing the computation time for Sim 5 at $\Delta x = 0.2$ mm and $\Delta t = 0.01$ ms on an 8-core Intel i7–10875H processor using 1, 2, 4, and 8 processes. In the double-logarithmic plot, it can be seen with linear regression that the computational speed increases almost linearly going from one to four processes, $t_{\text{sim}} \propto N_{\text{proc}}^{-p}$ with $p \approx 1$. Going to eight processes leads to diminishing returns, resulting in a smaller speed-up in computational time, as the overhead due to the boundary-exchange between processes grows. On this system using eight processes, at $\Delta x = 0.2$ mm and $\Delta t = 0.01$ ms, solving the benchmark problem takes 65.733 s of computation time in Ithildin, while it takes 611.231 s in cbcbeat [58]. For both codes, we have turned off output to the disk and included the initial setup of the problem, such as compilation and memory allocation.

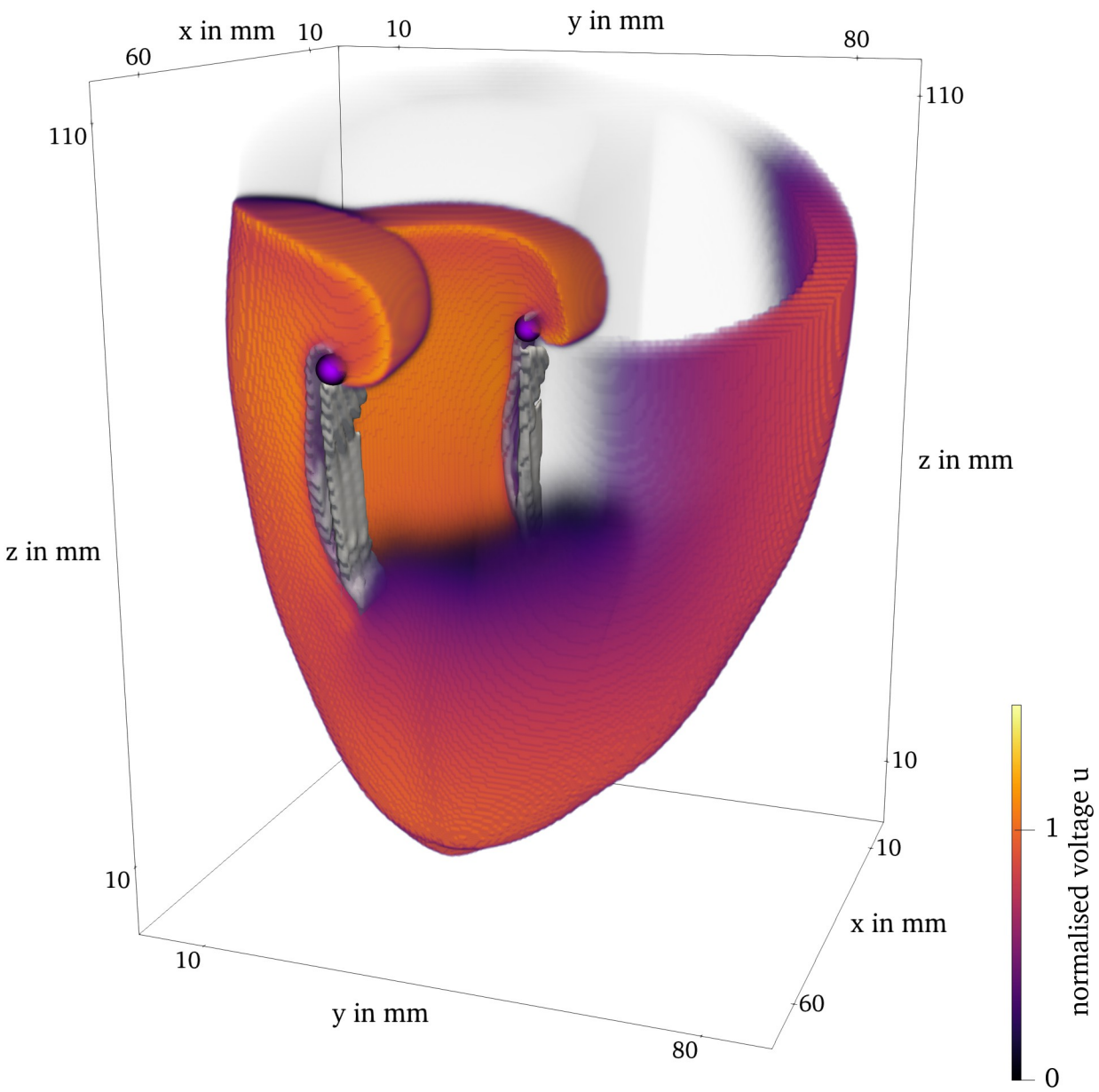

Fig 10. Final frame of Sim 4 of the BOCF model in ventricle geometry. The classical filament is plotted as the purple lines. The phase defect surface is contained in the gray contour.

https://doi.org/10.1371/journal.pone.0303674.g010

6 Discussion

The Ithildin framework is a tool for the simulation of cardiac electrophysiology. The code offers a number of assets that allow for simulations targeting a variety of phenomena. For example, there are many instances available to set the local anisotropy of the myocardium via the `Geometry` class. Please refer to the example presented in section 4.1 for further details. The `Geometry` class also allows for the simulation and filament tracking in a domain with an arbitrary number of spatial dimensions [17, 31, 47, 54, 55].

These include the ability to define inhomogeneities, which can be used to model domains of any shape and with any kind of obstacles. This is used in Sim 1 and Sim 4. Furthermore, the

Table 4. Overview of the numerical simulations used in this paper.

simulation	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5
cell model	`SmooKa`	`AP`	`BO`	`BO`	`TP06`
parameter set	default	default	EPI	EPI	EPI
references	[42–45]	[21]	[35]	[35]	[4, 46]
dimensions	2	2	2	3	3
anisotropy type	isotropic	isotropic	isotropic	orthotropic	orthotropic
diffusivity P_{uu}	0.031	1.6	0.12	0.12	0.1
ratio $D_{\parallel}/D_{\perp}$	1.0	1.0	1.0	4.0	7.6
inhomogeneities	sphere, rectangle, waves	none	none	biventricular geometry	none
grid size	100×70	120×120	450×450	168×208×231	various
spacing [mm]	0.2, 0.2	1, 1	0.3, 0.3	0.43, 0.43, 0.5	various
time step [ms]	0.1	0.1	0.1	0.1	various
stim. times [ms]	0, 600, 921	0, 426	0, 394.1	0	0
duration [ms]	2000.1	838.6	680.1	600.1	150.0
used in	Figs 1, 4, 6, 8 and 9	Fig 5	Sim 4	Figs 1, 3 and 10	Fig 11

https://doi.org/10.1371/journal.pone.0303674.t004

user can define a variety of stimulation protocols, of which some are showcased in the example simulations. We also note the following options: EGM calculation, the possibility of recording the temporal evolution of variables at sensor positions, and the Python module for post-processing and analysis.

It is evident that Ithildin represents but one instance of software designed for the simulation of cardiac electrophysiology. Another finite-differences solver is BeatBox [59], which also relies on domain splitting. A non-exhaustive list of examples of established simulation software based on the finite element method (FEM) are Chaste [60], openCARP [61], lifex-ep [62], cbcbeat [58], GEMS [63], CEPS [64], and simcardems [65]. Besides the methods of finite differences, elements, and volumes, approaches from computational fluid dynamics such as the lattice Boltzmann method may be used [57]. Several of these packages have more advanced features than our software. In addition, some have broader applications than cardiac electrophysiology. More software can be found in [56].

Limitations of our software are the fact that, in the context of cardiac electrophysiology, Ithildin only supports the monodomain model as it is a reaction-diffusion problem conforming with Eq 2, and the fact that the domain decomposition happens solely in one coordinate direction. Further limitations are that Ithildin is fundamentally voxel-based and hence does not support tetrahedral meshes, and that only Neumann boundary conditions are implemented. Additionally, it is only maintained by a small research team. This paper serves to enhance the visibility of the software and to invite fellow cardiac modelers to use and contribute to the project. We believe that the GitLab environment is an effective medium for facilitating interaction between users, identifying issues and suggesting improvements.

Our work with Ithildin has demonstrated its ability to study complex wave patterns in cardiac tissue [17, 26, 31, 47, 54, 55], but it is important to acknowledge the limitations of the fully explicit numerical scheme used: High resolution in time and space may be required for numerical stability. Nevertheless, there are use cases where Ithildin is practical in the context of clinical applications. For example, it can be used to study rotor waves and their implications for cardiac function, which have important implications for diagnosing and treating cardiac diseases. Additionally, Ithildin can be used to develop new models or simulations that are tailored to specific experimental conditions or clinical scenarios.

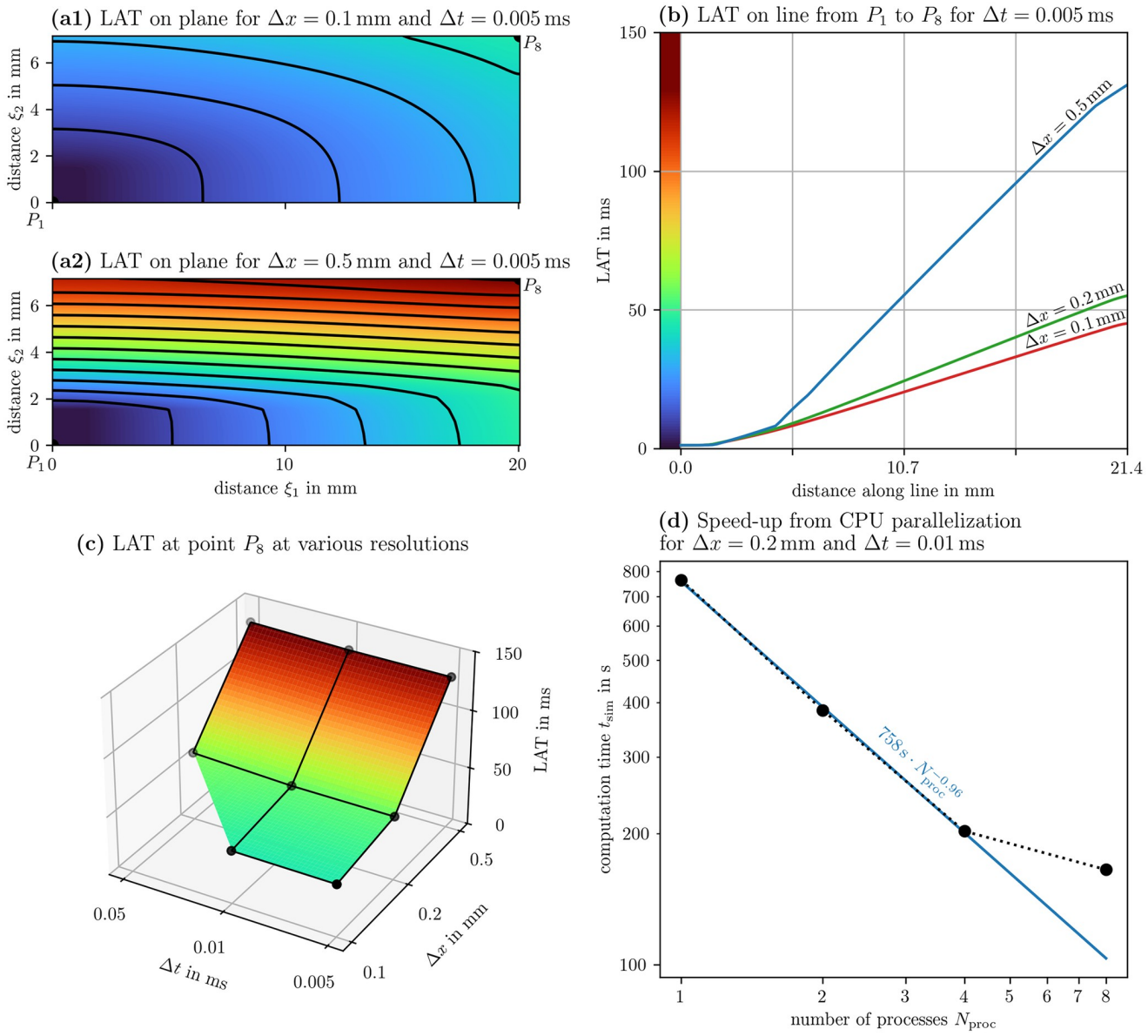

Fig 11. Results of the cardiac electrophysiology benchmark. The benchmark was proposed by Niederer *et al.* (2011) [56] and is implemented in Sim 5. To ease comparisons with the paper introducing the benchmark, we present our results in a similar style. In panels (a)-(c), we present the LAT in the medium at different spatial and temporal resolutions. Coloring according to LAT is consistent across these panels. In panel (d), we measure the computational speed of Ithildin at different number of used processes. A linear fit is shown in the double-logarithmic plot which excludes the data point at $N_{\mathrm{proc}} = 8$.

https://doi.org/10.1371/journal.pone.0303674.g011

7 Conclusion

In this work, we introduced Ithildin, an open-source library that allows for numerical simulation and analysis of rotor waves. We demonstrated the versatility of Ithildin through a series of simulations, including spiral break-up in the Smooth-Karma model, the S1S2 protocol in the AP96 model, and 2D and 3D spiral waves in the BOCF model in ventricular geometry.

Our simulations highlighted several key features of Ithildin, such as the different implemented geometries and reaction terms, inhomogeneities, and stimuli, as well as recording data such as the pseudo-EGM or filament trajectories. These findings contribute to the growing

understanding of rotor waves in cardiac electrophysiology and have the potential to inform future experimental and theoretical studies.

Overall, our work demonstrates the power of Ithildin as a tool for studying complex wave patterns in cardiac tissue. We hope that this library will be useful to researchers seeking to better understand the dynamics of rotor waves and their implications for cardiac function.

Supporting information

S1 Appendix.
(PDF)

Acknowledgments

We are grateful to Daniël A. Pijnappels, Antoine A.F. de Vries and our other collaborators at the LUMC, as well as Louise Arno, Lore Leenknegt, and Nathan Dermul at KU Leuven in Kortrijk for useful feedback and discussions.

While no generative AI has been used to write the Ithildin source code, generative AI has been used by the authors to aid in the writing of the text, specifically the Large Language Model (LLM) implementations GitHub Copilot, ChatGPT using GPT-3.5 and GPT-4o, as well as Mistral 0.2 via the Ollama software. The authors confirm that they have followed the current ethical publishing practices of PLOS One.

Author Contributions

Conceptualization: Desmond Kabus, Hans Dierckx.

Data curation: Desmond Kabus, Marie Cloet.

Formal analysis: Desmond Kabus, Marie Cloet.

Funding acquisition: Hans Dierckx.

Investigation: Desmond Kabus, Marie Cloet.

Methodology: Desmond Kabus, Marie Cloet.

Project administration: Desmond Kabus, Hans Dierckx.

Resources: Hans Dierckx.

Software: Desmond Kabus, Marie Cloet, Christian Zemlin, Olivier Bernus, Hans Dierckx.

Supervision: Hans Dierckx.

Validation: Desmond Kabus, Marie Cloet.

Visualization: Desmond Kabus, Marie Cloet.

Writing – original draft: Desmond Kabus, Marie Cloet.

Writing – review & editing: Desmond Kabus, Marie Cloet, Hans Dierckx.

References

1. Tolkien JRR. The Lord of the Rings: The Fellowship of the Ring. Allen & Unwin; 1954.
2. Clayton RH, Bernus O, Cherry EM, Dierckx H, Fenton FH, Mirabella L, et al. Models of cardiac tissue electrophysiology: Progress, challenges and open questions. Progress in Biophysics and Molecular Biology. 2011; 104(1-3):22–48. https://doi.org/10.1016/j.pbiomolbio.2010.05.008 PMID: 20553746

3. Fenton F, Karma A. Vortex dynamics in three-dimensional continuous myocardium with fiber rotation: Filament instability and fibrillation. Chaos: An Interdisciplinary Journal of Nonlinear Science. 1998; 8 (1):20–47. https://doi.org/10.1063/1.166311
4. Niederer SA, Lumens J, Trayanova NA. Computational models in cardiology. Nature Reviews Cardiology. 2019; 16(2):100–111. https://doi.org/10.1038/s41569-018-0104-y PMID: 30361497
5. Gillette K, Gsell MAF, Prassl AJ, Karabelas E, Reiter U, Reiter G, et al. A Framework for the generation of digital twins of cardiac electrophysiology from clinical 12-leads ECGs. Medical Image Analysis. 2021; 71:102080. https://doi.org/10.1016/j.media.2021.102080 PMID: 33975097
6. Trayanova Natalia A, Doshi Ashish N, Prakosa Adityo. How personalized heart modeling can help treatment of lethal arrhythmias: A focus on ventricular tachycardia ablation strategies in post-infarction patients. WIREs Systems Biology and Medicine. 2020; 12(3). https://doi.org/10.1002/wsbm.1477 PMID: 31917524
7. Koopsen T, Gerrits W, van Osta N, van Loon T, Wouters P, Prinzen FW, et al. Virtual pacing of a patient's digital twin to predict left ventricular reverse remodelling after cardiac resynchronization therapy. Europace. 2024; 26(1):euae009. https://doi.org/10.1093/europace/euae009
8. ISO. ISO/IEC 14882:2011: Information technology—Programming languages—C++. 3rd ed. Geneva, Switzerland: International Organization for Standardization; 2011.
9. ISO. ISO/IEC 14882:2014: Information technology—Programming languages—C++. 4th ed. Geneva, Switzerland: International Organization for Standardization; 2014.
10. ISO. ISO/IEC 14882:2017: Programming languages—C++. 5th ed. Geneva, Switzerland: International Organization for Standardization; 2017.
11. ISO. ISO/IEC 14882:2020: Programming languages—C++. Sixth ed. Geneva, Switzerland: International Organization for Standardization; 2020.
12. ISO. ISO/IEC 14882:2023: Programming languages—C++. Seventh ed. Geneva, Switzerland: International Organization for Standardization; 2023.
13. Graham RL, Shipman GM, Barrett BW, Castain RH, Bosilca G, Lumsdaine A. Open MPI: A high-performance, heterogeneous MPI. In: 2006 IEEE International Conference on Cluster Computing. IEEE; 2006. p. 1–9. Available from: https://doi.org/10.1109/CLUSTR.2006.311904.
14. van Heesch D. Doxygen 1.9.7; 2023. Available from: https://www.doxygen.nl.
15. döt Net I, Müller T, Antoniou P, Aro E, Smith T, Evans CC, et al. YAML Ain't Markup Language, Revision 1.2.2; 2023. Available from: https://yaml.org/spec/1.2.2/.
16. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature. 2020; 585(7825):357–362. https://doi.org/10.1038/s41586-020-2649-2 PMID: 32939066
17. Kabus D, Arno L, Leenknegt L, Panfilov AV, Dierckx H. Numerical methods for the detection of phase defect structures in excitable media. PLOS ONE. 2022; 17(7):1–31. https://doi.org/10.1371/journal.pone.0271351 PMID: 35819963
18. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods. 2020; 17:261–272. https://doi.org/10.1038/s41592-019-0686-2 PMID: 32015543
19. Hunter JD. Matplotlib: A 2D graphics environment. Computing in Science & Engineering. 2007; 9 (3):90–95. https://doi.org/10.1109/MCSE.2007.55
20. Ahrens J, Geveci B, Law C, Hansen C, Johnson C. 36-paraview: An end-user tool for large-data visualization. The visualization handbook. 2005; 717:50038–1. https://doi.org/10.1016/b978-012387582-2/50038-1
21. Aliev RR, Panfilov AV. A simple two-variable model of cardiac excitation. Chaos, Solitons & Fractals. 1996; 7(3):293–301. https://doi.org/10.1016/0960-0779(95)00089-5
22. Euler L. Institutiones calculi integralis. vol. 4. Academia Imperialis Scientiarum; 1794.
23. Press WH. Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press; 2007.
24. Courant R, Friedrichs K, Lewy H. Über die partiellen Differenzengleichungen der mathematischen Physik. Mathematische Annalen. 1928; 100(1):32–74. https://doi.org/10.1007/BF01448839
25. Li N, Steiner J, Tang S. Convergence and stability analysis of an explicit finite difference method for 2-dimensional reaction-diffusion equations. The Journal of the Australian Mathematical Society Series B Applied Mathematics. 1994; 36(2):234–241. https://doi.org/10.1017/S0334270000010377
26. Kabus D, De Coster T, de Vries AA, Pijnappels DA, Dierckx H. Fast creation of data-driven low-order predictive cardiac tissue excitation models from recorded activation patterns. Computers in Biology and Medicine. 2024; p. 107949. https://doi.org/10.1016/j.compbiomed.2024.107949 PMID: 38199206

27. Mark E, et al. Enhancements to the extensible data model and format (XDMF). In: 2007 DoD High Performance Computing Modernization Program Users Group Conference. IEEE; 2007. p. 322–327. Available from: https://doi.org/10.1109/HPCMP-UGC.2007.30.

28. Streeter DD, Spotnitz HM, Patel DP, Ross J, Sonnenblick EH. Fiber Orientation in the Canine Left Ventricle during Diastole and Systole. Circulation Research. 1969; 24(3):339–347. https://doi.org/10.1161/01.RES.24.3.339 PMID: 5766515

29. Hren R. A realistic model of the human ventricular myocardium: Application to the study of ectopic activation [PhD thesis]. Dalhousie University; 1996. Available from: https://dalspace.library.dal.ca/handle/10222/55139.

30. Ten Tusscher KHWJ, Hren R, Panfilov AV. Organization of Ventricular Fibrillation in the Human Heart. Circulation Research. 2007; 100(12). https://doi.org/10.1161/CIRCRESAHA.107.150730 PMID: 17540975

31. Cloet M, Arno L, Kabus D, Van der Veken J, Panfilov AV, Dierckx H. Scroll Waves and Filaments in excitable Media of higher spatial Dimension. Physical Review Letters. 2023; 131(20):208401. https://doi.org/10.1103/PhysRevLett.131.208401 PMID: 38039450

32. Dierckx H, Brisard E, Verschelde H, Panfilov AV. Drift Laws for Spiral Waves on Curved Anisotropic Surfaces. Physical Review E. 2013; 88(1):012908. https://doi.org/10.1103/PhysRevE.88.012908 PMID: 23944539

33. Pravdin S, Dierckx H, Markhasin VS, Panfilov AV. Drift of scroll wave filaments in an anisotropic model of the left ventricle of the human heart. BioMed research international. 2015; 2015. https://doi.org/10.1155/2015/389830 PMID: 26539486

34. Dierckx H, Arens S, Li BW, Weise LD, Panfilov AV. A theory for spiral wave drift in reaction-diffusion-mechanics systems. New Journal of Physics. 2015; 17(4):043055. https://doi.org/10.1088/1367-2630/17/4/043055

35. Bueno-Orovio A, Cherry EM, Fenton FH. Minimal model for human ventricular action potentials in tissue. Journal of theoretical biology. 2008; 253(3):544–560. https://doi.org/10.1016/j.jtbi.2008.03.029 PMID: 18495166

36. Barkley D. A model for fast computer simulation of waves in excitable media. Physica D: Nonlinear Phenomena. 1991; 49(1-2):61–70. https://doi.org/10.1016/0167-2789(91)90194-E

37. FitzHugh R. Impulses and physiological states in theoretical models of nerve membrane. Biophysical journal. 1961; 1(6):445–466. https://doi.org/10.1016/s0006-3495(61)86902-6 PMID: 19431309

38. Nagumo J, Arimoto S, Yoshizawa S. An active pulse transmission line simulating nerve axon. Proceedings of the IRE. 1962; 50(10):2061–2070. https://doi.org/10.1109/JRPROC.1962.288235

39. Kazantsev V, Nekorkin V, Binczak S, Bilbault J. Spiking patterns emerging from wave instabilities in a one-dimensional neural lattice. Physical Review E. 2003; 68(1):017201. https://doi.org/10.1103/PhysRevE.68.017201

40. Luo Ch, Rudy Y. A dynamic model of the cardiac ventricular action potential. I. Simulations of ionic currents and concentration changes. Circulation research. 1994; 74(6):1071–1096. https://doi.org/10.1161/01.RES.74.6.1071 PMID: 7514509

41. Mitchell C. A Two-Current Model for the Dynamics of Cardiac Membrane. Bulletin of Mathematical Biology. 2003; 65(5):767–793. https://doi.org/10.1016/S0092-8240(03)00041-7 PMID: 12909250

42. Marcotte CD, Grigoriev RO. Dynamical mechanism of atrial fibrillation: A topological approach. Chaos: An Interdisciplinary Journal of Nonlinear Science. 2017; 27(9):093936. https://doi.org/10.1063/1.5003259 PMID: 28964130

43. Byrne G, Marcotte CD, Grigoriev RO. Exact coherent structures and chaotic dynamics in a model of cardiac tissue. Chaos: An Interdisciplinary Journal of Nonlinear Science. 2015; 25(3):033108. https://doi.org/10.1063/1.4915143

44. Karma A. Spiral breakup in model equations of action potential propagation in cardiac tissue. Physical review letters. 1993; 71(7):1103. https://doi.org/10.1103/PhysRevLett.71.1103 PMID: 10055449

45. Karma A. Electrical alternans and spiral wave breakup in cardiac tissue. Chaos: An Interdisciplinary Journal of Nonlinear Science. 1994; 4(3):461–472. https://doi.org/10.1063/1.166024 PMID: 12780121

46. Ten Tusscher KH, Panfilov AV. Alternans and spiral breakup in a human ventricular tissue model. American Journal of Physiology-Heart and Circulatory Physiology. 2006; 291(3):H1088–H1100. https://doi.org/10.1152/ajpheart.00109.2006 PMID: 16565318

47. Arno L, Quan J, Nguyen NT, Vanmarcke M, Tolkacheva EG, Dierckx H. A Phase Defect Framework for the Analysis of Cardiac Arrhythmia Patterns. Frontiers in Physiology. 2021; 12. https://doi.org/10.3389/fphys.2021.690453 PMID: 34630135

48. Bishop MJ, Plank G. Bidomain ECG Simulations Using an Augmented Monodomain Model for the Cardiac Source. IEEE transactions on bio-medical engineering. 2011; 58(8). https://doi.org/10.1109/TBME.2011.2148718 PMID: 21536529

49. Winfree AT. Electrical turbulence in three-dimensional heart muscle. Science. 1994; 266:1003–1006. https://doi.org/10.1126/science.7973648 PMID: 7973648

50. Clayton R, Zhuchkova E, Panfilov A. Phase singularities and filaments: Simplifying complexity in computational models of ventricular fibrillation. Prog Biophys Mol Biol. 2006; 90(1-3):378–398. https://doi.org/10.1016/j.pbiomolbio.2005.06.011 PMID: 16098568

51. Tomii N, Yamazaki M, Ashihara T, Nakazawa K, Shibata N, Honjo H, et al. Spatial phase discontinuity at the center of moving cardiac spiral waves. Computers in Biology and Medicine. 2021; 130:104217. https://doi.org/10.1016/j.compbiomed.2021.104217 PMID: 33516959

52. Goryachev A, Kapral R. Spiral Waves in Chaotic Systems. Physical Review Letters. 1996; 76 (10):1619–1622. https://doi.org/10.1103/PhysRevLett.76.1619 PMID: 10060475

53. Mermin ND. The topological theory of defects in ordered media. Reviews of Modern Physics. 1979; 51 (3):591–648. https://doi.org/10.1103/RevModPhys.51.591

54. Arno L, Kabus D, Dierckx H. Analysis of cardiac arrhythmia sources using Feynman diagrams; 2023. Available from: https://doi.org/10.48550/arXiv.2307.01508.

55. Arno L, Kabus D, Dierckx H. Strings, branes and twistons: topological analysis of phase defects in excitable media such as the heart; 2024. Available from: https://doi.org/10.48550/arXiv.2401.02571.

56. Niederer SA, Kerfoot E, Benson AP, Bernabeu MO, Bernus O, Bradley C, et al. Verification of Cardiac Tissue Electrophysiology Simulators Using an N-version Benchmark. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2011; 369(1954):4331–4351. https://doi.org/10.1098/rsta.2011.0139 PMID: 21969679

57. Campos JO, Oliveira RS, dos Santos RW, Rocha BM. Lattice Boltzmann method for parallel simulations of cardiac electrophysiology using GPUs. Journal of Computational and Applied Mathematics. 2016; 295:70–82. https://doi.org/10.1016/j.cam.2015.02.008

58. Rognes M E, Farrell P E, Funke S W, Hake J E, Maleckar M M C. Cbcbeat: An Adjoint-Enabled Framework for Computational Cardiac Electrophysiology. The Journal of Open Source Software. 2017; 2 (13):224. https://doi.org/10.21105/joss.00224

59. Antonioletti M, Biktashev VN, Jackson A, Kharche SR, Stary T, Biktasheva IV. BeatBox—HPC Simulation Environment for Biophysically and Anatomically Realistic Cardiac Electrophysiology. PLOS ONE. 2017; 12(5):e0172292. https://doi.org/10.1371/journal.pone.0172292 PMID: 28467407

60. Cooper FR, Baker RE, Bernabeu MO, Bordas R, Bowler L, Bueno-Orovio A, et al. Chaste: Cancer, Heart and Soft Tissue Environment. Journal of Open Source Software. 2020; 5(47):1848. https://doi.org/10.21105/joss.01848 PMID: 37192932

61. Plank G, Loewe A, Neic A, Augustin C, Huang YL, Gsell MAF, et al. The openCARP Simulation Environment for Cardiac Electrophysiology. Computer Methods and Programs in Biomedicine. 2021; 208:106223. https://doi.org/10.1016/j.cmpb.2021.106223 PMID: 34171774

62. Africa PC, Piersanti R, Regazzoni F, Bucelli M, Salvador M, Fedele M, et al. Lifex-Ep: A Robust and Efficient Software for Cardiac Electrophysiology Simulations. BMC Bioinformatics. 2023; 24(1):389. https://doi.org/10.1186/s12859-023-05513-8 PMID: 37828428

63. Arens S, Dierckx H, Panfilov AV. GEMS: A Fully Integrated PETSc-Based Solver for Coupled Cardiac Electromechanics and Bidomain Simulations. Frontiers in Physiology. 2018; 9:1431. https://doi.org/10.3389/fphys.2018.01431 PMID: 30386252

64. CARMEN. Cardiac ElectroPhysiology Simulator (CEPS);. IHU Liryc, Inria.

65. Finsberg HNT, van Herck IGM, Daversin-Catty C, Arevalo H, Wall S. simcardems: A FEniCS-based cardiac electro-mechanics solver. Journal of Open Source Software. 2023; 8(81):4753. https://doi.org/10.21105/joss.04753