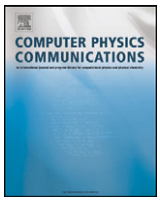
Computer Physics Communications 180 (2009) 2452–2471

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc



40th Anniversary Issue

Chaste: A test-driven approach to software development for biological modelling ☆

Joe Pitt-Francis [a,*], Pras Pathmanathan [a,**], Miguel O. Bernabeu [a], Rafel Bordas [a], Jonathan Cooper [a], Alexander G. Fletcher [b,c], Gary R. Mirams [d], Philip Murray [b], James M. Osborne [a,c], Alex Walter [b], S. Jon Chapman [b], Alan Garny [d], Ingeborg M.M. van Leeuwen [e], Philip K. Maini [b,c], Blanca Rodríguez [a], Sarah L. Waters [b], Jonathan P. Whiteley [a,b], Helen M. Byrne [f], David J. Gavaghan [a,c]

[a] Oxford University Computing Laboratory, Wolfson Building, University of Oxford, Parks Road, Oxford, OX1 3QD, UK
[b] Mathematical Institute, 24–29 St. Giles, Oxford, OX1 3LB, UK
[c] Oxford Centre for Integrative Systems Biology, South Parks Road, Oxford, OX1 3QU, UK
[d] Department of Physiology, Anatomy and Genetics, Sherrington Building, University of Oxford, Parks Road, Oxford, OX1 3PT, UK
[e] Department of Surgery and Molecular Oncology Ninewells Hospital, University of Dundee, Dundee, DD1 9SY, UK
[f] School of Mathematical Sciences, University of Nottingham, University Park, Nottingham, NG7 2RD, UK

ARTICLE INFO

Article history:
Received 31 March 2009
Received in revised form 29 June 2009
Accepted 1 July 2009
Available online 26 August 2009

PACS:
87
87.19.Hh
87.19.xj

Keywords:
Cardiac electrophysiology
Intestinal crypt simulation

ABSTRACT

Chaste ('Cancer, heart and soft-tissue environment') is a software library and a set of test suites for computational simulations in the domain of biology. Current functionality has arisen from modelling in the fields of cancer, cardiac physiology and soft-tissue mechanics. It is released under the LGPL 2.1 licence.

Chaste has been developed using agile programming methods. The project began in 2005 when it was reasoned that the modelling of a variety of physiological phenomena required both a generic mathematical modelling framework, and a generic computational/simulation framework. The Chaste project evolved from the Integrative Biology (IB) e-Science Project, an inter-institutional project aimed at developing a suitable IT infrastructure to support physiome-level computational modelling, with a primary focus on cardiac and cancer modelling.

Program summary

Program title: Chaste
Catalogue identifier: AEFD_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEFD_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: LGPL 2.1
No. of lines in distributed program, including test data, etc.: 5 407 321
No. of bytes in distributed program, including test data, etc.: 42 004 554
Distribution format: tar.gz
Programming language: C++
Operating system: Unix
Has the code been vectorised or parallelized?: Yes. Parallelized using MPI.
RAM: < 90 Megabytes for two of the scenarios described in Section 6 of the manuscript (Monodomain re-entry on a slab or Cylindrical crypt simulation). Up to 16 Gigabytes (distributed across processors) for full resolution bidomain cardiac simulation.
Classification: 3.
External routines: Boost, CodeSynthesis XSD, CxxTest, HDF5, METIS, MPI, PETSc, Triangle, Xerces
Nature of problem: Chaste may be used for solving coupled ODE and PDE systems arising from modelling biological systems. Use of Chaste in two application areas are described in this paper: cardiac electrophysiology and intestinal crypt dynamics.
Solution method: Coupled multi-physics with PDE, ODE and discrete mechanics simulation.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Principal corresponding author.
** Corresponding author.
E-mail addresses: Joe.Pitt-Francis@comlab.ox.ac.uk (J. Pitt-Francis), Pras.Pathmanathan@comlab.ox.ac.uk (P. Pathmanathan).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2009.07.019

Running time: The largest cardiac simulation described in the manuscript takes about 6 hours to run on a single 3 GHz core. See results section (Section 6) of the manuscript for discussion on parallel scaling.

Long write-up

1. Introduction

Chaste is a software development project which was set up in order to produce high-quality software for solving a variety of related problems in the field of computational biology. In this context, 'high-quality' means that the software is extensible, robust, fast, accurate, maintainable and uses state-of-the-art numerical techniques. In the past, typical academic software development methods involved researchers producing small pieces of code with no long-term plan for maintenance or extensibility. There are a few large-scale academic research codes for modelling problems arising from physiology which have been built and maintained over the last decade or so, but these have had limited release to the community at large. The reasons behind holding back source code, or releasing software to a limited number of academics, are varied but include prior claim to intellectual property, a need by the institution to commercialise the product and the researchers' own needs to extract as much scientific result as possible before passing the software on. By contrast, Chaste is the first package of its type to be released under an open source licence. In order to meet the 'high-quality' requirements given above, Chaste development is built on modern agile programming techniques and the software components within Chaste use highly-regarded existing software libraries wherever possible.

In the rest of this section we briefly describe the agile programming approach used in Chaste so that the user can see how the process has affected the structure of the code. We then present the theory and mathematical modelling context that underpin two of the physiological problems which Chaste is able to compute (Section 2). The two modelling domains are cardiac electrophysiology and intestinal crypt dynamics. The code-base is outlined first from a top–down view, showing how the libraries fit together (Section 3), and then from the level of an application, showing how the classes used in exemplars from the two application domains relate to each other (Section 4). After giving brief installation instructions (Section 5), we outline some results from the two application domains (Section 6) and conclude with a discussion of future directions for the Chaste software (Section 7).

1.1. Test-driven development

Chaste is written using an agile method adapted from a technique known as 'eXtreme programming' [1]. Although the methodology used to develop Chaste increases development time in the short term, the effort invested leads to savings in the medium to long term as we reap the benefits of the features that it offers. Specific features of this programming methodology that are relevant to the scientific computing domain are test-driven development, continuous integration, collective code ownership via pair programming, and frequent refactoring. For more details of the use of these features of eXtreme programming within the Chaste framework, and of how these features have been adapted for the scientific programming domain, see [2].

The use of agile techniques and, in particular, the use of test-driven development help to explain the hierarchical nature of the code—basic functionality is built and tested in a core library folder before it is deployed in the large-scale code. Before any new incremental functionality is added, code that tests this functionality is written. The code is then developed to incorporate the new functionality. After this, all tests including the new test are run. Should any test fail, then the code is corrected before any subsequent development takes place. As a consequence, many tests are written that cover very specific parts of the code, and all lines of code are tested. Should a test fail then it is relatively easy to identify the location of the problem. The use of small unit tests enables developers rapidly to discover, diagnose and fix bugs. Again, the interested reader is referred to [2] for a more detailed discussion of the software engineering approach.

2. The application domain (theoretical background)

Chaste has been designed with the aim of being a multi-purpose library supporting computational simulations for a wide range of biological problems. While it is a generic extensible library, software development to date has focused on the relatively mature area of cardiac electrophysiology (where the earliest computational simulation based on a mathematical model of a heart cell dates back to 1962 [3]) and the less well-developed areas of tissue growth and cancer growth. Example computational models from each of these two areas are selected for description in this paper, since they not only describe the main uses of the Chaste software, but also give insights into its generic nature. The exemplars allow us to discuss future directions for the software and further generic uses of Chaste. Both build on a common base of core library components such as geometric meshes, and the numerical solution of ordinary and partial differential equations (ODEs and PDEs, respectively), using state-of-the-art methods. Since cardiac modelling is such a mature field, the mathematical modelling approach is well established and in Section 2.1 we give only a brief outline together with the equations. Full details of the current state of the art in cardiac electrophysiology modelling can be found in the recent review papers [4,5]. In Section 2.2, however, where we describe the cancer modelling supported by Chaste, the more novel aspects of the underlying models require us to give much more detail of both the biology and the models.

2.1. Mathematical modelling of cardiac electrophysiology

Tissue-level cardiac electrophysiology is usually modelled using the bidomain equations, which comprise two PDEs coupled at each point in space with a system of ODEs [6]. Each system of ODEs represents the concentrations of ions and the state of the membrane proteins involved in ion transport within individual cardiac cells, while the PDEs model the electrical potential in intracellular and extracellular spaces as a reaction–diffusion system. Under certain circumstances [6] assumptions allow the bidomain equations to be reduced to the monodomain equations. In this section we outline how the bidomain equations are derived, and then briefly describe the algorithm used to compute their numerical solution.

2.1.1. The governing equations

We denote by Ω the region occupied by the cardiac tissue. The volume of cardiac tissue occupied by cells—the intracellular space—is denoted by Ω_i. The remainder of the cardiac tissue—the extracellular space—is denoted by Ω_e. Two strong determinants of cardiac electrophysiology are the intracellular and extracellular potentials, denoted by ϕ_i and ϕ_e. When modelling at the level of individual cells, ϕ_i is defined only on Ω_i, and ϕ_e is defined only on Ω_e. Making the assumption that the typical length-scale for cells is shorter than the typical length-scale of the features of the tissue-level solution allows the use of homogenisation techniques [6]. The homogenisation allows us to extend the definition of ϕ_i and ϕ_e to the whole of Ω, and write down a system of differential equations for ϕ_e, the transmembrane potential $V_m = \phi_i - \phi_e$ and various ionic concentrations and gating variables $\mathbf{u}$:

$$\chi\left(C_m\frac{\partial V_m}{\partial t} + I_{\text{ion}}(\mathbf{u}, V_m)\right) - \nabla\cdot\left(\sigma_i\nabla(V_m+\phi_e)\right) = 0, \tag{1}$$

$$\nabla\cdot\left((\sigma_i+\sigma_e)\nabla\phi_e + \sigma_i\nabla V_m\right) = 0, \tag{2}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(\mathbf{u}, V_m); \tag{3}$$

where σ_i is the intracellular conductivity tensor, σ_e is the extracellular conductivity tensor, χ is the surface area to volume ratio, C_m is the membrane capacitance per unit area, and I_{ion} is the transmembrane ionic current density. Functional forms for I_{ion} and $\mathbf{f}$ are determined by an electrophysiological cell model (see, for example, [7]).[1]

The intracellular and extracellular current densities, $\mathbf{i}_i$ and $\mathbf{i}_e$ respectively, are given by

$$\mathbf{i}_i = -\sigma_i\nabla\phi_i = -\sigma_i\nabla(V_m+\phi_e),$$

$$\mathbf{i}_e = -\sigma_e\nabla\phi_e.$$

Appropriate boundary conditions for Eqs. (1) and (2) are the specification of current density applied across the boundary

$$\mathbf{n}\cdot\left(\sigma_i\nabla(V_m+\phi_e)\right) = I_{s_i}, \tag{4}$$

$$\mathbf{n}\cdot(\sigma_e\nabla\phi_e) = I_{s_e}, \tag{5}$$

where $\mathbf{n}$ is the outward pointing unit normal vector to the tissue, I_{s_i} is the intracellular current density applied across the boundary, and I_{s_e} is the extracellular current density applied across the boundary. The system of equations (1)–(5) is then closed by specifying suitable initial conditions for V_m and $\mathbf{u}$ at all points of Ω.

2.1.2. Numerical solution of the governing equations

Eqs. (1)–(3) are solved using the finite element method at a discrete set of times $t_0 < t_1 < \cdots < t_N$. If the solution is known at time t_n we calculate the solution at time t_{n+1} using a semi-implicit method [8,9] to solve Eqs. (1)–(2). We make an implicit approximation to the linear terms in Eqs. (1)–(2), and an explicit approximation to the non-linear I_{ion} term. This results in the linear system

$$A\mathbf{v} = \mathbf{b}, \tag{6}$$

where the entries of matrix A are constant across time-steps (assuming a constant value of time-step) and the vector $\mathbf{v}$ contains the unknown values of V_m and ϕ_e. The linear terms are approximated implicitly to enhance numerical stability, whilst the non-linear term is evaluated explicitly to avoid having to solve a non-linear system at each time-step. The entries of $\mathbf{b}$ must be computed at each time-step. The system of ODEs at each point in space, Eq. (3), is solved by a suitable numerical scheme. A typical ODE system, such as Luo–Rudy 1991 [7] can be described in the implementation-independent CellML format [10] (see Section 4.2.1), contains 10 to 50 differential equations (representing the rate of change of ionic concentrations and of the number of open cell-membrane gates) and can be solved using any numerical ODE solver. The exact choice of numerical method for the ODE solution will vary according to the time- and space-resolutions required and according to the stability and convergence properties of particular schemes.

Furthermore, the computational efficiency for evaluating $\mathbf{b}$ can be improved by noting that most entries of this vector may be computed at a much lower resolution than is required for the fastest varying terms [11]. While the algorithms from [11] are not included in the Chaste release, they have been implemented using the Chaste framework and the details of the software engineering issues involved have been described by Bernabeu et al. [12].

2.2. Lattice-free model of tissue growth

Our second exemplar is a tissue growth model that bridges across spatial and temporal scales within a single, generic modelling framework [13,14]. The main focus of this component of Chaste is the modelling of the initiation of colorectal cancer in *intestinal crypts*, although the software has developed into a general tissue modelling framework which has also been used to model the growth of *tumour spheroids*, a common *in vitro* experimental system that mimics the morphology and growth of avascular tumours *in vivo*. In this paper we concentrate on the model of intestinal crypts. Chaste has already been used to explore novel science in cancer modelling such as the Wnt signalling pathway's role in cell proliferation in the crypts [15].

Colorectal cancer was originally chosen due to the availability of experimental data and because its biological understanding is sufficiently advanced to allow such a systems-level approach [14]. However, taking a systems, multi-scale approach means that the complexity of the resultant model is greatly increased, with a concomitant decrease in analytical tractability. Numerical methods of solution, and large-scale (and often expensive) simulation techniques therefore become increasingly important.

[1] A curated list of models is available at http://www.cellml.org/models.

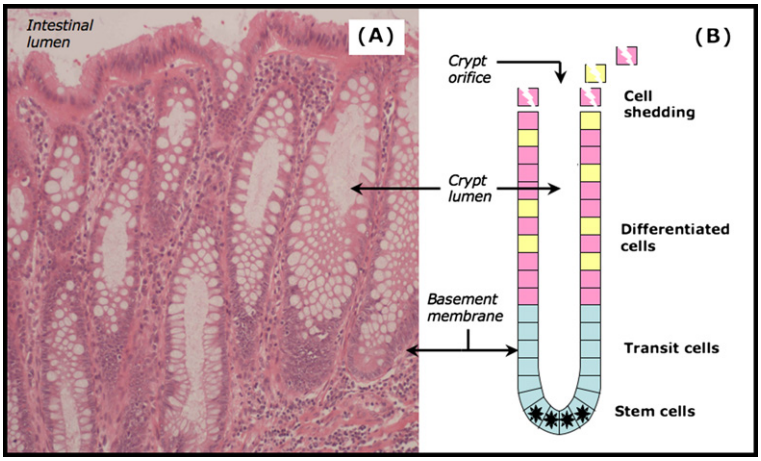


Fig. 1. Normal colonic mucosa. (A) Microdissection image courtesy of the Department of Pathology, Ninewells Hospital, Dundee, UK. The mucosa is characterised by the presence of numerous crypts of Lieberkühn. Murine and human intestinal crypts contain about 250 and 2000 epithelial cells, respectively [16,17]. (B) Schematic of a normal colonic crypt. These glands have the geometry of a 'three-dimensional test-tube' and are embedded in soft tissue; their orifices open to the intestinal lumen. The epithelial cells lining the crypt are tightly attached to their neighbours and to the underlying basement membrane, to maintain the integrity of the epithelial sheet and, thereby, it functions as a protective barrier and nutrient absorption engine.

Colorectal cancers originate within the epithelium that lines the luminal surface of the gut, as a result of an accumulation of genetic and epigenetic changes. This lining is made up of intestinal crypts, small test-tube-shaped structures embedded in the surface of the gut. A schematic of an intestinal crypt is given in Fig. 1. At the bottom of the crypt, a small population of stem cells is believed to continue regular division, producing 'transit' cells which divide several times before undergoing terminal differentiation. As new cells are added to (approximately) the lower third of the crypt, the sheet of epithelial cells moves upwards towards the orifice, where cells are shed into the intestinal lumen. This process is relatively rapid, with the epithelium renewing itself every few days.

As a first approximation, we model the crypt as a monolayer of cells lying on a cylindrical surface. Note that this over-estimates the number of cells in the bottom of the crypt. We have also extended this model using a projection method to account for the rounded form of the crypt bottom (Section 7). However, in this paper, where our aim is to illustrate a proof-of-principle, we omit this subtlety. We take a discrete approach, modelling each cell individually. For simulation purposes, it is convenient to roll the crypt out onto a two-dimensional (2D) plane, imposing periodic boundary conditions along the sides of the domain. Our basic model comprises three main components: (i) a model of the Wnt (a subcellular protein) signalling pathway; (ii) cell-cycle models, which together with information from the Wnt signalling pathway determine when cells are ready to divide and differentiate; and (iii) a mechanical model that controls cell migration at the macro-scale. In this paper we use modules that are based on recent advances in mathematical modelling [18–20]. However, it is important to point out that the flexibility of the Chaste framework is such that different mechanical, cell-cycle and Wnt-signalling modules (e.g. [21–24]) can be easily implemented and compared. The tumour spheroid model also comprises these three components, together with a fourth component which is a PDE solver. Reaction–diffusion PDEs model the diffusion and uptake of particular nutrients (for example, oxygen and glucose), with the cell model being dependent on the local nutrient concentrations.

2.2.1. Wnt-dependent cell-cycle model

The cell-cycle is the orderly sequence of events in which a cell duplicates its contents before dividing into two cells [25,26]. Since cancer is a disease caused in part by uncontrolled cell proliferation, the cell-cycle constitutes a major target for anti-cancer drug development. This has stimulated extensive experimental research and the formulation of detailed mathematical models, designed to enhance understanding of the regulatory networks involved and to explore potential therapeutic interventions. Such models are typically formulated as systems of coupled non-linear ODEs that characterise the change in the levels of key cell-cycle proteins [20,22]. In our multi-scale crypt model, every cell carries its own cell-cycle model, which can be influenced by the environment as follows. It has been proposed that the proliferative hierarchy observed along the longitudinal crypt axis—heavy proliferation at the crypt base, little or no proliferation at the top—correlates with the presence of a gradient of extracellular Wnt factors [27]. We therefore superimpose a spatial gradient of Wnt on our 2D surface, with high levels of Wnt at the crypt base and low levels at the orifice. We then employ a model for the Wnt pathway [18], formulated as a system of non-linear ODEs, to calculate the associated position-dependent levels of gene expression and use these to link the outcome of the Wnt model to the cell-cycle model developed in [20]. As a result, near the bottom of the crypt, where cells are exposed to high levels of Wnt, the production of Wnt-dependent cell-cycle control proteins is enhanced and cells progress through the cell-cycle. In contrast, near the crypt orifice where Wnt levels are low, little or no cell-division takes place.

2.2.2. Mechanical models

There are a variety of discrete model frameworks that can be used to describe the mechanical behaviour of tissue, ranging from lattice-based models (e.g. [28]), cell-centre ('point mass') models [29–31], and the non-point-mass vertex-based models [32]. In this paper we describe a tessellation-based cell-centre approach, in which the centres of adjacent cells are connected by linear springs [19]. The main feature of this model is that a Delaunay triangulation [33] is performed at each time-step, in order to determine which cells are connected. Let $\mathbf{r}_i$ be the position of the centre of the i-th cell, and define $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, and $\hat{\mathbf{r}}_{ij} = \mathbf{r}_{ij}/\|\mathbf{r}_{ij}\|$. The force exerted on cell i by an adjacent cell j is defined to be

$$\mathbf{F}_{ij} = \mu \hat{\mathbf{r}}_{ij}\big(\|\mathbf{r}_{ij}\| - s_{ij}\big), \tag{7}$$

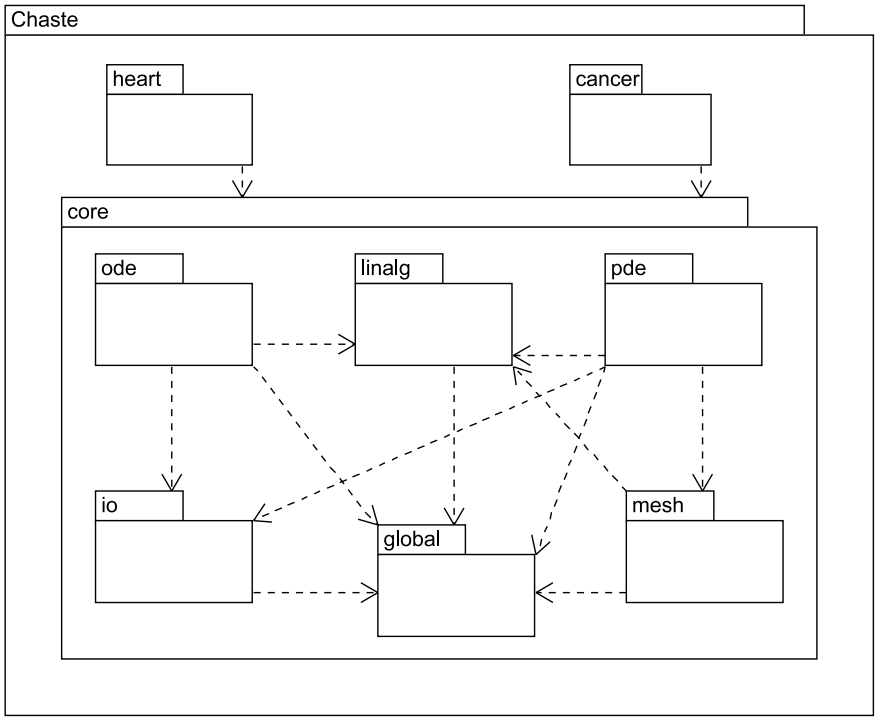


Fig. 2. The structure of the code-base. Arrows represent dependencies between components. The 'core' component is merely a convenient shorthand for referring to the libraries shown within it. Both the heart and cancer libraries depend on all core libraries.

where μ is the spring stiffness, and s_{ij} is the prescribed rest-length between cells i and j (i.e. the distance between them for which $\mathbf{F}_{ij} = \mathbf{0}$). The total force exerted on cell i by its neighbouring cells is

$$\mathbf{F}_i = \sum_j \mathbf{F}_{ij}, \tag{8}$$

where the sum is over all cells j that are connected to cell i. An over-damped limit is assumed, for which inertial effects are negligible compared to dissipative terms, so that the equation of motion is $\nu \frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \mathbf{F}_i$, where the cell viscosity, ν, models cell-substrate adhesion. This equation is discretised numerically using a forward Euler approach, from which it is straightforward to deduce that the position of the cell at time $(t + \Delta t)$ is related to its position at time t via

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \left(\frac{\Delta t}{\nu}\right)\mathbf{F}_i(t). \tag{9}$$

The rest-length s_{ij} between cells is assumed to be the typical diameter of a crypt cell. When a cell divides, as determined by its internal cell-cycle model, a new cell is placed a smaller fixed distance away from original cell's position in a random direction. The position of the original cell is perturbed to match. As in [19] the rest length, s_{ij}, between the cells which have just divided increases linearly over the course of an hour to the mature cell rest length (to emulate the mitosis phase of the cell-cycle). Thus, the cell-cycle model influences the mechanical model and, hence, cell movement, and the cell-cycle is influenced by the Wnt gradient imposed along the crypt axis.

3. Overview of software structure

Before giving detailed examples of how the models from Section 2 can be solved within Chaste, in this section we give an overview of the software structure. This serves as an indication of the shared functionality required by the main strands of the work (cardiac electrophysiology and tissue growth), a guide to the design choices behind the languages and libraries adopted, and a map of how the main library components and their main classes fit together.

3.1. Choice of language and libraries

3.1.1. Language

The main Chaste code has been written in C++ because it is a fast, object-oriented language. C++ is to some extent a compromise between speed and readability, since claims are made that older, more procedural languages such as FORTRAN can be better optimised by compilers and that, due to virtual redirection, C++ code will always be slower than the equivalent C code. Claims about the true comparisons of speed between languages are known to be dependent on the benchmarks and architectures chosen and also the skill of the writers of specific compilers [34]. Meanwhile, due mainly to encapsulation, object-oriented languages lead naturally to more modular code—software which is easier to abstract, to modify, and to document. The need for an object-oriented language arises from the requirements of testing, generality and extensibility, as we will discuss after the components of the system have been introduced. Fig. 2 displays the library components of Chaste, which are described in detail in Section 3.2.

The use of C++ (which has a large community of programmers) was also influenced by our requirement for the software to be easily tested. Software developers using object-oriented languages such as Java and C++ have built testing frameworks which lend themselves to automatic verification that certain components of the code function as intended. All functional code developed for Chaste has a corresponding test suite. Since the project uses test-driven development, the test suites were written at the same time as the functionality and are of comparable length [2]. The test suites are written using the C++ library CxxTest[2] which is a light-weight portable testing framework. A typical test suite file for a Chaste C++ class will include approximately 10 individual test methods. Each method will instantiate and set up relevant classes, check that no exceptions are thrown and then check that any numerical output is correct within some given tolerance.

3.1.2. Numerical and parallel libraries

The design of the code and decisions about when to re-use existing libraries are constantly evolving with each software development iteration. However, certain software decisions were taken at the outset of the project. This was because, in certain parts of the code's functionality, there are highly optimised and reliable software libraries available which can provide trusted functionality. Notably, linear algebra routines provide the most mature software libraries in the field of scientific computing with some packages having the advantage of many decades of development, optimisation and maintenance.

In the first design meetings it was decided that the Chaste code should be available to be run in a High Performance Computing platform. If this requirement were to be taken into account, then proper thought would have to be given to the best way of parallelising the code from the outset. Experience shows that sequential code can often be parallelised for a multi-core machine using a shared-memory paradigm such as OpenMP, but that sequential codes cannot often be ported to a distributed-memory cluster. The amount of reprogramming required to refactor a sequential code into a parallel code using, for example, the PVM or MPI libraries is often so great that the program is rewritten completely.

In order to make the best use of mature linear algebra routines and to ease the route to rapid parallelisation of parts of the code, we decided to use PETSc[3] for sparse large-scale linear algebra and MPI[4] for parallelisation. Since the core linear algebra used within Chaste is a small subset of PETSc, we have provided light-weight wrappers to common routines such as the creation of sparse matrices, choice of pre-conditioners and iterative solution of linear systems.

3.1.3. Other libraries

Experience with Chaste and previous projects [35] shows that parallel output from distributed memory codes provides a bottleneck which must be solved by either invoking a distributed file writer, concentrating all data onto one process, or collating multiple sequential files as a post-processing step. In Chaste, we have opted for the first of these options and are using an HDF5[5] distributed data writing object.

Also used throughout the code are: the Standard Template Library (for extensible vector structures and iterators); Boost[6] uBLAS (for small-scale sequential vectors and matrices); Boost serialisation (for check-pointing and parameter definitions in the cancer-related code); and CodeSynthesis XSD[7] (for XML parameter input in cardiac-related code).

In the area of ODE solvers, where it is required to solve large stiff non-linear sets of equations we preferred to code our own classes, and we have stayed close to our original designs. This is because having our own interface into ODE solution of the cell models enables us to experiment with various options for their solution (forward, backward and semi-implicit methods; multi-step and higher-order methods; error predicting and time-step adaptive methods). Although the experience gained from hand-crafting our own ODE solvers has been valuable, we realise that other users of the code may require more trusted, widely-used solvers. To that end we have provided an option to use light-weight interfaces to link against the CVODE solvers provided by Sundials.[8]

In the area of fault-tolerance and check-pointing there is, at present, only limited functionality in Chaste. While we have considered using fault-tolerant MPI [36] or a large-scale semi-automatic check-point library such as Condor [37], we realise that there is more to check-pointing than recovering data after a hardware failure. There are situations in computational biology where it is important to run a model to a steady-state solution before investigating patho-physiological conditions. If a steady-state can be saved as a check-point, then it can be used repeatedly. Similarly, in the case of cardiac electrophysiology, once a pathological case of tachycardia or fibrillation has been obtained *in silico*, it can be reused to investigate a variety of treatment regimes. Added to these use-cases is the need by some researchers to steer their computations—to interactively roll back to the last check-point, change some parameters and investigate their affect on the future computation. We currently use the Boost serialisation library to provide portable check-point files for crypt simulations in colorectal cancer. In cardiac simulations, there is limited check-point functionality: the linear algebra system and the associated cell models can be written to disk and restored. Check-pointing of entire simulations is an area of active development.

3.2. Top–down view of the libraries

Fig. 2 shows a schematic of the library structure within Chaste. The mathematical modelling of both cardiac and cancer domains introduced in Section 2 relies heavily on (i) accurate representation of a complex, realistic geometry provided primarily through unstructured finite-element meshes; (ii) conversion, using the finite element method (FEM) [38], of PDEs from a given domain into a sparse system of linear (or non-linear) equations, known as *assembly*; (iii) the iterative solution of large sparse systems via PETSc and (iv) the numerical solution of ODE systems. These four common features of the code become four component libraries: `mesh`, `pde`, `linalg` and `ode` respectively. They are supported by two lower-level libraries, `global` (which is responsible for initialisation and book-keeping calls to

[2] http://cxxtest.tigris.org.
[3] http://www.mcs.anl.gov/petsc/.
[4] http://www.mcs.anl.gov/mpi/.
[5] http://www.hdfgroup.org/HDF5/.
[6] http://www.boost.org.
[7] http://www.codesynthesis.com/products/xsd/.
[8] http://www.llnl.gov/CASC/sundials/.

PETSc as well as taking care of exceptions) and `io` (which handles input and output, mainly via wrapped calls to HDF5 functionality). Built upon these are the two modelling components of Chaste, `heart` and `cancer`.

Each of the six core components and the two main modelling components comprise a number of interacting C++ classes, each in their own source file, together with a set of CxxTest test suites. The test suites are written in order to verify correct functionality of the given component and to cover all lines of code in that component (together with some lines from lower-level components). (Coverage of the functionality of each component by its test suites is verified on a daily basis using gcov, which is part of the Gnu gcc project.[9])

The build system used is Scons.[10] When a developer compiles the code of Chaste, they typically build and test either a single test-suite, a library component or an entire raft of test-suites—known as the continuous test pack. In every case, code is compiled as necessary and executables are constructed corresponding to each test-suite. These test-suite executables are run as they become available and summary pass–fail web-pages are produced for the entire set of tests. This system makes it easy for a developer to understand the impact that a change in source code has across the entire archive of test-suites.

3.3. The Chaste cardiac executable

Most cardiac electrophysiological simulations are variations on common tests but with differing input parameters: meshes, stimulation protocols, cell models, numerical time-steps, etc. Because of this, a single Chaste cardiac executable has been developed which can read the input parameters of a simulation from an XML description, run a single simulation and dump the results ready for post-processing. The executable is produced from a `main` function which has limited functionality. All the executable's features (from opening an XML parameter file, through setting up and running a simulation on a given number of processors, to writing results back to disk) are implemented in the main libraries and are tested in lower-level tests throughout the hierarchy, so that the amount of code unique to the executable is minimised. Simple use of the XML input to the executable is demonstrated in Section 6.

4. A look at the Chaste code via two exemplars

In this section we examine two typical scenarios for running Chaste tests (following the cardiac and cancer modelling mathematical descriptions given in Section 2). These exemplar programs are taken from the Chaste tutorial tests (available within the Open Source release). They serve to give the developer an idea of the functionality available within some of the main library components.

The first exemplar is concerned with propagation of electrical activity in a heart geometry using the monodomain or bidomain equations. The result of such a study is illustrated by Fig. 6 in Section 6. This type of simulation requires the instantiation of about 4.3 million Luo–Rudy 1991 cell ODE models [7] and solution of a linear system with 8.6 million unknowns at every time-step. Despite the complexity of such a simulation, it will be seen in Section 4.2 that the application programmer's code is straightforward.

The second exemplar (Section 4.3) shows how a simple model of intestinal crypt turnover can be written using Chaste classes. This exemplar models the cell monolayer as a cylindrical mesh with stem cells at the base of the cylinder and natural removal at the top (see Fig. 8 in Section 6 for a visualisation of this type of model). It is also shown that, with a few simple changes in the code, the application programmer is able to simulate the growth of a tumour spheroid.

4.1. Frequently used components

Before explaining the two exemplars it is necessary to introduce a few of the classes which are frequently used throughout the codebase. This will illustrate the firm foundation which is provided by the generic core code.

4.1.1. Singleton classes

There are a number of C++ objects used in physiological simulations for which there ought to be at most one instantiation per simulation. Notably, if simulation parameters are needed it is important that they are recorded and used in a consistent way. For this reason the parameters classes use a *singleton design pattern* [39]. For example, the following code will use a given configuration file and override the duration of a cardiac simulation to 1 ms:

```
HeartConfig *inst = HeartConfig::Instance();
inst->SetParametersFile("./ChasteParameters.xml");
inst->SetSimulationDuration(1.0); // ms
```

When the `Instance()` method is first called, then either the existing `HeartConfig` object is taken and modified or, if no object exists, a new one is created using a set of default configuration parameters. Similar behaviour is seen in the cancer modelling code:

```
TissueConfig::Instance()->SetSDuration(11.0); // hours
```

The cancer code also makes use of the `SimulationTime` singleton class, which provides a consistent current time to the multiple time-dependent components of the tissue growth models.

[9] http://gcc.gnu.org.
[10] http://www.scons.org.

Another notable singleton class is `RandomNumberGenerator`. Using this class, which records how the pseudo-random numbers are seeded and logs every use, it is possible to produce repeatable streams of random numbers even if the generator is stored in a check-point and restarted.

4.1.2. Time-stepping

The use of time-stepping is common within all physiological models.

```
TimeStepper my_stepper(start_time, end_time, time_step);
while ( !my_stepper.IsTimeAtEnd() )
{
    // do something
    my_stepper.AdvanceOneTimeStep();
}
```

The idea behind the time-stepper is to provide a robust way to deal with time loops. A naïve programmer might explicitly state the number of steps in a simulation and iterate using fixed point numbers or they might iterate using a floating point loop and trust that an inequality test will always give the expected answer. Since neither of these alternatives is ideal, the time-stepper uses an incremented integer counter to store its state (and to calculate the current time) and will always finish at the correct end time. This may be achieved by making the final time-step smaller than others.

4.1.3. The unstructured mesh hierarchy

All mesh geometry within Chaste is based on simplices (triangles in 2D, tetrahedra in 3D). There is a hierarchy of meshes derived from an `AbstractMesh` class which are templated over the dimensionality of their elements (1 for beams, 2 for triangles or 3 for tetrahedra) and over the space in which they are embedded. Normally the dimensions of the elements and of the embedding space will be the same, but it is possible to introduce a 1D mesh in 3D (say, to model the Purkinje system in the heart, which behaves like a network of insulated electrical cables) or a 2D mesh in 3D (say, to model a monolayer of cells in a colonic crypt). There are mesh readers and writers available for a variety of file formats. The following code loads a triangular mesh in a 3D geometry:

```
TrianglesMeshReader<2,3> mesh_reader("./slab_395_elements");
TetrahedralMesh<2,3> mesh;
mesh.ConstructFromMeshReader(mesh_reader);
```

The default concrete mesh class is `TetrahedralMesh` but there are three notable others: `MutableMesh`, which supports changes in mesh geometry and topology, as well as re-meshing via external library calls[11]; `ParallelTetrahedralMesh` which partitions the geometry using METIS[12] and then loads only a required part of the mesh (and corresponding halo nodes) into each process; and `QuadraticMesh`, which has more nodes per element than the meshes made from linear simplices.

4.1.4. The finite element assemblers and solvers

A hierarchy of finite element assemblers exists within the Chaste `pde` library. Their role is to take a mathematical description of a set of PDEs together with their boundary conditions on a relevant mesh, to form a corresponding linear system and solve it. A straightforward assembler would take a tetrahedral mesh and the weak form of a static (time-independent) PDE together with appropriate boundary conditions, form a linear system (as in Eq. (6)) and solve it via a wrapped call to a PETSc linear system solver. The assembler used to solve the bidomain PDEs is a dynamic (time-dependent) assembler, although for this problem the left-hand matrix A from Eq. (6) needs only be assembled once at the outset, with the right-hand-side vector $\mathbf{b}$ assembled each time-step.

The family of PDE assemblers available within Chaste includes static and dynamic assemblers, assemblers which use linear or quadratic basis functions (the latter using the `QuadraticMesh` class mentioned above), and linear and non-linear assemblers (assemblers which are able to solve a non-linear PDE through repeated solution of a linear approximation). The hierarchy of finite element assemblers makes full use of the multiple inheritance available within C++, so that, for example, one is able to instantiate an assembler for an approximation to a non-linear, dynamic, problem by constructing a class which inherits each of the required features.

4.2. Cardiac electrophysiology exemplar code

4.2.1. Automatically generated cell-level code

At the lowest level in cardiac simulation is the cardiac cell model. The complexity of the electrophysiology models that prescribe the I_{ion} term in Eq. (1) and the size and complexity of the system of ODEs in Eq. (3) have grown substantially in recent years. These models typically consist of several separate currents, are dependent on tens of quantities and consist of systems of tens of ODEs. Furthermore, each of these ODEs includes several constant parameters. When one research group develops a model as complex as this, it is clearly difficult for another group to use this model, as coding it without introducing errors in any of the constants is a near impossible task.

[11] Triangle (http://www.cs.cmu.edu/~quake/triangle.html) and Tetgen (http://tetgen.berlios.de/) provide re-meshing functionality in 2D and 3D respectively.

[12] http://www.cs.umn.edu/~metis.

The CellML mark-up language [40] was developed to facilitate sharing and re-use of these models. CellML is an XML-based language that represents electrophysiological models (amongst others) as a structured document that may be read by both humans and computers. Chaste programs utilise the locally developed open source tool PyCml [41,42] for CellML verification and for automatic generation of efficient C++ code. For details of the current state of CellML, see [42] and the CellML website.[13]

4.2.2. Cell factories

Fundamental to cardiac electrophysiological simulation is the idea that one should be able to take a given tissue geometry (FEM mesh), to provide cell models at various geometric nodes and to allow these cell models to undergo electrical stimulation. To facilitate this construction of geometry-dependent cell models, there is an `AbstractCardiacCellFactory` class which visits every node in the geometry expecting to associate a (possibly stimulated) cell model to the node. Here is an example of a concrete 2D subclass of `AbstractCardiacCellFactory` which, on construction, produces a cell stimulus which lasts for the first 0.5 ms of the simulation. This stimulus is attached to a Luo–Rudy 1991 cell model [7] on the node at the origin. Other nodes have unstimulated Luo–Rudy 1991 cells.

```
// User-defined cell factory inheriting from the 2D
// AbstractCardiacCellFactory, providing a
// CreateCardiacCellForTissueNode() method
class PointStimulus2dCellFactory : public AbstractCardiacCellFactory<2>
{
private:
  // Store the stimulus to be used at the point of stimulus
  SimpleStimulus *mpStimulus;

public:
  //Class constructor
  PointStimulus2dCellFactory() : AbstractCardiacCellFactory<2>()
  {
    //Create stimulus of -6V for the first 0.5 ms of the simulation
    mpStimulus = new SimpleStimulus(-6000.0, 0.5);
  }

  AbstractCardiacCell* CreateCardiacCellForTissueNode(unsigned nodeIdx)
  {
    // Cache the location of the tissue node
    double x = this->mpMesh->GetNode(nodeIdx)->rGetLocation()[0];
    double y = this->mpMesh->GetNode(nodeIdx)->rGetLocation()[1];


    // Only stimulate the node if it is at the origin
    if (fabs(x)+fabs(y)<1e-6)
    {
       // The stimulated cell
       return new LuoRudyIModel1991OdeSystem(mpSolver, mpStimulus);
    }
    else
    {
      // The other cells have zero stimuli
      return new LuoRudyIModel1991OdeSystem(mpSolver, mpZeroStimulus);
     }
  }
}
```

4.2.3. The cardiac problem, PDE and assembler classes

From the application writer's point-of-view, the main machinery of the solution of the bidomain equations is encapsulated inside the `BidomainProblem` class which, as can be seen from Fig. 3, is a subclass of `AbstractCardiacProblem`. This allows for the solution of the simplified *monodomain* equation via the `MonodomainProblem` subclass from much the same code as the bidomain equations.

The `BidomainProblem` class brings together the geometric mesh `AbstractMesh`, the cell factory in Section 4.2.2, the bidomain equations (Eqs. (1)–(2)) in a `BidomainPde` class, the boundary conditions from Eqs. (4)–(5), and the family of finite element assemblers into a single place. This problem class initialises data from the `HeartConfig` singleton, verifies that nothing is missing and that the data are consistent. It then solves the problem using time loops in a PDE time-step, writing data out where required.

In order to elucidate the use of the singleton parameter class from Section 4.1.1 and the cell factory from Section 4.2.2 within a simple bidomain simulation, we now present a code fragment. Note that the reference to `PointStimulus2dCellFactory` enables the `BidomainProblem` class to visit the nodes of the "SmallSlab" mesh and to associate a given cell model with each point in the mesh.

[13] http://www.cellml.org.

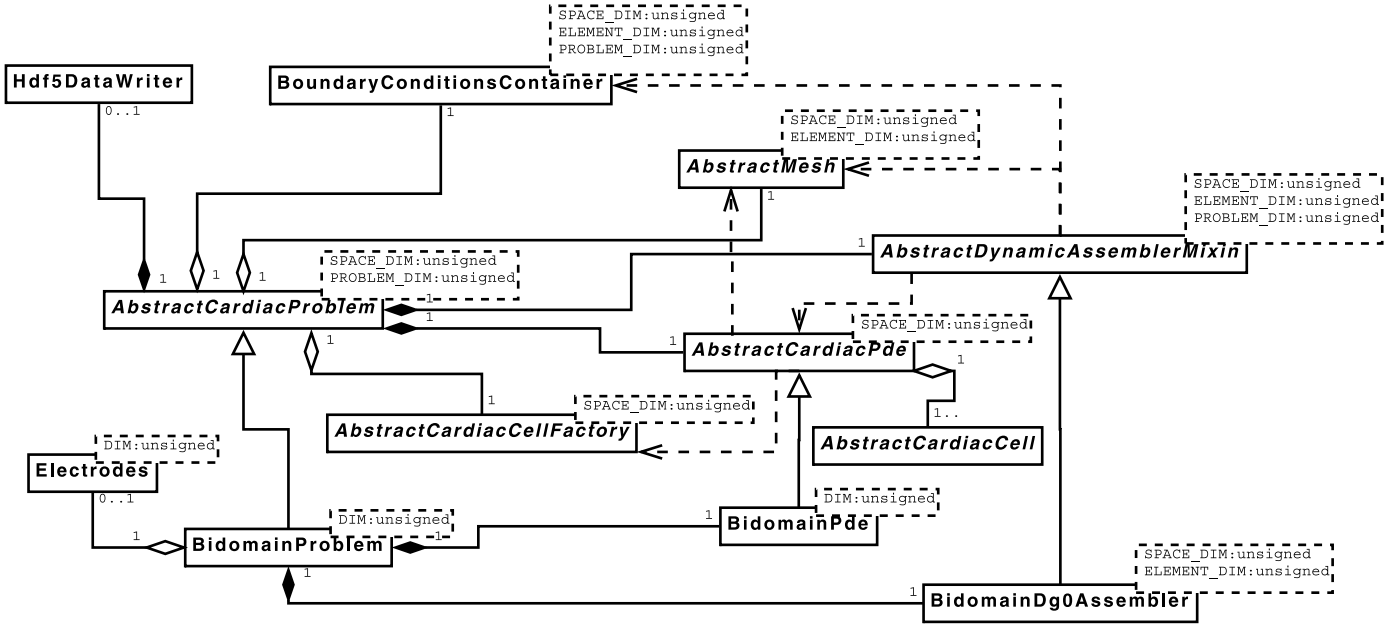


Fig. 3. A UML diagram showing the primary classes involved in a cardiac electrophysiology simulation. The core class is `AbstractCardiacProblem` which collects together the components of a simulation. It has two subclasses for monodomain and bidomain simulations; only the latter is shown here. The `BidomainDg0Assembler` class is responsible for using information from the mesh, PDE, cardiac cell and boundary conditions classes to assemble and solve a finite-element linear system at each time-step. Data are periodically written via `Hdf5DataWriter` which is a light-weight wrapper to the HDF5 library.

```
// Use of the BidomainProblem class
HeartConfig *inst = HeartConfig::Instance();
inst->SetSimulationDuration(25.0); // ms
inst->SetOutputDirectory("BidomainTutorial");
inst->SetMeshFileName("SmallSlab");

// Create a cell factory of the type we defined above
PointStimulus2dCellFactory cell_factory;

// Create a problem class using (a pointer to) the cell factory
BidomainProblem<2> bidomain_problem( &cell_factory);

// Initialise (and check input)
bidomain_problem.Initialise();

// Enable post-processing of data
bidomain_problem.ConvertOutputToMeshalyzerFormat();

bidomain_problem.Solve();
```

This code fragment gives an indication of the ease with which a modelling experiment may be written and amended. For example, one might run a similar experiment on a realistic 3D tissue geometry by editing the call to `SetMeshFileName` and changing template parameters from 2 to 3. More examples are available as tutorials on the Chaste website.

4.3. Cancer exemplar code

In this section we provide an overview of the structure of the cancer code-base. This is illustrated in Fig. 4, which displays the main classes forming the cancer system. Note that many of these are abstract classes, with several concrete implementations.

4.3.1. Cell-cycle models and cells

At the lowest level in the code are the *cell-cycle models*. A number of these have been defined, including a `WntCellCycleModel` which implements the model described in Section 2.2.1 (and [18]). Other examples of cell-cycle models include those where division occurs after a fixed time; or after a normally-distributed time; and an `OxygenBasedCellCycleModel`, where cell-division is dependent on the local oxygen concentration. A `TissueCell` class then takes in a cell-cycle model, and provides all the functionality required of a single cell, in particular: functionality relating to age, type, generation, and mutational status; functionality to undergo mitosis; and

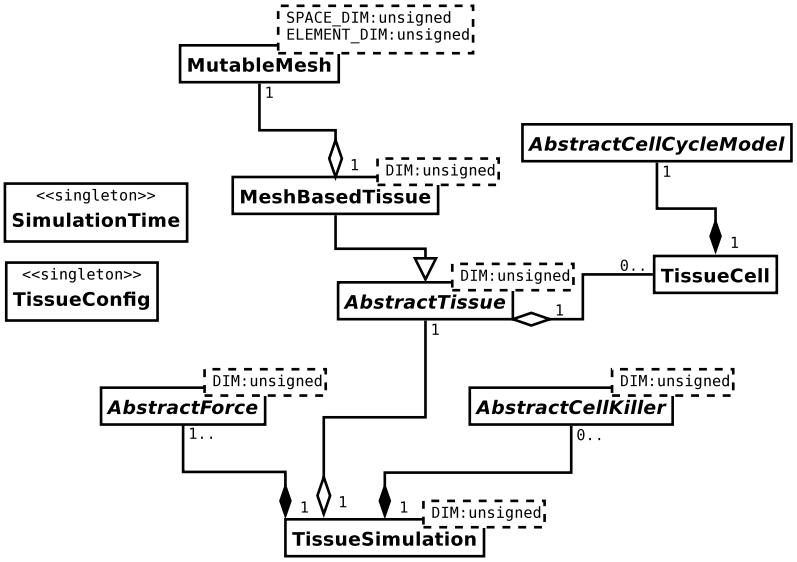


Fig. 4. A UML diagram showing the primary classes involved in a cancer simulation. The tissue classes link a collection of cells (which are agnostic of spatial location) with geometric information, usually provided by a mesh. The `TissueSimulation` class combines this with force balances which determine how cells move, and objects defining when cells should die, in order to perform a simulation. Parameters and the simulation time are provided by singleton classes. Each cell has a corresponding cell-cycle model. Different implementations of the abstract classes can be provided to investigate different scenarios.

functionality to undergo apoptosis (programmed cell death). For example, the following code generates a healthy transit cell with a Wnt-based cell-cycle model:

```
TissueCell cell(TRANSIT, HEALTHY, new WntCellCycleModel());
```

4.3.2. The cylindrical mesh and tissue classes

Crypt simulations on cylindrical geometries make use of the `Cylindrical2dMesh` class, which inherits from `MutableMesh` but implements the extra functionality required for enforcing periodic boundaries, including the ability to re-mesh on a cylindrical domain. This means that no other component has to deal with issues relating to the imposition of periodic boundary conditions. With the help of the `HoneycombMeshGenerator` class, which creates a mesh where all connected nodes are a unit distance apart, the following code can be used for generating periodic meshes for the cylindrical crypt simulation (an example of such a mesh can be seen in Fig. 8 in Section 6). Here, we create a mesh which has 16 cells in the horizontal direction and 18 cells in the vertical direction, and the `true` parameter indicates that the mesh should be cylindrical:

```
HoneycombMeshGenerator generator(16, 18, true);
Cylindrical2dMesh* p_cylindrical_mesh = generator.GetCylindricalMesh();
```

By contrast the following code can be used for generating a non-cylindrical mesh (for example, for use in a tumour spheroid simulation—which in 2D is a tumour ovoid):

```
HoneycombMeshGenerator generator(16, 18, false);
MutableMesh<2,2>* p_mesh = generator.GetMesh();
```

The `MeshBasedTissue` class then takes in a set of cells and a mesh, which may be cylindrical or not, and maintains coherence between the two as cells are added or deleted as part of cell-birth or cell-death. For example:

```
MeshBasedTissue<2> tissue(cells, p_cylindrical_mesh);
```

or

```
MeshBasedTissue<2> tissue(cells, p_mesh);
```

where `cells` is of type `std::vector<TissueCell>`, i.e. all the cells corresponding to the nodes in the mesh. Note that the `MeshBasedTissue` class is for models where cell-connectivity is determined through triangulation—a second tissue class, `NodeBasedTissue`, has been defined for 'overlapping spheres' models, where connectivity is based on relative distance.

4.3.3. Force objects

Several force objects have been defined, the simplest being `GeneralisedLinearSpringForce`, which provides the forces acting on each cell using Eq. (7). It inherits from the `AbstractForce` class, which defines an interface for all force objects. In the simulations described in this paper, we consider only a simple mechanical model where the total force on a particular cell arises wholly through linear interactions with neighbouring cells. However, more complex scenarios are possible, either through non-linear interactions or by introducing additional terms on the right-hand side of Eq. (8) (for example, representing chemotactic or gravitational contributions to the total force). Further force classes have been implemented, each derived from `AbstractForce`, and each only implementing the relevant *contribution* to the force, not the total force. Since the simulation objects (defined below) can take in any number of `AbstractForce` classes, simulations using a wide range of (total) force laws can be put together in a straightforward manner by picking the required contributory force classes.

4.3.4. Cell-killer objects

Cell-killer objects contain rules used to determine which cells should be removed from the simulation and when. For example, a `SloughingCellKiller` has been introduced to identify cells for sloughing in the crypt by virtue of their position—cells above a certain height are removed to simulate sloughing into the lumen. Additionally, an `OxygenBasedCellKiller` is used in tumour spheroid simulations to eliminate cells whose local nutrient concentration is too low for them to remain alive.

4.3.5. The tissue simulation classes

At the highest level are the tissue simulation classes, `CryptSimulation2d` and `TissueSimulationWithNutrients`. These inherit common functionality from a `TissueSimulation` class, which deals with cell-birth and death, and is written to run in all spatial dimensions, so that the same code is used to simulate a 2D crypt or a 3D tumour spheroid. The derived classes `CryptSimulation2d` and `TissueSimulationWithNutrients` deal with crypt- and tumour-spheroid-specific concerns only (such as boundary conditions and integrating the PDE governing nutrient concentration, respectively). They take in a tissue, any number of force classes, and any number of cell-killers, as illustrated in the following code, which creates a crypt simulation on a cylindrical mesh. Here, a `GeneralisedLinearSpringForce` is created and placed in a `std::vector` of `AbstractForce`s. An object of type `CryptSimulation2d` is then created, taking in the tissue and the forces. Next, it is given a `SloughingCellKiller` and sets an end-time, following which `Solve()` is called.

```
GeneralisedLinearSpringForce<2> force;
std::vector<AbstractForce<2>*> force\_collection;
force_collection.push_back(&force);

CryptSimulation2d simulator(tissue, force_collection);

SloughingCellKiller<2> sloughing_cell_killer(&tissue);
simulator.AddCellKiller(&sloughing_cell_killer);
simulator.SetEndTime(100); // 100 hours
simulator.Solve();
```

Tumour spheroid simulations are set up in a similar manner, as shown below. Here, `SimpleNutrientPde` is a class which encapsulates the PDE governing the oxygen concentration in the spheroid. This PDE is a quasi-steady reaction–diffusion equation, with point-wise sink terms modelling cellular consumption of oxygen. `TumourSpheroidSimulation` contains an assembler class which solves the PDE at each time-step by using FEM.

```
SimpleNutrientPde<2> pde;

GeneralisedLinearSpringForce<2> force;
std::vector<AbstractForce<2>*> force_collection;
force_collection.push_back(&force);

OxygenBasedCellKiller<2> oxygen_based_killer(&tissue);

TumourSpheroidSimulation<2> simulator(tissue, force_collection, &pde);
simulator.AddCellKiller(&oxygen_based_killer);
simulator.SetEndTime(100); // 100 hours
simulator.Solve();
```

We note that the object-based structure, in addition to being amenable to component-wise testing as described above, is especially advantageous in such discrete tissue models, since it allows several different simulations to be created in a straightforward manner, by using whichever components are desired, and prevent unnecessary repetition of code shared by the crypt and tumour spheroid models. More examples are available as tutorials on the Chaste website.

5. Installation instructions

In the current release, Chaste has many dependencies including Scons, MPI, PETSc, METIS, HDF5 and XSD. Cardiac Chaste is available as a standalone executable (Section 3.3) which has all the dependencies wrapped by way of static linking. Use of the executable only requires the user to configure XML input and to pre- and post-process mesh data.

Full installation of the library dependencies, in order to create a development environment, requires more work. However, we have provided a set of 'README' files, which provide detailed, step-by-step instructions on the installation of each dependency. We are also working on a semi-automatic installation script and on an installer based on Linux packages.

6. Results

We now illustrate the capability of the Chaste framework by presenting results from some computational simulations. As stated above, we focus on the monodomain/bidomain equations in cardiac electrophysiology and on the cylindrical crypt model in cancer growth.

6.1. Cardiac results

6.1.1. Tissue slab with re-entrant spiral wave

Our first example involves using the Chaste cardiac executable (introduced in Section 3.3) to model a 2D monodomain experiment with a simple double stimulation routine. We shall describe this S1–S2 stimulation protocol via excerpts from the XML input file. The `<Simulation>` section of the XML input describes the basic geometry, a mesh representing a 2D sheet, 4 cm by 6 cm which is triangulated on the fly in a regular manner to have an inter-node space of 0.02 cm.

```
<SpaceDimension>2</SpaceDimension>
<SimulationDuration unit="ms">2000.0</SimulationDuration>
<Domain>Mono</Domain>
<Mesh unit="cm">
  <Sheet inter\_node\_space="0.02" x="4.0" y="6.0"/>
</Mesh>
```

The stimulus protocol consists of an initial stimulus (S1) on the left edge of the mesh. As the cells are undergoing repolarisation, a second stimulus (S2) is applied to a lower-left rectangular region. This sets up a re-entrant spiral wave, as can be seen in Fig. 5.

```
<Stimuli>
  <Stimulus>  <!-- S1 -->
    <Strength unit="uA/cm^3">-255000.0</Strength>
    <Duration unit="ms">2.0</Duration>
    <Delay unit="ms">0.0</Delay>  <!-- At start -->
    <Location unit="cm">
      <Cuboid> <!-- Left edge (z values ignored, since 2D)-->
        <LowerCoordinates x="0" y="0" z="0"/>
        <UpperCoordinates x="0.02" y="6" z="0.1"/>
      </Cuboid>
    </Location>
  </Stimulus>
  <Stimulus>  <!-- S2 -->
    <Strength unit="uA/cm^3">-255000.0</Strength>
    <Duration unit="ms">2.0</Duration>
    <Delay unit="ms">215.0</Delay> <!-- 215ms after S1 -->
    <Location unit="cm">
      <Cuboid> <!- Lower-left rectangle (z values ignored)-->
        <LowerCoordinates x="0" y="0" z="0"/>
        <UpperCoordinates x="2" y="4.5" z="0.1"/>
      </Cuboid>
    </Location>
  </Stimulus>
</Stimuli>
```

6.1.2. Bidomain wave propagation in realistic geometry

A study of bidomain propagation can be carried out using new cardiac anatomical data. The resulting mesh [43] is at an unprecedented resolution, due to advanced MRI technology. The mesh, cell models and linear system would take up to 16 Gigabytes of memory were it to be run as a sequential job. However, the code given below uses `ParallelTetrahedralMesh` which partitions the mesh in order to load balance a parallel job.

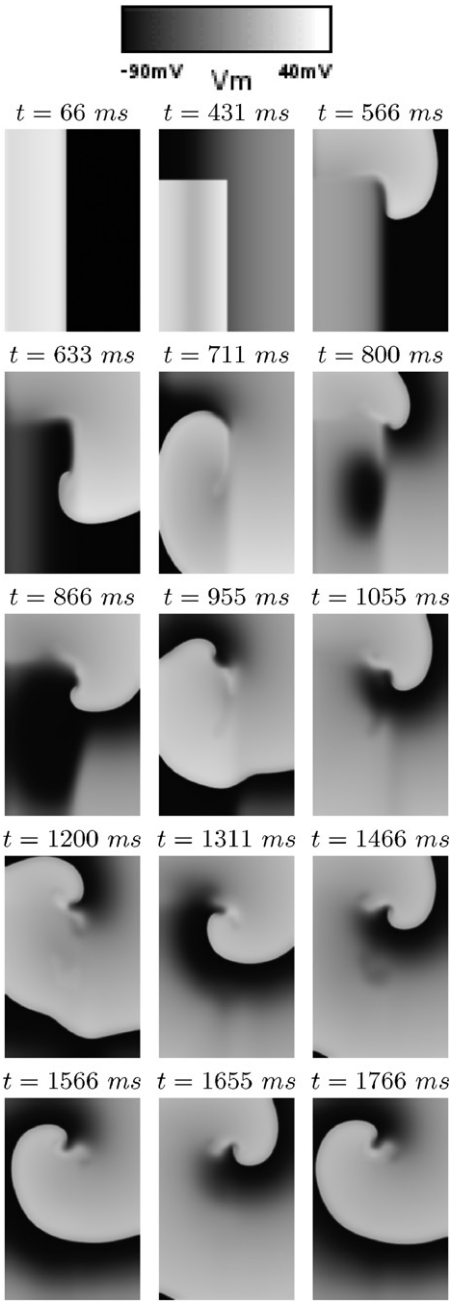


Fig. 5. Inducing a spiral wave in a 2D tissue by application of an S1–S2 stimulus protocol. The S1 stimulus of the left edge is applied for the first 2 ms, inducing a wave travelling from left to right (first frame). The second stimulus (S2) of a rectangular region is applied from 255–257 ms (between the first and second frames). The newly excited rectangular region can be seen in the second frame. The variable plotted is the transmembrane potential (V_m).

The `ApexStimulusHeartCellFactory` used in this code segment is similar to the cell factory introduced in Section 4.2.2 but stimulates a region at the apex (bottom) of the heart based on the z-coordinate of mesh nodes. This stimulus, which lasts for the first half millisecond of simulation time, mimics the way electrical excitation travels from the apex after it has been delivered by the Purkinje system. This code was used to produce Fig. 6.

```
HeartConfig::Instance()->Reset();
HeartConfig::Instance()->SetOdePdeAndPrintingTimeSteps(0.005, 0.01, 1);
HeartConfig::Instance()->SetSimulationDuration(200); // ms
HeartConfig::Instance()->SetOutputFilenamePrefix("ApexStimulus");
HeartConfig::Instance()->SetKSPSolver("symmlq");
HeartConfig::Instance()->SetKSPPreconditioner("bjacobi");

ApexStimulusHeartCellFactory<LuoRudyIModel1991OdeSystem> cell_factory;
BidomainProblem<3> cardiac_problem(&cell_factory);
```

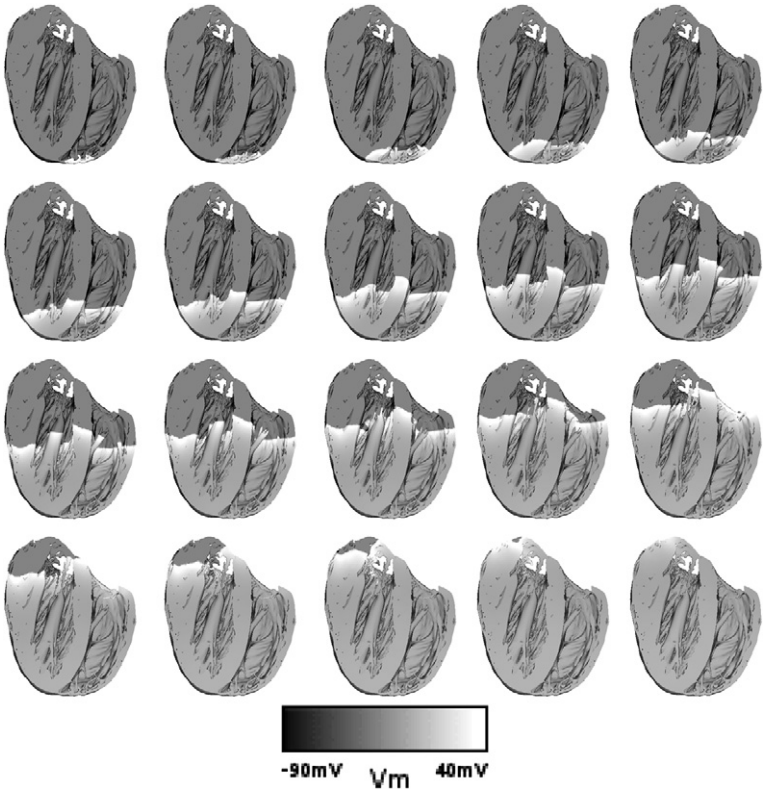


Fig. 6. Bidomain propagation on a high-resolution rabbit ventricle mesh with about 4.3 million nodes (8.6 million unknowns in the PDEs). The sequence shows 0.2 seconds of simulation time after stimulating a small region around the apex. The variable plotted is the transmembrane potential (V_m).

```
TrianglesMeshReader<3,3> mesh_reader("data/OxfordHeart_i_triangles");
ParallelTetrahedralMesh<3,3> mesh;
mesh.ConstructFromMeshReader(mesh_reader);

cardiac_problem.SetMesh(&mesh);

cardiac_problem.Initialise();
cardiac_problem.Solve();
```

6.1.3. Parallel efficiency

Fig. 7 shows the parallel performance of Chaste on a 3.0 GHz Xeon cluster using a low-latency Infiniband interconnect. The testbed is 10 ms of bidomain cardiac activity of a rabbit ventricular mesh containing 24,217,344 elements. The model was stimulated at the apex at $t = 0$ ms to simulate standard apex to base propagation. Scalability stays over 85% for up to 8 processors, 75% for 32 processors and 53% for 64 processors.

Our benchmark results show that our bidomain simulator spends 80% of the time solving the linear system arising from FEM discretisation. In this scenario, the overall parallel performance of Chaste is bounded by the scalability achieved for the linear system solver. As mentioned earlier, PETSc is used for the iterative solution of our linear systems. Therefore, Fig. 7 plots the parallel performance of PETSc as a theoretical maximum. To evaluate the scalability of PETSc, we have used systems of linear equations from the bidomain simulations, since parallel performance is problem-dependent in this case.

It is worth pointing out that the parallel scaling benchmark used to produce Fig. 7 represents a relatively simple electrophysiological experiment (pacing the tissue at the apex of the heart). However, because there is no spatial or temporal adaptivity in the standard Chaste bidomain simulation, the parallel scaling seen for this experiment should be typical of a range of experiments such as stimulation via the Purkinje system or shock-induced fibrillation. If Chaste were to be used to model fibrillation on this high-resolution data we would expect the parallel performance to be very similar. This is because the work done in solving the ODE systems will remain unchanged (if the ODE and PDE time-steps are everywhere constant) while the cost of solving the linear system may increase. This increase in the number of iterations of the linear solver would arise because the temporal coherence of the wavefront in normal pacing provides the solver with a good initial guess to the next solution.

Recent developments in multi-core architectures have made parallel computing accessible to a wider audience. It is now possible to find up to 8 cores in desktop machines at low-to-moderate cost. Chaste achieves better performance than that presented in Fig. 7 on multi-core platforms because of the much higher communication bandwidth, making tissue-level simulations possible on a wider range of computational resources.

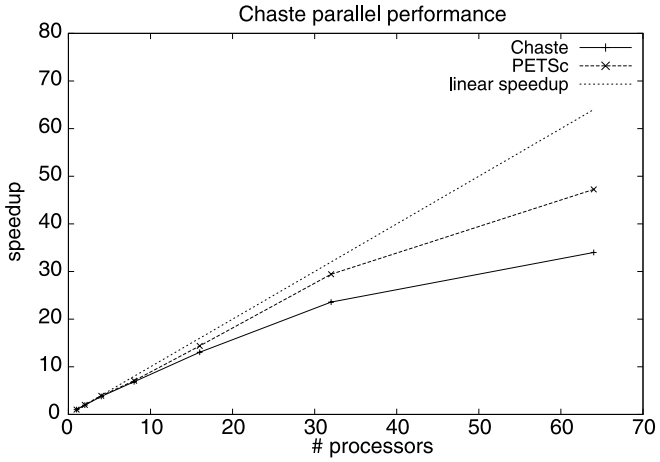


Fig. 7. Speedup for a bidomain propagation experiment on the high-resolution mesh (Fig. 6). The speedup data for PETSc represent the solution of a series of typical linear algebra systems extracted from the full problem.

6.2. Cancer results

While cardiac electrophysiological modelling is a mature field, cancer modelling is less well-developed. In this area Chaste has been used to model novel science as is demonstrated by the following two examples.

6.2.1. Monoclonal conversion in crypts

Two theories have been proposed with regard to the stem cells in the crypt. First, multiple stem cells remain at the base of the crypt *ad infinitum* (this idea is incorporated in [19] as a modelling assumption). By contrast, the second theory states that a single cell can dominate the stem cell pool and hence the entire crypt. Recently, mitochondrial DNA mutations have been used to track the progress of different populations within the crypt [44–46]. The data suggest that, over time, a single cell's progeny can dominate a crypt, via a process termed monoclonal conversion.

As stated above, the model developed in [19] is consistent with the first theory—multiple stem cells reside at the base of the crypt where they are held or 'pinned' in position. To model the second theory we 'un-pin' the stem cells and impose a spatially-varying Wnt profile which decreases linearly along the crypt axis. For simplicity, we assume that a cell divides after a fixed period of time if its Wnt stimulus is greater than a threshold value, chosen so that the overall rate of cell turnover in the crypt is the same for both models. When the cells at the crypt base are un-pinned, it is no longer possible to distinguish between stem and transit cells, and no cells have a fixed position.

The following code illustrates how such a cylindrical crypt simulation is run by assembling the components introduced in Section 4 into a tissue-level simulation (introduced in Section 4.3.5).

```
// Initialise the global time object
SimulationTime::Instance()->SetStartTime(0.0);

// Create a cylindrical mesh (16 cells by 18 cells, true->cylindrical)
HoneycombMeshGenerator generator(16, 18, true);
Cylindrical2dMesh* p_cylindrical_mesh = generator.GetCylindricalMesh();

// Create a set of cells, each using a WntCellCycleModel
std::vector<Cell> cells;
CellsGenerator<2>::GenerateForCrypt(cells, *p_mesh, WNT);

// (label some cells, code omitted for brevity)

// Create a Tissue object (naming it 'crypt'), using mesh & cells
Tissue<2> crypt(*p_mesh, cells);

// Set up the 2D WntConcentration information: linearly increasing
// up the crypt, and associate it with the Tissue object
WntConcentration<2>::Instance()->SetType(LINEAR);
WntConcentration<2>::Instance()->SetTissue(crypt);

// Create a system of linear springs
GeneralisedLinearSpringForce<2> linear_force;
std::vector<AbstractForce<2>*> force_collection;
force_collection.push_back(&linear_force);
```

```
// Create the simulator object, and set the end time (100 hours)
CryptSimulation2d simulator(crypt, force_collection);
simulator.SetEndTime(100);

// Create a sloughing cell-killer and add it to the simulator object
SloughingCellKiller sloughing_cell_killer(&crypt);
simulator.AddCellKiller(&sloughing_cell_killer);

// Run the simulation
simulator.Solve();
```

Fig. 8 shows how the spatial distribution of daughter cells produced by a single labelled stem cell changes over time and how the evolution depends on the underlying mathematical model. The results presented in the first column reveal that, when the stem cells are fixed at the base of the crypt, the label spreads through the crypt as the cells divide and migrate upwards. However, there is little lateral movement and, while the labelled cells persist in the crypt, they do not dominate it, i.e. the crypt is polyclonal. By contrast, the results presented in columns 2 and 3 suggest that when the stem cells are not fixed at the crypt base, the crypt eventually becomes monoclonal. Columns 2 and 3 display results from the same simulation, in which stem cells are un-pinned and a spatially varying Wnt stimulus is imposed along the crypt axis. Different cells are labelled at $t = 0$ hours in columns 2 and 3. Over time, the progeny of the cell that was initially labelled in column 2 is washed out, whereas the progeny of the cell that is labelled in column 3 persists, eventually occupying the entire crypt.

6.2.2. Tumour spheroids

We conclude this section by describing a further example to illustrate the generality of the Chaste framework, the modelling of multicellular tumour spheroids. Fig. 9 displays a number of snap-shots of the development of a 2D tumour spheroid (produced by running code similar to that outlined in Section 4.3.5). In this simulation, the nutrient concentration is determined through the solution of a quasi-steady reaction–diffusion PDE, and, using the `OxygenBasedCycleModel` class, a cell's progress through the cell-cycle depends on its oxygen uptake. A cell that experiences low nutrient levels for a prolonged period enters cell-cycle arrest and may, eventually, undergo apoptosis/necrosis, with cells experiencing intermediate nutrient levels becoming quiescent. This model reproduces the standard spheroid structure found experimentally, in which a necrotic core forms in the centre of the spheroid due to nutrient deprivation. This region is bounded by a thin quiescent layer and an outer proliferating rim. Depending on the precise modelling assumptions, the spheroid may eventually exhibit growth saturation and a steady state. By developing more complex cell-cycle models, we may investigate the overall dynamics of a spheroid in which some cells have a mutation (for example, in the hypoxia-inducible factor HIF-1) which affects their proliferative or apoptotic behaviour under hypoxic conditions.

7. Discussion and future work

Chaste is a well-engineered suite of software libraries built in a test-driven development environment. This test-first policy has led to a piece of software which is readily understandable (via test suites and documentation) and which is easily extensible without having to change the existing functionality.

In this description of the Chaste software, it has not been possible fully to document all the functionality and modelling capabilities of the code. Therefore in this section we will summarise recent work and future directions.

In the area of cardiac electrophysiology, we have, for example, implemented a multi-scale modelling algorithm, in which full cell models are only evaluated at a coarse spatial scale and in which partial cell models use interpolated values for slowly varying quantities [12]. We have also incorporated a solver of incompressible non-linear elasticity problems in order to model soft-tissue deformation, and coupled this model to the cardiac electrophysiological equations and a cellular active-tension ODE model, using a semi-implicit stable algorithm [47], to simulate cardiac electromechanics. Another recent line of work includes a stochastic model of ion exchange in the cell membranes. Further work in cardiac modelling will involve fluid-mechanics coupling, further load-balancing and parallel efficiency improvements and parallel check-pointing. Cardiac Chaste is being used as the base software in the European VPH preDiCT project for modelling how pharmaceutical compounds affect cardiac rhythm; therefore the preDiCT project is informing some of the future software requirements.

In the area of cancer research, recent developments have included a crypt projection model. Here the crypt bottom (missing from the cylindrical approximation) is projected from a 2D planar model onto the surface of a paraboloid that resembles a colonic crypt. In this approximation, the spring–spring interactions and the Delaunay triangulation are adjusted to accurately measure distance in the metric space of the paraboloid [15].

Biological function and behaviour arise as a result of the integration of multiple processes acting across a wide range of spatio-temporal scales. It is therefore important to develop multi-scale computational models to, for example, investigate the effectiveness of combinations of therapies that may operate at various spatial and temporal scales. The Chaste crypt model has allowed us to start addressing such issues in the context of colorectal cancer growth. In addition, it provides us with a valuable computational tool for exploring how the choice of different cell-based models and/or choice of reduced intracellular dynamics model may affect global dynamics and error propagation.

Chaste already has the infrastructure for biological simulation and is ripe for use by other related application areas, e.g. the study of cell seeding policies in tissue engineering. Further details on Chaste, including visualisation movies and user-support, are available at `http://web.comlab.ox.ac.uk/chaste/`. Available on the website to support both users and developers are the code source, the compiled cardiac executable, code-development tutorials, workshop exercises and automatically-generated code documentation. The tutorial and exercise material was used recently (March 2009) in an international Chaste workshop.

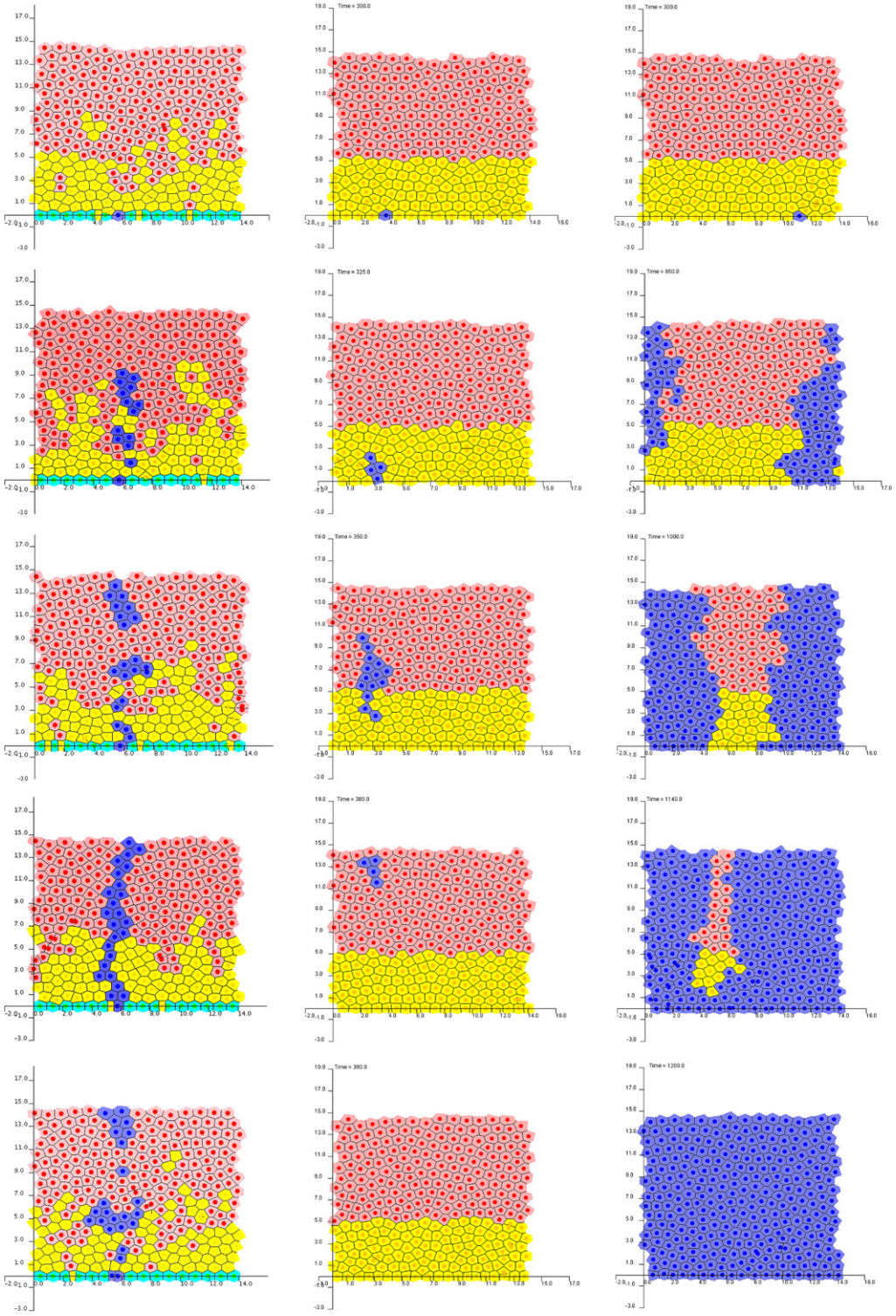


Fig. 8. Clonal expansion in the crypt. Transit cells are light grey, differentiated cells are medium-dark grey, and labelled cells are dark grey. Stem cells, only present in the first column, are also medium-dark grey (the bottom row of cells in each image in column one). Top row: A cell at the base of the crypt is labelled. Subsequent rows progress through time (different times for each column to best display progression of the clone). Column 1 shows a simulation with stem cells pinned at the crypt base, where each stem cell supports a population of cells but no single population dominates. Column 2 shows a simulation in which stem cells are not pinned and a particular (labelled) stem cell and its progeny are washed out; the same simulation is shown in column 3, but here the marked clone eventually takes over the crypt.

Acknowledgements

The authorship of this paper represents contributions to the Chaste code from researchers in both cardiac and cancer modelling. We should like to thank numerous other contributors to the code-base.

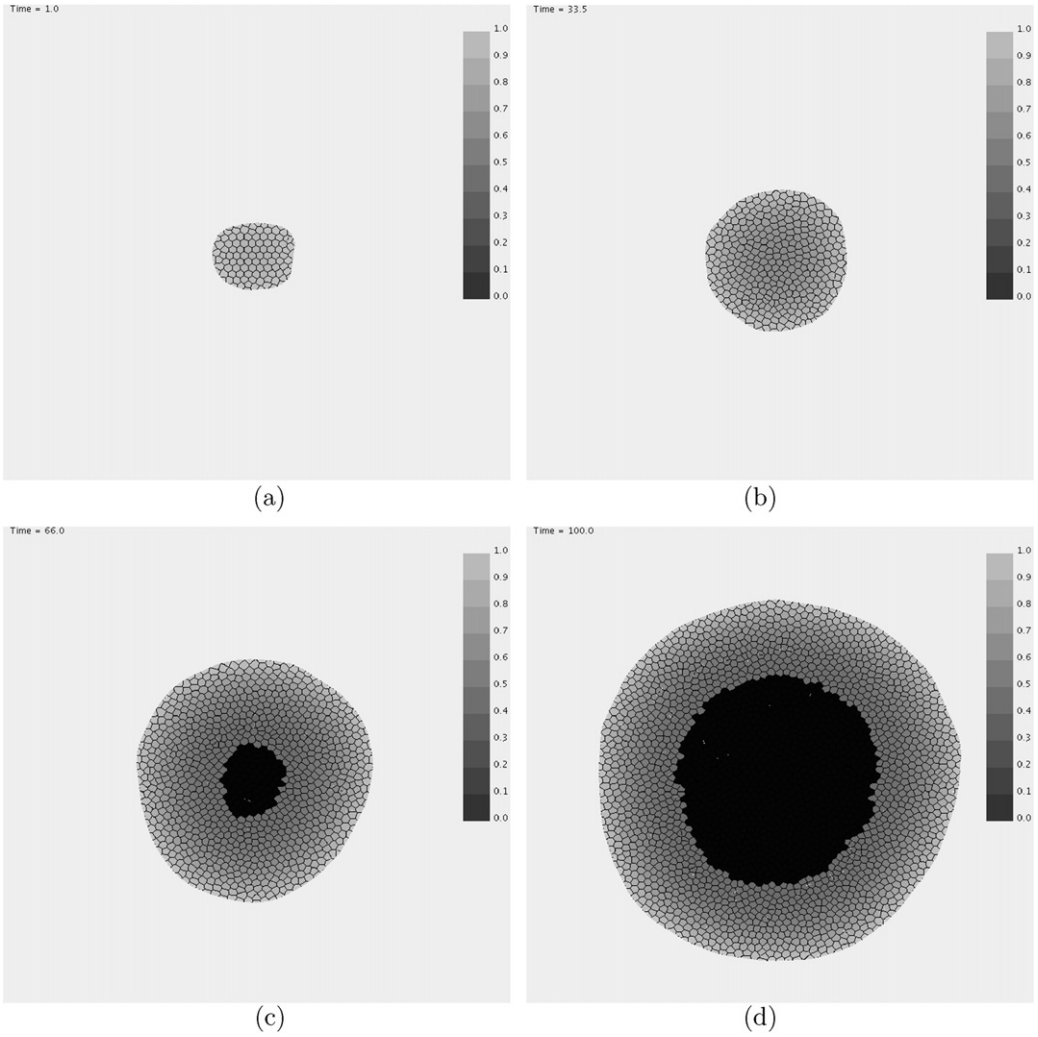


Fig. 9. A 2D tumour spheroid simulation. In all frames, dead cells are shown in black and living cells in grey, with the colour representing the oxygen concentration (lighter grey representing higher oxygen concentration). (a) $t = 3$ h, (b) $t = 33.5$ h, (c) $t = 66$ h, (d) $t = 100$ h.

Chaste began thanks to the Engineering and Physical Sciences Research Council (EPSRC) of the UK under their e-Science pilot project in Integrative Biology (GR/S72023/01). Recent development work on Cardiac Chaste is supported by the EPSRC as part of a Software for High Performance Computing project (EP/F011628/1).

PP is supported by the EPSRC through grant EP/D048400/1. JPW and AW are supported by Award No. KUK-C1-013-04, made by King Abdullah University of Science and Technology (KAUST). BR is supported by an MRC Career Development Award (G0700278). SLW gratefully acknowledges funding from the EPSRC in the form of an Advanced Research Fellowship. JC is funded through the VPH NoE project (#223920) and AG through the VPH preDiCT and VPH euHeart projects (#224381 and #224495, respectively). All three VPH projects are supported by the European Commission, DG Information Society, through the Seventh Framework Programme of Information and Communication Technologies. JMO and AGF are funded through the OCISB project (BB/D020190/1) and the Life Sciences Interface and Systems Biology Doctoral Training Centres. PKM was partially supported by a Royal Society-Wolfson Merit award and NIH Grant U56CA113004 from the National Cancer Institute. PKM and HMB acknowledge support from PMI2 (British Council).

Thanks to Fujitsu Laboratories of Europe for providing validation and timings necessary for Fig. 7.

References

[1] K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, Addison–Wesley, Boston, 2004.
[2] J. Pitt-Francis, M.O. Bernabeu, J. Cooper, A. Garny, L. Momtahan, J. Osborne, P. Pathmanathan, B. Rodríguez, J.P. Whiteley, D.J. Gavaghan, Chaste: Using agile programming techniques to develop computational biology software, Phil. Trans. R. Soc. A 366 (2008) 3111–3136.
[3] D. Noble, A modification of the Hodgkin–Huxley equations applicable to Purkinje fibre action and pacemaker potentials, J. Physiol. 160 (1962) 317–352.
[4] E.J. Vigmond, R. Weber dos Santos, A.J. Prassl, M. Deo, G. Plank, Solvers for the cardiac bidomain equations, Prog. Biophys. Mol. Biol. 96 (2008) 3–18.
[5] R. Bordas, B. Carpentieri, G. Fotia, F. Maggio, R. Nobes, J. Pitt-Francis, J. Southern, Simulation of cardiac electrophysiology on next-generation high-performance computers, Phil. Trans. R. Soc. A 367 (1895) (2009) 1951–1969, doi:10.1098/rsta.2008.0298.
[6] J.P. Keener, J. Sneyd, Mathematical Physiology, Springer, New York, 1998.
[7] C.-H. Luo, Y. Rudy, A model of the ventricular cardiac action potential: Depolarization, repolarization, and their interaction, Circ. Res. 68 (1991) 1501–1526.
[8] J.P. Keener, K. Bogar, A numerical method for the solution of the bidomain equations in cardiac tissue, Chaos 8 (1998) 234–241.
[9] J.P. Whiteley, An efficient numerical technique for the solution of the monodomain and bidomain equations, IEEE Trans. Biomed. Eng. 53 (2006) 2139–2147.
[10] C. Lloyd, M. Halstead, P. Nielsen, CellML: its future, present and past, Prog. Biophys. Mol. Biol. 85 (2004) 433–450.
[11] J.P. Whiteley, An efficient technique for the numerical solution of the bidomain equations, Ann. Biomed. Eng. 36 (2008) 1398–1408.

[12] M. Bernabeu, R. Bordas, P. Pathmanathan, J. Pitt-Francis, J. Cooper, A. Garny, D. Gavaghan, B. Rodríguez, J. Southern, J. Whiteley, Chaste: incorporating a novel multi-scale spatial and temporal algorithm into a large-scale open source library, Phil. Trans. R. Soc. A 367 (1895) (2009) 1907–1930, doi:10.1098/rsta.2008.0309.
[13] I.M.M. van Leeuwen, H.M. Byrne, O.E. Jensen, J.R. King, Crypt dynamics and colorectal cancer: advances in mathematical modelling, Cell Proliferation 39 (3) (2006) 157–181.
[14] I.M.M. van Leeuwen, C.M. Edwards, M. Ilyas, H.M. Byrne, Towards a multiscale model for colorectal cancer, World J. Gastroenterol. 13 (9) (2007) 1399–1407.
[15] I.M.M. van Leeuwen, G.R. Mirams, A. Walter, A. Fletcher, P. Murray, J. Osborne, S. Varma, S.J. Young, J. Cooper, B. Doyle, J. Pitt-Francis, L. Momtahan, P. Pathmanathan, J.P. Whiteley, S.J. Chapman, D.J. Gavaghan, O.E. Jensen, J.R. King, P.K. Maini, S.L. Waters, H.M. Byrne, An integrative computational model for intestinal tissue renewal, Cell Proliferation 42 (5) (2009) 617–636, doi:10.1111/j.1365-2184.2009.00627.x.
[16] C.S. Potten, C. Booth, D. Hargreaves, The small intestine as a model for evaluating adult tissue stem cell drug targets, Cell proliferation 36 (2003) 115–129.
[17] Y. Yatabe, S. Tavaré, D. Shibata, Investigating stem cells in human colon by using methylation patterns, PNAS 98 (19) (2001) 10839–10844.
[18] I.M.M. van Leeuwen, H.M. Byrne, O.E. Jensen, J.R. King, Elucidating the interactions between the adhesive and transcriptional functions of beta-catenin in normal and cancerous cells, J. Theor. Biol. 247 (1) (2007) 77–102.
[19] F. Meineke, C. Potten, M. Loeffler, Cell migration and organization in the intestinal crypt using a lattice-free model, Cell proliferation 34 (4) (2001) 253–266.
[20] M. Swat, A. Kel, H. Herzel, Bifurcation analysis of the regulatory modules of the mammalian G_1/S transition, Bioinformatics 20 (10) (2004) 1506–1511.
[21] M. Weliky, G. Oster, The mechanical basis of cell rearrangement. I. Epithelial morphogenesis during Fundulus epibody, Development 109 (1990) 373–386.
[22] B. Novak, J.J. Tyson, A model for restriction point control of the mammalian cell cycle, J. Theor. Biol. 230 (2004) 563–579.
[23] E. Lee, A. Salic, R. Krüger, R. Heinrich, M.W. Kirschner, The roles of APC and axin derived from experimental and theoretical analysis of the Wnt pathway, Pub. Lib. Sci. Biol. 1 (2003) 116–132.
[24] G. Mirams, H. Byrne, J. King, A multiple timescale analysis of a mathematical model of the Wnt/β-catenin signalling pathway, J. Math. Biol. (online ahead of print), doi:10.1007/s00285-009-0262-y.
[25] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, Molecular Biology of the Cell, Garland Science, New York, 2002.
[26] D.O. Morgan, The Cell Cycle. Principles of Control, Oxford University Press, Oxford, 2006.
[27] C. Gaspar, R. Fodde, APC dosage effects in tumorigenesis and stem cell differentiation, Intl. J. Dev. Biol. 48 (5–6) (2004) 377–386.
[28] M. Loeffler, R. Stein, H.E. Wichmann, C.S. Potten, P. Kaur, S. Chwalinski, Intestinal crypt proliferation. I. A comprehensive model of steady-state proliferation in the crypt, Cell Tissue Kinetics 19 (1986) 627–645.
[29] D. Drasdo, S. Holme, A single-cell-based model of tumor growth in vitro: monolayers and spheroids, Physical Biology 2 (3) (2005) 133–147.
[30] J. Galle, Modeling the effect of deregulated proliferation and apoptosis on the growth dynamics of epithelial cell populations in vitro, Biophysical Journal 88 (2005) 62–75.
[31] E. Pallson, A three-dimensional model of cell movement in multicellular systems, Future Generation Computer Systems 17 (7) (2001) 835–852.
[32] M. Weliky, S. Minsuk, R. Keller, G. Oster, Notochord morphogenesis in Xenopus Laevis: simulation of cell behavior underlying tissue convergence and extension, Development 113 (4) (1991) 1231–1244.
[33] B. Delaunay, Sur la sphère vide, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk 7 (1934) 793–800.
[34] J.R. Cary, S.G. Shasharina, J.C. Cummings, J.V. Reynders, P.J. Hinker, Comparison of C++ and Fortran 90 for object-oriented scientific programming, Computer Physics Communications 105 (1) (1997) 20–36, doi:10.1016/S0010-4655(97)00043-X.
[35] J. Pitt-Francis, A. Garny, D. Gavaghan, Enabling computer models of the heart for high-performance computers and the grid, Phil. Trans. R. Soc. A 364 (1843) (2006) 1501–1516.
[36] G.E. Fagg, J. Dongarra, FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world, PVM/MPI 2000 (2000) 346–353.
[37] P. Lemarinier, A. Bouteiller, G. Krawezik, F. Cappello, Coordinated checkpoint versus message log for fault tolerant MPI, Int. J. High Perform. Comput. Netw. 2 (2-4) (2004) 146–155, doi:10.1504/IJHPCN.2004.008899.
[38] J.N. Reddy, An Introduction to the Finite Element Method, McGraw–Hill, New York, 1993.
[39] E. Gamma, R. Helm, R. Johnson, J.M. Vlissides, Design Atterns: Elements of Reusable Object-Oriented Software, Addison–Wesley, 1995.
[40] A.A. Cuellar, C.M. Lloyd, P.M.F. Nielsen, D.P. Bullivant, D.P. Nickerson, P.J. Hunter, An overview of CellML 1.1, a biological model description language, Simulation 79 (2003) 740–747.
[41] J. Cooper, S. McKeever, A. Garny, On the application of partial evaluation to the optimisation of cardiac electrophysiological simulations, in: PEPM '06: Proceedings of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, ACM Press, New York, NY, USA, 2006, pp. 12–20.
[42] A. Garny, D.P. Nickerson, J. Cooper, R. Weber dos Santos, A.K. Miller, S.W. McKeever, P.M.F. Nielsen, P.J. Hunter, CellML and associated tools and techniques, Phil. Trans. R. Soc. A 366 (2008) 3017–3043.
[43] M. Bernabeu, M. Bishop, J. Pitt-Francis, D. Gavaghan, V. Grau, B. Rodriguez, High performance computer simulations for the study of biological function in 3D heart models incorporating fibre orientation and realistic geometry at para-cellular resolution, Computers in Cardiology (2008) 721–724, doi:10.1109/CIC.2008.4749143.
[44] R.W. Taylor, M.J. Barron, G.M. Borthwick, A. Gospel, P.F. Chinnery, D.C. Samuels, G.A. Taylor, S.M. Plusa, S.J. Needham, L.C. Greaves, T.B.L. Kirkwood, D.M. Turnbull, Mitochondrial DNA mutations in human colonic crypt stem cells, J. Clin. Invest. 112 (9) (2003) 1351–1360, doi:10.1172/JCI19435.
[45] L.C. Greaves, S.L. Preston, P.J. Tadrous, R.W. Taylor, M.J. Barron, D. Oukrif, S.J. Leedham, M. Deheragoda, P. Sasieni, M.R. Novelli, et al., Mitochondrial DNA mutations are established in human colonic stem cells, and mutated clones expand by crypt fission, PNAS 103 (3) (2006) 714–719, doi:10.1073/pnas.0505903103.
[46] S.A. McDonald, S.L. Preston, L.C. Greaves, S.J. Leedham, M.A. Lovell, J.A. Jankowski, D.M. Turnbull, N.A. Wright, Clonal expansion in the human gut: Mitochondrial DNA mutations show us the way, Cell Cycle 5 (8) (2006) 808–811.
[47] P. Pathmanathan, J.P. Whiteley, A numerical method for cardiac mechanoelectric simulations, Annals of Biomedical Engineering 37 (5) (2009) 860–873, doi:10.1007/s10439-009-9663-8.