



OXFORD CENTRE FOR COLLABORATIVE APPLIED MATHEMATICS

Report Number 09/14

**Chaste: a test-driven approach to software
development for physiological modelling**

by

**Joe Pitt-Francis, Pras Pathmanathan, Miguel O.
Bernabeu, Raf Bordas, Jonathan Cooper,
Alexander Fletcher, Gary R. Mirams, Philip
Murray, James Osborne, Alex Walter, S. Jon
Chapman, Alan Garny, Inge M. M. van
Leeuwen, Philip K. Maini, Blanca Rodriguez,
Jonathan P. Whiteley, Helen M. Byrne, David
J. Gavaghan**



Oxford Centre for Collaborative Applied Mathematics
Mathematical Institute
24 - 29 St Giles'
Oxford
OX1 3LB
England

Chaste: a test-driven approach to software development for physiological modelling

Joe Pitt-Francis^{*,a}, Pras Pathmanathan^{**a}, Miguel O. Bernabeu^a, Raf Bordas^a, Jonathan Cooper^a, Alexander Fletcher^c, Gary R. Mirams^b, Philip Murray^c, James Osborne^a, Alex Walter^c, S. Jon Chapman^c, Alan Garny^b, Inge M. M. van Leeuwen^e, Philip K. Maini^c, Blanca Rodríguez^a, Jonathan P. Whiteley^{a,c}, Helen M. Byrne^d, David J. Gavaghan^a

^a*Oxford University Computing Laboratory, Wolfson Building, University of Oxford, Parks Road, Oxford OX1 3QD, UK*

^b*Department of Physiology, Anatomy and Genetics, Sherrington Building, University of Oxford, Parks Road, Oxford OX1 3PT, UK*

^c*Mathematical Institute, 24–29 St. Giles, Oxford OX1 3LB, UK*

^d*School of Mathematical Sciences, University of Nottingham, University Park, Nottingham, NG7 2RD, UK*

^e*Division of Mathematics, University of Dundee, Dundee DD1 4HN, UK*

Abstract

Chaste is a software library and set of test suites for modelling phenomena in the domain of computational biology. Specifically these the modelling problems arise in the fields of cancer modelling, cardiac simulation and soft-tissue engineering (Chaste stands for ‘Cancer, heart and soft-tissue environment’). It is released under the LGPL 2.1 licence.

Chaste has been developed using agile programming methods. The project began 5 years when it was reasoned that the modelling of a variety of physiological phenomena requires both a generic mathematical modelling framework, and a generic computational/simulation framework. The Chaste project was formed as part of the larger Integrative Biology e-Science (IB) Project ([14]), an inter-institutional project aimed at developing a suitable IT infrastructure to support physiome-level computational modelling, with a focus on cardiac and cancer modelling.

PROGRAM SUMMARY

Manuscript Title: Chaste: a test-driven approach to software development for physiological modelling

Authors: Pitt-Francis, Pathmanathan *et al.*

Program Title: Chaste

Journal Reference: TBD

*Principal Corresponding author

**Corresponding author

Catalogue identifier: TBD

Licensing provisions: LGPL 2.1

Programming language: C++

Operating system: Unix

RAM: < 90 Megabytes for either of the scenarios described here (Bidomain slab or Cylindrical crypt simulation). Up to 16 Gigabytes (distributed across processors) for full resolution cardiac simulation.

Number of processors used: 1–64

Supplementary material:

Keywords: Cardiac electrophysiology, colorectal crypt simulation

PACS:

Classification: 3 Biology

External routines/libraries: PETSc, MPI, HDF5

Nature of problem:

Solution method: Coupled multiphysics with PDE, ODE and discrete mechanics simulation

Running time:

See results section (§6) for execution times and 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 computational biology. In this context, ‘high-quality’ means that the software is required to be extensible, robust, fast, accurate, maintainable and is required to use 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. 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 where possible.

In the rest of this section we briefly describe the agile programming approach used in Chaste with a view to informing the user as to how the process has affected the structure of the code. We then give the theory and mathematical modelling context behind two of the physiological problems which Chaste is able to model (§2). The two modelling domains are cardiac electrophysiology and colorectal crypt dynamics. The code-base is outlined first from a top-down view, showing how the libraries fit together (§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 (§4). After giving brief installation instructions (§5), we outline some results from the two application domains (§6) and conclude with a discussion on the next directions for the Chaste software (§7).

1.1. Test-driven development

From the first, the developers of Chaste decided to adopt an agile software engineering approach. 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, it is anticipated that savings will be made in the medium to long term when we reap the benefits of the features offered through this software engineering approach. 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, please 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 to rapidly discover, diagnose and fix bugs.

2. The application domain (Theoretical background)

Chaste was designed to be a multi-purpose library of solution techniques for a wide range of biological problems. While it is intended to be a generic extensible library, software development so far has focused on two main areas. These areas are the relatively mature area of cardiac electrophysiology (where the earliest mathematical models of heart cells date back to 1962 [3]) and the more speculative modelling area in tissue growth and cancer genesis. 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 the generic nature of the software. These two exemplars allow us to discuss future directions for the software and more generic uses of Chaste. Both build on a common base of core library components such as geometric meshes, and the numerical solution of ordinary differential equations (ODEs) and partial differential equations (PDEs) using state-of-the-art methods.

2.1. Mathematical modelling of cardiac electrophysiology

Tissue level cardiac electrophysiology is usually modelled using the bidomain equations, which comprise two partial differential equations (PDEs) coupled at each point in space with a system of ordinary differential equations (ODEs) [4]. In this section we first give an outline of how the equations are derived, and then briefly describe the algorithm used to compute the numerical solution of these equations.

2.1.1. The governing equations

We denote the region occupied by cardiac tissue by Ω . The volume of cardiac tissue occupied by cells—the intracellular space—is denoted by Ω_i . The remainder of 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 lengthscale for cells is shorter than the typical lengthscale of the features of the tissue level solution allows the use of homogenisation techniques [4]. 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} :

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

$$\nabla \cdot ((\sigma_i + \sigma_e) \nabla \phi_e + \sigma_i \nabla V_m) = 0, \quad (2)$$

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

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. Functional forms for I_{ion} and \mathbf{f} are determined by an electrophysiological cell model (see for example [8]).

The intracellular and extracellular currents, \mathbf{i}_i , \mathbf{i}_e , are given by

$$\begin{aligned} \mathbf{i}_i &= -\sigma_i \nabla \phi_i = -\sigma_i \nabla (V_m + \phi_e), \\ \mathbf{i}_e &= -\sigma_e \nabla \phi_e. \end{aligned}$$

Appropriate boundary conditions for Equations (1) and (2) are the specification of current applied across the boundary:

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

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

where \mathbf{n} is the outward pointing unit normal vector to the tissue, I_{s_i} is the intracellular current applied across the boundary, and I_{s_e} is the extracellular current applied across the boundary. The system of equations (1)–(5) are 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

Equations (1)–(3) are solved using the finite element method at a discrete set of times $t_0 < t_1 < \dots < t_N$. If the solution is known at time t_n we calculate the solution at time t_{n+1} using a semi-implicit method [5, 6] to solve Equations (1)–(2). This semi-implicit method uses an implicit approximation to the linear terms in Equations (1)–(2), and an explicit approximation to the non-linear I_{ion} term. This results in the linear system

$$A\mathbf{v} = \mathbf{b}, \quad (6)$$

where the matrix A is the same on each timestep 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 on each timestep. The entries of \mathbf{b} must be computed on each timestep. Computational efficiency for evaluating \mathbf{b} is 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 [7]. The system of ODEs at each point in space, Equation (3) is solved by a suitable numerical scheme. A typical ODE system, such as Luo-Rudy 1991 [8] can be described in the implementation-independent CellML format [9] (see §4.3.1), contains on the order of 10 differential equations (representing the rate of change of ionic concentrations and cell-membrane gates) and can be solved

using an ODE solver such as backward Euler. The exact choice of numerical method chosen 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. Details of the software engineering issues in implementing this algorithm have been described by Bernabeu *et al.*, [10].

2.2. Lattice-free model of tissue growth

In our second exemplar, our tissue growth model is an attempt to build a mathematical and computational model that bridges across spatial and temporal scales within a single, generic modelling framework ([11, 12]). The main focus of cancer Chaste is the initiation of colorectal cancer (CRC) in *intestinal crypts*, although the software has developed into a general tissue modelling framework and 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 mainly on the model of intestinal crypts.

Some sort of reference to [13] here, or in discussion.

CRC was originally chosen due to the availability of particularly good experimental data, and since the biological understanding of CRC is sufficiently advanced to allow such a systems level approach ([12]). 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.

CRCs 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 Figure 2.2. At the bottom of the crypt, a small population of stem cells is believed to divide continually, producing transit cells which divide several times before undergoing terminal differentiation. As new cells are added to the lower third of the crypt, the sheet of epithelial cells gradually 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. We take a discrete approach, modelling each cell individually. For simulation purposes, it is convenient to roll the crypt out onto a two-dimensional 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 (i) determine when cells are ready to divide and differentiate; and (iii) a mechanical model that controls cell migration at the macroscale. In this paper we use modules that are based on ([17, 18, 19]). 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. [20, 21, 22, 23])

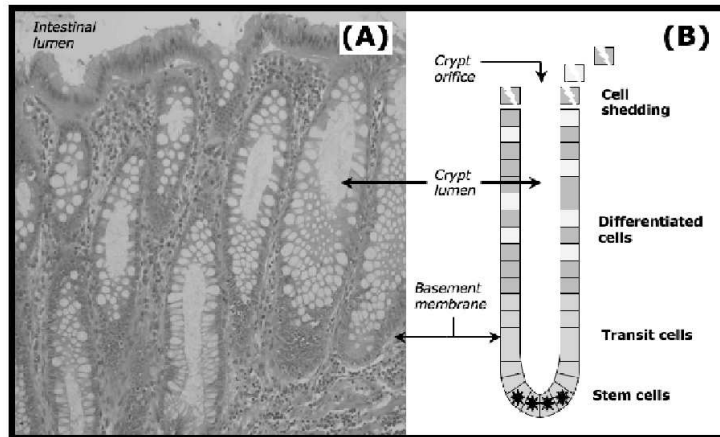


Figure 1: Normal colonic mucosa. **(A)** Microdissection image courtesy of the Department of Pathology, Ninewells Hospital, Dundee. The mucosa is characterised by the presence of numerous crypts of Lieberkühn. Murine and human intestinal crypts contain about 250 and 2000 epithelial cells, respectively ([15, 16]). **(B)** Schematic of a normal colonic crypt. These glands have the geometry of a ‘3D 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 protective barrier and nutrient absorption engine.

can be easily implemented and compared. The tumour spheroid model is also comprised of these three components, together with a fourth component: PDEs governing the diffusion and uptake of particular nutrients (for example, oxygen and glucose), with the cell model being heavily dependent on the local nutrient concentration.

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 ([24, 25]). Since cancer is a disease caused 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 nonlinear ODEs that characterise the change in the levels of key cell-cycle proteins ([19, 21]). In our multiscale 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—results from the presence of a gradient of extracellular Wnt factors (cf. [26]). 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 ([17]), formulated as a system of nonlinear 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 ([19]). 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. [27]), cell-centre ('point mass') models [28, 29, 30], and the non-point-mass vertex-based models [31]. In this paper we describe a tessellation-based cell-centre approach, in which the centres of adjacent cells are connected by linear springs ([18]). The main feature of this model is that a Delaunay triangulation ([32]) is performed on each timestep, 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

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

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

$$\mathbf{F}_i = \sum_j \mathbf{F}_{ij}, \quad (8)$$

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 $\nu \frac{d\mathbf{r}_i}{dt} = \mathbf{F}_i$, where ν is the cell viscosity, and models cell-substrate adhesion. This equation is discretised numerically using a forward Euler approach.

2.2.3. Coupling the Cell-Cycle and Mechanical Models

We couple the mechanical and cell-cycle models in the following way. When a cell divides, as determined by its internal cell-cycle model, a direction is chosen at random, and a new cell is placed a fixed distance away in that direction. The original cell is moved the same distance away in the opposite direction. As in ([18]) the rest length, s_{ij} , between a pair of cells which have just divided is initially set to be the distance between them, and allowed to increase linearly over the course of an hour. The rest-length between pairs of cells which have not just divided is assumed to be the typical diameter of a crypt cell. Thus, in the basic model, the cell-cycle model influences the mechanical model and, hence, cell movement, but not vice-versa. However, 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 §2 can be solved within Chaste, in this section we give an overview of the software structure. This overview 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

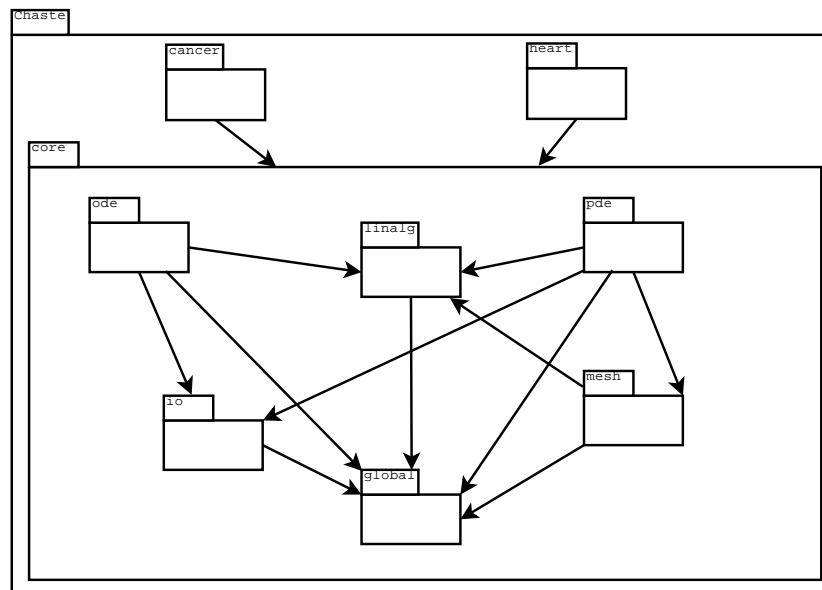


Figure 2: The structure of the code-base. Arrows represent dependencies between components. The ‘core’ component is merely a convenient shorthand for referring to the libraries shown within it. Both the heart and cancer libraries depend on all core libraries.

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 ([33]). 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. Figure 2 displays the library components of Chaste, which are described in detail in §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 lead to an easy route to automatically verify 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 CxxTest¹ which is a light-weight portable testing framework. A typical test suite file for a Chaste C++ class will include around a dozen individual test methods. Each method will instantiate and setup relevant classes, check that no exceptions are thrown and then check that any numerical output is within some given tolerance.

3.1.2. Numerical and parallel libraries

The design of the code and the decisions as to where to re-use existing libraries are constantly evolving with each software development iteration. However, there were certain software decisions 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 are able to 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 was expected to 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 from scratch.

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 PETSc² to provide sparse large-scale linear algebra routines and parallelism via MPI³.

¹cxxtest.tigris.org

²www.mcs.anl.gov/petsc/

³www.mcs.anl.gov/mpi/

Since the core linear algebra use 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 ([34]) 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 HDF5⁴ distributed data writing object which shows two orders of magnitude speedup over a home-written concentrator approach.

Also used throughout the code are: the Standard Template Library (for extensible vector structures and iterators); Boost⁵ uBLAS (for small scale sequential vectors and matrices); Boost serialisation for check-pointing and parameter definitions in the cancer-related code; and CodeSynthesis⁶ XSD for XML parameter input in cardiac-related code.

There were certain areas of the functionality where we decided initially not to use other libraries. Notably, we began by writing our own classes for small matrices and vectors (such as a ‘stencil matrix’ to be used in assembling a finite element problem into a PETSc linear algebra system). We also wrote our own parallel output routines and our own ODE solvers. As the code developed we experimented with the usability and speed of alternative solutions (such as HDF5 for parallel output and Boost uBLAS for small matrices). The speedups achieved via these two libraries were impressive enough for us to replace our original code with the library alternatives.

In the area of ODE solution, where it is required to solve large stiff nonlinear 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 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 Sundials⁷.

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 [35] or large-scale semi-automatic check-point library such as Condor [36],

⁴www.hdfgroup.org/HDF5/

⁵www.boost.org

⁶www.codesynthesis.com/products/xsd/

⁷www.llnl.gov/CASC/sundials/

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 physiological phenomena. 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 discovered *in silico* it can be reused to investigate a variety of treatment regimes. Added to these use-cases is the need for 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 relatively little check-point functionality: some, but not all, of the parameters in cell models can be saved to disk and the linear algebra systems can be written in PETSc format for investigation by another program. Check-pointing is an area of active development.

3.2. Top-down view of the libraries

Figure 2 shows a schematic of the library structure within Chaste. The mathematical modelling of both cardiac and cancer domains introduced in §2 rely heavily on (i) complex, realistic geometry provided primarily through unstructured finite-element meshes; (ii) conversion, using the finite element method [37] (FEM), 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. These components are supported by two lower-level libraries, `global` (which is responsible for initialisation and bookkeeping calls to 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 of 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 such a way that they verify correct functionality of the given component and that they entirely cover all lines of code in that component (together with some lines from lower-level components).

The build system used is Scons⁸. 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

⁸www.scons.org

set of tests. This system makes it easy for a developer to understand what effect 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 is able to 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 a limited amount of 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.

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 §2). These two 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 from straight propagation of electrical activity in a heart geometry using the bidomain equations. The result of such a study is illustrated by Figure 5 in §6. This type of simulation requires the instantiation of about 4.3 million Luo-Rudy cell ODE models and solution of a linear system with 8.6 million unknowns at every space step. Despite the complexity of such a simulation, it will be seen in §4.3 that the application programmer's code is straight-forward.

The second exemplar (§4.4) shows how a simple model of colorectal crypt turn-over 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 Figure 7 in §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 model the growth of a spheroid of cells (such as the early stages of tumour growth).

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 code-base. This will serve to give an indication of the firm foundation which is provided by the generic core code.

4.2. 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*. For example, the following code will use a given configuration file and override the duration of a cardiac simulation to 1ms:

```
HeartConfig::Instance()->SetParametersFile("./ChasteParameters.xml");
HeartConfig::Instance()->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. On the very first call to the `Instance()`, within the scope of a particular executable or test, a new object is created and filled with defaults; subsequent calls to `Instance()` return the same object which remains in scope for the duration of the program. Similar behaviour is seen in the cancer modelling code:

```
CancerParameters* inst = CancerParameters::Instance();
inst->SetSDuration(11.0);
```

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

Another notable singleton class is the random number generator. Using this class, which records how the 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 checkpoint and restarted.

4.2.1. Time stepping

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

```
TimeStepper my_stepper(start_time, end_time, timestep);
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 coder 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 having the final time step being smaller than others.

4.2.2. The unstructured mesh hierarchy

All mesh geometry within Chaste is based on Delaunay triangulations. There is a hierarchy of meshes inheriting 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) or a 2D mesh in 3D (say, to model a mono-layer 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 calls⁹; `ParallelTetrahedralMesh` which partitions the geometry using METIS 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.

4.2.3. The finite element assemblers

A hierarchy of finite element assemblers exists within the Chaste `pde` library. Their role is to take a mathematical description of a set PDEs together with their boundary conditions on a relevant mesh, to form a corresponding linear system and solve it. A straight-forward 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 Equation 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 Equation 6 need only be assembled once at the outset, with the right-hand-side vector \mathbf{b} assembled each timestep.

The family of PDE assemblers available within Chaste includes static and dynamic assemblers, assembly which use linear or quadratic basis functions (the latter using the `QuadraticMesh` class mentioned above), and linear and nonlinear assemblers (assemblers which are able to solve a nonlinear 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 a approximation to a nonlinear, dynamic, problem by constructing a class which inherits each of the required features.

⁹Triangle www.cs.cmu.edu/~quake/triangle.html and Tetgen tetgen.berlios.de/ provide re-meshing functionality in 2D and 3D respectively

4.3. Cardiac electrophysiology exemplar code

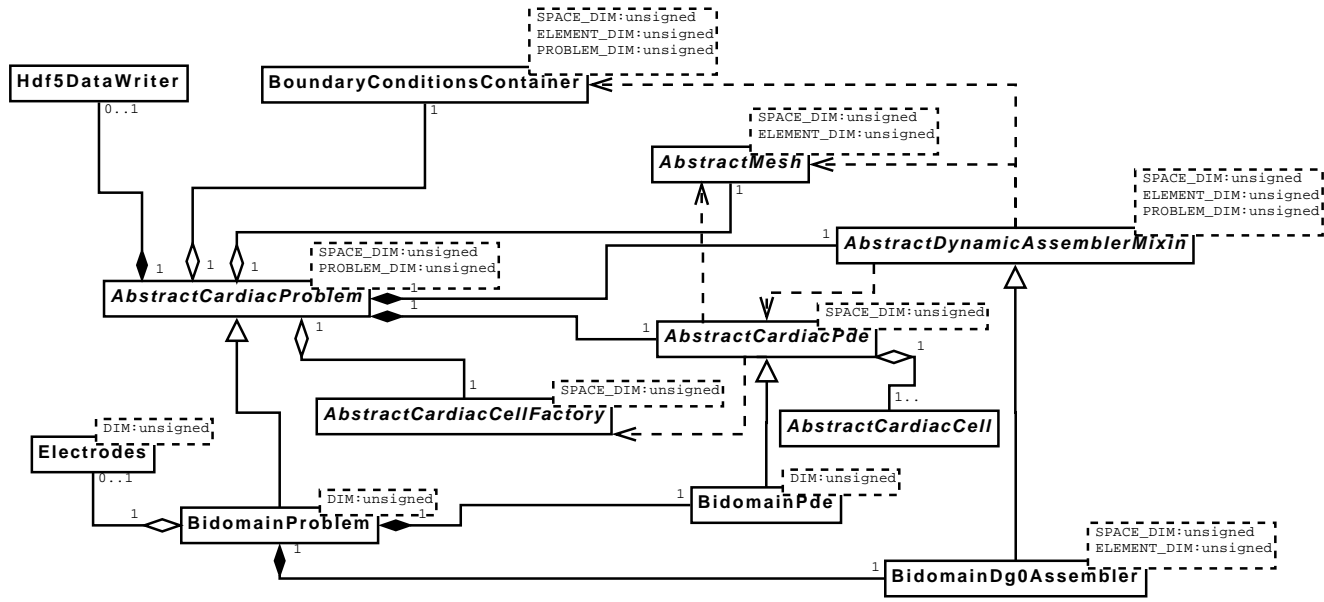


Figure 3: A UML diagram showing the primary classes involved in a cardiac electrophysiology simulation. The core class is **AbstractCardiacProblem** which collects together the various components of a simulation. It has two subclasses for monodomain and bidomain simulations; only the latter is shown 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 is periodically written via **Hdf5DataWriter** which is a light-weight wrapper to the HDF5 library.

4.3.1. 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 equation (1) and the size and complexity of the system of ODEs in equation (3) has grown substantially in models developed in recent years. These recent models typically consist of several separate currents, and are dependent on tens of quantities and also thus 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 this model without introducing errors in any of the constants is a near impossible task. In order to facilitate sharing and re-use of these models the CellML mark-up language was developed ([9]). 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 tool PyCml ([38]) for CellML verification and for automatic generation of efficient C++ code. For details of the current state of CellML, see [38] and the CellML website¹⁰.

4.3.2. 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` cell which visits every node in the geometry expecting to associate a (potentially stimulated) cell model to the node. Here is an example of a concrete 2-D instantiation 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 cell on the node close to the origin. Other nodes have unstimulated Luo-Rudy cells.

```
class PointStimulus2dCellFactory : public AbstractCardiacCellFactory<2>
{
private:
    SimpleStimulus *mpStimulus;

public:
    PointStimulus2dCellFactory() : AbstractCardiacCellFactory<2>()
    {
        mpStimulus = new SimpleStimulus(-6000.0, 0.5);
    }

    AbstractCardiacCell* CreateCardiacCellForTissueNode(unsigned nodeIndex)
```

¹⁰<http://www.cellml.org/>

```

    {
        double x = this->mpMesh->GetNode(nodeIndex)->rGetLocation()[0];
        double y = this->mpMesh->GetNode(nodeIndex)->rGetLocation()[1];
        if (fabs(x)+fabs(y)<1e-6)
        {
            /* Stimulated cell */
            return new LuoRudyIModel19910deSystem(mpSolver, mpStimulus);
        }
        else
        {
            /* The other cells have zero stimuli. */
            return new LuoRudyIModel19910deSystem(mpSolver, mpZeroStimulus);
        }
    }
}

```

4.3.3. 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 `BidomainCardiacProblem` class which, as can be seen from Figure 3, is a subclass of `AbstractCardiacProblem`. This allows for the solution of the simplified *monodomain* equations from much the same code as the bidomain equations.

The `BidomainCardiacProblem` class brings together the geometric mesh `AbstractMesh`, the cell factory in §4.3.2, the bidomain equations (Equations (1)–(2) in a `BidomainPde` class, the boundary conditions from Equations (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 is consistent. It then solves the problem using time loops in a PDE timestep, writing data out where required.

In order to elucidate the use of the singleton parameter class from §4.2 and the cell factory from §4.3.2 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 space.

```

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

```

4.4. Cancer exemplar code

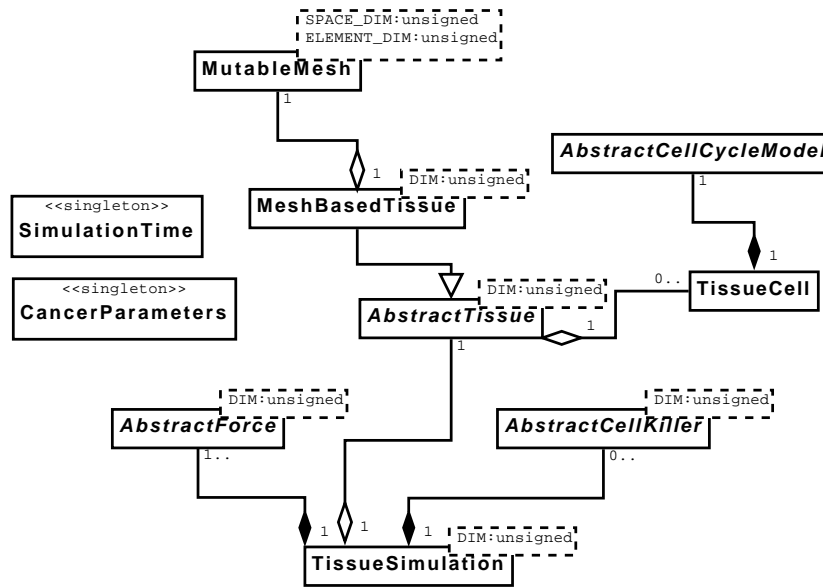


Figure 4: A UML diagram showing the primary classes involved in a cancer simulation. The tissue classes link a collection of cells (which are agnostic of spatial location) with geometry information, usually provided by a mesh. The `TissueSimulation` class combines this with force laws determining how 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.

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

4.4.1. 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 [17]). 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 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());
```

4.4.2. The cylindrical mesh and tissue classes

Crypt simulations on cylindrical geometries make use of a `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 a `HoneycombMeshGenerator` class, which creates a mesh where all connected nodes are 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 Figure 7 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` 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):

```
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.

4.4.3. Force objects

Various force objects have been defined, the most simple being `GeneralisedLinearSpringForce`, which provides the forces acting on each cell using Equation (7). This inherits from an `AbstractForce` class, which defines an interface for all force classes. In the simulations described in this paper we only consider a simple mechanical model where the total force on a particular cell arises wholly through linear interaction with neighbouring cells. However, more complex scenarios are possible, either through nonlinear interactions or by introducing addition terms on the right side of Equation (8) (for example, representing chemotactic or gravitational contributions to the total force). Further force classes have been implemented, each inheriting 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.

4.4.4. Cell-killer objects

Cell-killer objects contain rules which are used to determine which cells should be removed 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 to support them.

4.4.5. The tissue simulation classes

At the highest level are the tissue simulation classes, `CryptSimulation2d` and `TumourSpheroidSimulation`. 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 two-dimensional crypt or a three-dimensional tumour spheroid. The derived classes `CryptSimulation2d` and `TumourSpheroidSimulation` deal with crypt- and tumour-specific concerns only (such as, respectively, boundary conditions and integrating the PDE governing nutrient concentration). 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 `AbstractForces`. 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);

SloughingCellKiller<2> sloughing_cell_killer(&tissue);
```

```

CryptSimulation2d simulator(tissue, force_collection);
simulator.AddCellKiller(&sloughing_cell_killer);
simulator.SetEndTime(100); // 100 hours
simulator.Solve();

```

Tumour spheroid simulations are set up in a similar manner, as displayed 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 pointwise sink terms modelling cellular consumption of nutrient. `TumourSpheroidSimulation` contains an assembler class which solves the PDE each timestep by using the finite element method.

```

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 they allow myriad 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.

5. Installation instructions

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

Full installation of the library dependencies in order to create a development environment is non-trivial. However, we have provided a set of ‘README’ files in the release and we are developing a semi-automatic installation script.

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 bidomain equations in electrophysiology and on the cylindrical crypt model in cancer simulation.

6.1. Cardiac results

6.2. Bidomain wave propagation in realistic geometry

Miguel to supply more for this section Do we include slab mesh, s1-s2 etc.?

A study of bidomain propagation was carried out using new cardiac tissue geometry. This geometry ([?]) 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.

The `ApexStimulusHeartCellFactory` used in this code segment is similar to the cell factory introduced in §4.3.2 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 Figure 5.

```
HeartConfig::Instance()->Reset();
HeartConfig::Instance()->SetOdePdeAndPrintingTimeSteps(0.005, 0.01, 1);
HeartConfig::Instance()->SetSimulationDuration(10); //ms
HeartConfig::Instance()->SetOutputFilenamePrefix("ApexStimulus");
HeartConfig::Instance()->SetKSPSolver("symmlq");
HeartConfig::Instance()->SetKSPPreconditioner("bjacobi");

ApexStimulusHeartCellFactory<LuoRudyIModel19910deSystem> cell_factory;
BidomainProblem<3> cardiac_problem(&cell_factory);

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();
```

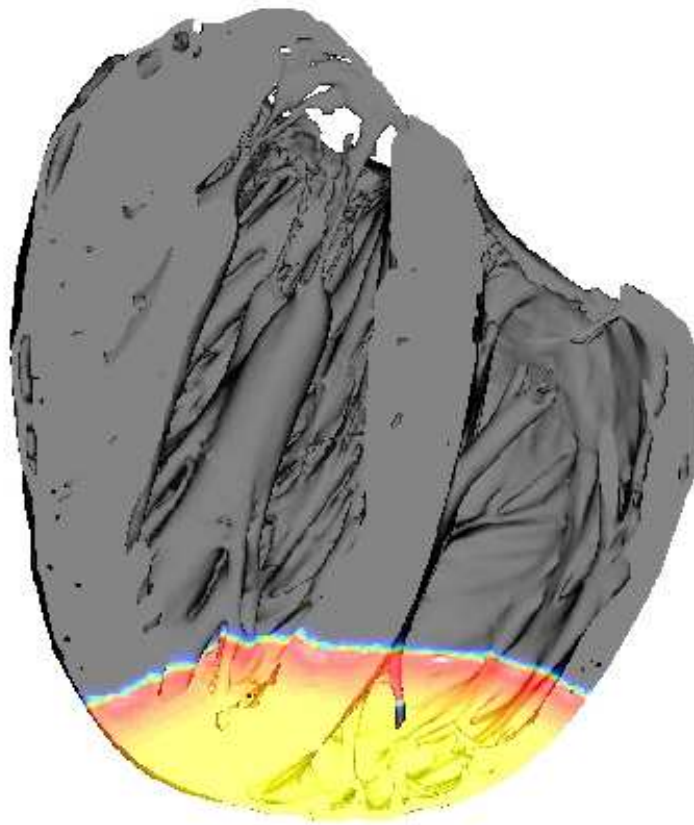


Figure 5: Bidomain propagation on a high-resolution rabbit ventricle model with 4.3 million nodes (8.6 million unknowns in PDE).

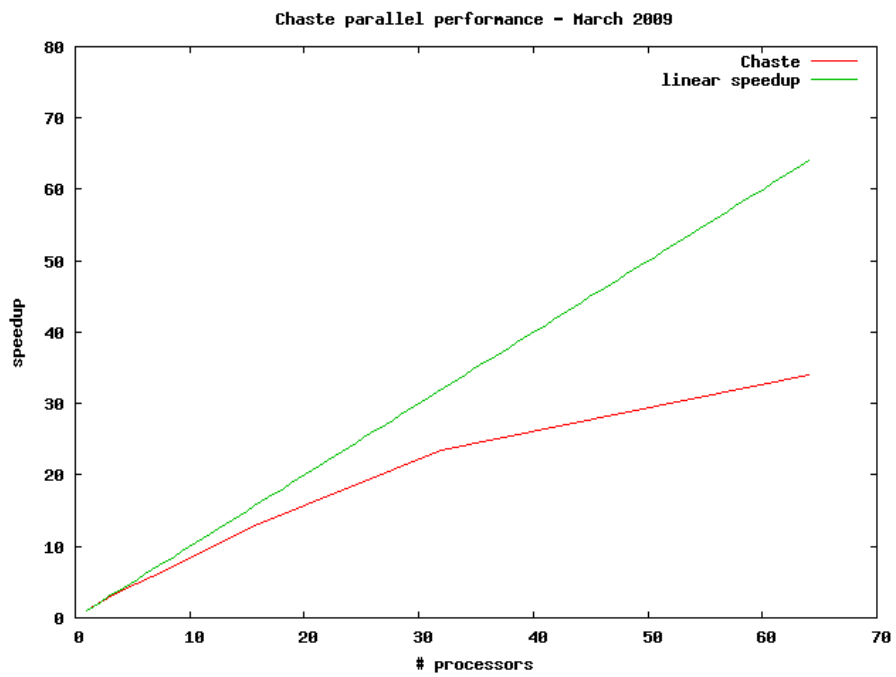


Figure 6: Speedup for a bidomain propagation experiment on the high-resolution model.

6.2.1. Parallel efficiency

Todo: Add some text to introduce Figure 6

6.3. Cancer results

6.3.1. Monoclonal conversion in crypts

Two theories have been proposed with regard to the stem cells in the crypt. According to the first hypothesis, multiple stem cells remain at the base of the crypt *ad infinitum* (this idea is incorporated in ([18]) 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 ([39, 40, 41]). 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 ([18]) 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 unpinned, it is no longer possible to distinguish between stem and transit cells, and no cells have a fixed position.

Figure 7 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 unpinned and a spatially varying Wnt stimulus is imposed along the crypt axis. Different cells are labelled at $t = 0$ 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.

6.3.2. 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. Figure 8 displays a number of snap-shots of the development of a two-dimensional tumour spheroid. In this simulation, the nutrient concentration is determined through solution of quasi-steady reaction-diffusion PDE, and, using the `OxygenBasedCycleModel`, a cell's progress through the cell-cycle depends on its oxygen uptake. A cell that

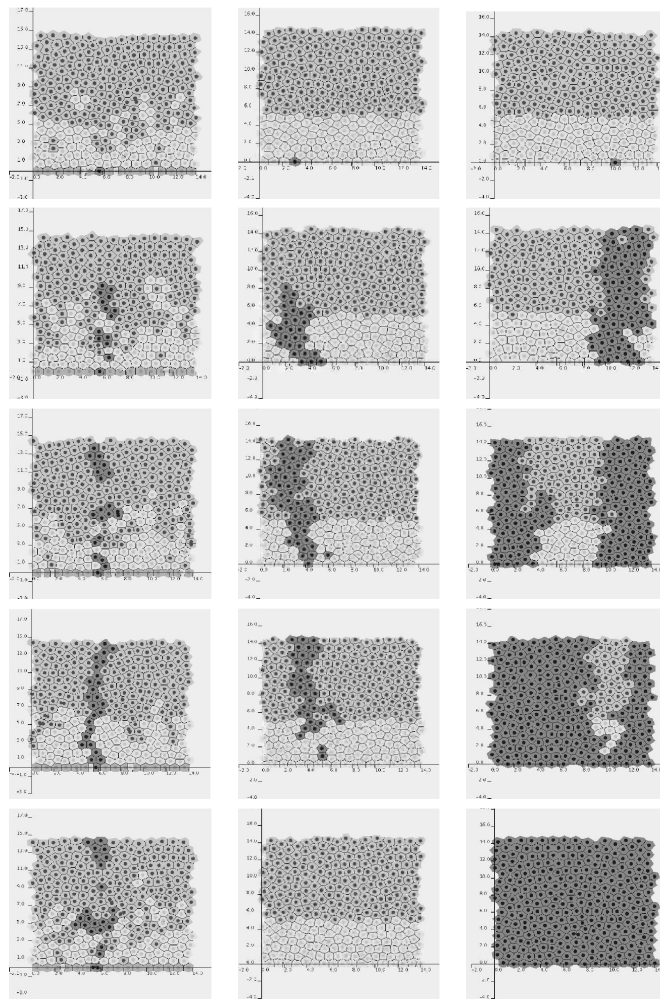


Figure 7: Clonal expansion in the crypt. Transit cells are light grey, differentiated cells are medium-dark grey, and labelled cells are dark grey. Stem cells, only present in the first column, are also medium-dark grey (the bottom row of cells in each image in column one). Top row: A cell at the base 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.

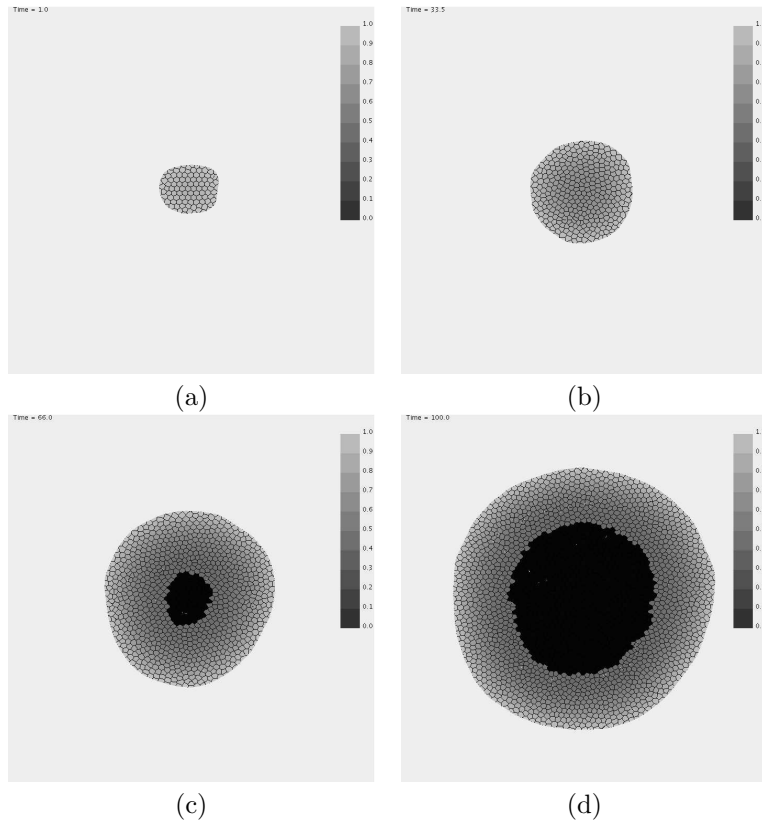


Figure 8: A two-dimensional tumour spheroid simulation. In all figures dead cells are shown in black, and living cells as grey, with the colour representing the oxygen concentration (lighter grey representing higher oxygen concentration). (a) $t = 3$ hours, (b) $t = 33.5$ hours, (c) $t = 66$ hours, (d) $t = 100$ hours.

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 affect their proliferative or apoptotic behaviour under hypoxic conditions.

7. Discussion

TODO

Brief discussion on cardiac electromechanical coupling and soft tissue elasticity.

Crypt dynamics. Crypt spheroids – Snapshots of Ozzy’s 3D Matlab visuals?

8. Acknowledgements

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

Chaste began thanks to the Engineering and Physical Sciences Research Council (EPSRC) of the UK under their eScience 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).

Some authors are supported by Award No. KUK-C1-013-04, made by King Abdullah University of Science and Technology (KAUST).

Thanks to Fujitsu Laboratories Europe for providing validation and timings necessary for Figure 6.

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, *Philosophical Transactions of the Royal Society 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] J. P. Keener, J. Sneyd, *Mathematical physiology*, New York: Springer, 1998.
- [5] J. P. Keener, K. Bogar, A numerical method for the solution of the bidomain equations in cardiac tissue., *Chaos* 8 (1998) 234–241.
- [6] J. P. Whiteley, An efficient numerical technique for the solution of the monodomain and bidomain equations, *IEEE Trans. Biomed. Eng.* 53 (2006) 2139–2147.

- [7] J. P. Whiteley, An efficient technique for the numerical solution of the bidomain equations, *Ann. Biomed. Eng.* 36 (2008) 1398–1408.
- [8] C.-H. Luo, Y. Rudy, A model of the ventricular cardiac action potential: Depolarization, repolarization, and their interaction, *Circ. Res.* 68 (1991) 1501–1526.
- [9] 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.
- [10] 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* (in press).
- [11] I. M. M. v. Leeuwen, H. M. Byrne, O. E. Jensen, J. R. King, Crypt dynamics and colorectal cancer: advances in mathematical modelling, *Cell Prolif* 39 (3) (2006) 157–181.
- [12] I. M. M. v. Leeuwen, C. M. Edwards, M. Ilyas, H. M. Byrne, Towards a multiscale model for colorectal cancer, *W J Gastroenterol* 13 (9) (2007) 1399–1407.
- [13] I. M. M. v. Leeuwen, G. R. Mirams, A. Walter, A. Fletcher, P. Murray, J. Osborne, S. Varma, S. J. Young, J. Cooper, J. M. Pitt-Francis, L. Mamtahan, P. Pathmanathan, J. P. Whiteley, S. J. Chapman, O. E. Jensen, J. R. King, P. K. Maini, S. L. Waters, D. J. Gavaghan, H. M. Byrne, An Integrative Computational Model for Intestinal Tissue Renewal, Cell Proliferation (in press).
- [14] D. J. Gavaghan, A. C. Simpson, S. Lloyd, D. F. M. Randal, D. R. S. Boyd, Towards a Grid infrastructure to support integrative approaches to biological research, *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Science* 363 (1883) (2005) 1829–1841.
- [15] C. S. Potten, C. Booth, D. Hargreaves, The small intestine as a model for evaluating adult tissue stem cell drug targets, *Cell Prolif* 36 (2003) 115–129.
- [16] Y. Yatabe, S. Tavaré, D. Shibata, Investigating stem cells in human colon by using methylation patterns, *Proc Natl Acad Sci U.S.A.* 98 (19) (2001) 10839–10844.
- [17] 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, *Journal Theoretical Biology* 247 (1) (2007) 77–102.

- [18] 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.
- [19] M. Swat, A. Kel, H. Herzl, Bifurcation analysis of the regulatory modules of the mammalian G₁/S transition, *Bioinformatics* 20 (10) (2004) 1506–1511.
- [20] M. Weliky, G. Oster, The mechanical basis of cell rearrangement. I. Epithelial morphogenesis during *Fundulus* epibody, *Development* 109 (1990) 373–386.
- [21] B. Novak, J. J. Tyson, A model for restriction point control of the mammalian cell cycle, *J. Theor. Biol.* 230 (2004) 563–579.
- [22] 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.
- [23] 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.
- [24] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, *Molecular Biology of the Cell*, Garland Science, New York, 2002.
- [25] D. O. Morgan, *The Cell Cycle. Principles of Control*, Oxford University Press, Oxford, 2006.
- [26] C. Gaspar, R. Fodde, APC dosage effects in tumorigenesis and stem cell differentiation, *Intl J Dev Biol* 48 (5–6) (2004) 377–386.
- [27] 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.
- [28] D. Drasdo, S. Holme, A single-cell-based model of tumor growth *in vitro*: monolayers and spheroids, *Physical Biology* 2 (3) (2005) 133–147.
- [29] 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.
- [30] E. Pallson, A three-dimensional model of cell movement in multicellular systems, *Future Generation Computer Systems* 17 (7) (2001) 835–852.
- [31] 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.

- [32] B. Delaunay, Sur la sphère vide, *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk* 7 (1934) 793–800.
- [33] 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.
- [34] J. Pitt-Francis, A. Garny, D. Gavaghan, Enabling computer models of the heart for high-performance computers and the grid, *Philosophical Transactions of the Royal Society A* 364 (1843) (2006) 1501 – 1516.
- [35] G. E. Fagg, J. Dongarra, FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world., *PVM/MPI 2000* (2000) 346–353.
- [36] 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.
- [37] J. N. Reddy, *An Introduction to the Finite Element Method*, McGraw-Hill, New York, 1993.
- [38] 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. Roy. Soc. Lond. A* 366 (2008) 3017–3043.
- [39] 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.
- [40] 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.
- [41] 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.

RECENT REPORTS

2009

01/09	A Mass and Solute Balance Model for Tear Volume and Osmolarity in The Normal And The Dry Eye	Gaffney Tiffany Yokoi Bron
02/09	Diffusion and permeation in binary solutions	Peppin
03/09	On the modelling of biological patterns with mechanochemical models: insights from analysis and computation	Moreo Gaffney Garcia-Aznar Doblare
04/09	Stability analysis of reaction-diffusion systems with time-dependent coefficients on growing domains	Madzvamuse Gaffney Maini
05/09	Onsager reciprocity in premelting solids	Peppin Spannuth Wettlaufer
06/09	Inherent noise can facilitate coherence in collective swarm motion	Yates <i>et al.</i>
07/09	Solving the Coupled System Improves Computational Efficiency of the Bidomain Equations	Southern Plank Vigmond Whiteley
08/09	Model reduction using a posteriori analysis	Whiteley
09/09	Equilibrium Order Parameters of Liquid Crystals in the Landau-De Gennes Theory	Majumdar
10/09	Landau-De Gennes theory of nematic liquid crystals: the Oseen-Frank limit and beyond	Majumdar Zarnescu
11/09	A Comparison of Numerical Methods used for Finite Element Modelling of Soft Tissue Deformation	Pathmanathan Gavaghan Whiteley
12/09	From Individual to Collective Behaviour of Unicellular Organisms: Recent Results and Open Problems	Xue Othmer Erban
13/09	Stochastic modelling of reaction-diffusion processes: algorithms for bimolecular reactions	Erban Chapman
14/09	Chaste: a test-driven approach to software development for physiological modelling	Pitt-Francis <i>et al.</i>

15/09	Block triangular preconditioners for PDE constrained optimization	Rees Stoll
16/09	From microscopic to macroscopic descriptions of cell migration on growing domains	Baker Yates Erban

Copies of these, and any other OCCAM reports can be obtained from:

**Oxford Centre for Collaborative Applied Mathematics
Mathematical Institute
24 - 29 St Giles'
Oxford
OX1 3LB
England**

www.maths.ox.ac.uk/occam