**Computational Tools for Cardiac Simulation**

GPU-Parallel Multiphysics

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Toby Simpson

under the supervision of
Rolf Krause

January 2022

Dissertation Committee

**Steven Niederer**      King's College London
**Gernot Plank**        Medical University of Graz
**Piotr Didyk**         Università della Svizzera italiana
**Michael Multerer**    Università della Svizzera italiana

Dissertation accepted on 9 January 2022

Research Advisor          PhD Program Director

**Rolf Krause**           **Walter Binder**

i

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Toby Simpson
Lugano, 9 January 2022

**Abstract**

Cardiovascular disease affects millions of people worldwide and its social and economic cost clearly motivates scientific research. Computer simulation can lead to a better understanding of cardiac physiology, and for pathology presents opportunities for low-cost and low-risk design and testing of therapies, including surgical and pharmacological intervention as well as automated diagnosis and screening.

Currently, the simulation of a whole heart model, including the interaction of electrophysiology, solid mechanics and fluid dynamics is the subject of ongoing research in computational science. Typically, the computation of a single heartbeat requires many processor hours on a supercomputer. The financial and ultimately environmental cost of such a computation prevents it from becoming a viable clinical or research solution.

We re-formulate the standard mathematical models of continuum mechanics, such as the Bidomain Model, Finite Strain Theory and the Navier-Stokes Equations, specifically for parallel processing and show proof-of-concept of a computational approach that can generate a complete description of a human heartbeat on a single Graphics Processing Unit (GPU) within a few minutes.

The approach is based on a Finite Volume Method (FVM) discretisation which is both matrix- and mesh-free, ideally suited to voxel-based medical imaging data. The solution of nonlinear ordinary and partial differential equations proceeds via the method of lines and operator-splitting. The resulting algorithm is implemented in the OpenCL standard and can run on almost any platform. It does not perform any CPU processing and has no dependence on third-party software libraries.

The implementation is simple and computationally cheap enough to be used as the kernel of more complex software. Used iteratively and in parallel across an array of GPUs, it would allow movement through a solution space of parameterised hearts in optimisation problems for parameter estimation, patient-specific fitting, or in the training of Artificial Neural Networks (ANN). The algorithm presents new opportunities in research and clinical practice and is readily extensible as a simulation tool for a broad range of multiphysics problems.

# Contents

# Part I

# Aim

# Chapter 1

# Introduction

## 1.1 Motivation

Cardiovascular disease affects millions of people worldwide, reducing both quality and duration of life. The high social and economic cost of the disease clearly motivate research into its treatment.

Computer simulation can replace physical procedures, reducing cost and removing associated risks. When simulation is performed accurately and cheaply with respect to the key computational resources of energy and time, then it can increase efficiency across a wide range of scientific and industrial disciplines.

In medical research and and clinical practice this advantage becomes even more apparent, since an *in-silico* medical intervention can be repeated and refined iteratively in a way that is not possible in the physical world. Similarly, the insight gained from experiment can be shared and re-used more effectively through repositories of standardised results. As technology improves such resources may be used to inform the automation of diagnosis and clinical decision-making.

## 1.2 Review

Even in its simplest form the simulation of a human heartbeat represents a significant computational challenge. We first try to define what constitutes a whole heart simulation by summarising some of the features that it might contain, most of which are common to the literature review which follows:

- Geometry that includes the torso, myocardium, four chambers of the heart,

proximal blood vessels and the valves that connect them. It may be extended to include pericardium and coronary blood supply although this is rarely present. The geometry may be schematic or generated from segmented medical imaging data.

- Microscopic Electrophysiology representing the cell membrane and ion channels that support a travelling wave of transmembrane potential through the myocardium, with anisotropic conductivity determined by tissue fibre directions. Models for this behaviour include the Bidomain and Monodomain reaction-diffusion equations [Keener and Sneyd, 2010] and ionic membrane currents described by the comprehensive [TenTusscher and Panfilov, 2006] or simplified Fitzhugh-Nagumo (FHN) [FitzHugh, 1955] [Nagumo et al., 1962] Ordinary Differential Equations (ODEs).

- Macroscopic Electrophysiology includes behaviour of larger electrophysiological structures such as the Sino-Atrial (SA) node, the Purkinje network and the Bundles of Bachmann and His.

- Excitation-Contraction (EC) coupling linking the action potential to actin and myosin filaments via intracellular Calcium ($Ca^{2+}$) dynamics that lead to contractile stress in the mechanically active myocardium. Models include [Nash and Panfilov, 2004] and [Courtemanche et al., 1998][O'Hara et al., 2011]. Mechano-Electric feedback also leads to additional ionic currents dependent upon extension and velocity [Trayanova et al., 2004].

- Hyperelastic Material models for the stress-strain relations of the myocardium include Mooney-Rivlin model [Mooney, 1940] [Rivlin and Rideal, 1948], Guccione [Guccione et al., 1995] Ogden [Ogden and Hill, 1972] [Holzapfel, 2000]. These are generally modified to include passive anisotropy and active contraction related to tissue fibre directions, as well as incompressibility, in accordance with experimental data.

- Fluid Dynamics of the blood modelled via the incompressible Navier-Stokes equations, sometimes modified to be non-Newtonian in small capillaries.

- Fluid-Structure Interaction (FSI) allowing changes in pressure and velocity to drive blood flow, which in turn feeds back onto solid tissue of the heart, valve leaflets and blood vessels. The problem may be strongly coupled with two-way feedback or weakly coupled in only one direction, usually with a prescribed solid motion. For the strongly-coupled dynamics there are two common approaches: firstly the Arbitrary Lagrangian Eulerian (ALE) [Hughes et al., 1981] formulation, and secondly the Immersed Boundary (IB) method [Kohl et al., 2001].

- Circulatory System has a significant effect on the behaviour of the heart with distal blood flow regulating pressure and velocity at the blood ves-

sels connected to the heart. The closed-loop systemic arterial tree and pulmonary system is usually modelled as a windkessel [Westerhof et al., 1971]. Some researchers use a similar system of ODEs within the heart chambers as a 0D substitute for the Fluid dynamics and FSI problems outlined above.

The following literature review considers some of the many publications that indicate the progress made towards the objective of whole heart simulation:

[Watanabe et al., 2004] Describes an idealised left ventricle and proximal aorta, with FHN electrophysiology, EC coupling, anisotropic Guccione constitutive relation, strongly coupled FSI via ALE form of the Navier-Stokes equations, and windkessel circulation. The problem is discretized and solved via the Galerkin Finite Element Method (FEM) on a tetrahedral mesh ( 100k DOF), to give a full description of fluid velocity, pressure and volume.

[Tang et al., 2007] Models left and right ventricles with respect to inflow and outflow after surgery. Using Navier-Stokes, ALE, FEM with a modified anisotropic Mooney-Rivlin model. Dynamic material parameters simulate systolic and diastolic tissue mechanics.

[Lee et al., 2009] Provides a review of multi-physics and coupled models, including strongly and weakly coupled FSI and gives results for an idealised ventricle at low and high resolution. He notes the computational intractability of fine model and considers the tradeoff between accuracy and computational cost.

[Formaggia, 2009] A book with very complete derivation of solid and fluid dynamics, and their coupling through the ALE formulation. The formulation and notation guides much of this work. Solution is via FEM and is subject to many simplifying assumptions. There is an anecdotal description of a stable algorithm for FSI time iteration.

[Nordsletten et al., 2011a] Studies an idealised ventricle with Finite elasticity, anisotropic fibre directions, parameters from ex-vivo studies, a model for calcium transfer, and a Lagrange multiplier form of ALE FSI coupling. FEM discretisation is solved with nonlinear Newton-Raphson iteration, no timings are given.

[Nordsletten et al., 2011b] is a key review paper for continuum mechanics, the ALE formulation, fibre directions giving the Guccione material model and electromechanical transfer, illustrated over single ventricle, discretised and solved with ALE FEM.

[Trayanova and Winslow, 2011] presents a review focussed mainly on electrophysiology and some electro-mechanical feedback for arrhythmias. Applications are given with respect to clinical, research and therapeutic applications.

[Khalafvand et al., 2011] Another review paper, considering Computational Fluid Dynamics (CFD) for valve leaflets. The work considers strongly and weakly coupled approaches including prescribed geometry with one-way coupling, ALE and fictitious methods such as the IB approach. There is no consideration of computational performance.

[Niederer et al., 2011a] Provodes a key benchmark study comparing the results of 11 submitted codes over the same electrophysiology problems. It forms the basis for the validation of electrophysiology in this work (see Chapter 6) and several others.

[Niederer et al., 2011b] A Bidomain electrophysiology study over a whole ventricle made with view to clinical use. The publication includes timings, introducing the metric of *real-time computational lag* (computation time divided by simulation time). The study uses a FEM tetrahedral mesh with width 0.25mm and 26 million Degrees of Freedom (DOF), a preconditioned Conjugate Gradient (CG) solver processed in parallel using the PETSc library [Balay et al., 1998] with Message Passing Interface (MPI) over 16384 cores. The result is a value of 240, or 1 second in 5 minutes and the authors note that 50% of the time is used for communication.

[Neic et al., 2012] gives the first consideration of GPU acceleration. An electrophysiology problem equivalent to the previous paper is accelerated at the lowest level, unrolling parallel loops within the PETSc library and passing to them to CUDA Fortran kernels. The strong scaling study reports a speedup of of 10 times using 20 GPUs on the coarsest grid. The authors note that this is achieved while *'minimally perturbing the code base'*.

[Sugiura et al., 2012] A prototype FEM ALE study with fine heart mesh with coarse torso mesh for ECG, Calcium and sarcomere dynamics via explicit ODE, automatic re-meshing for valves. Gives physiological values for ejection fractions, pressure and volume. RIKEN/Fujitsu k-computer with 88,128 CPUs, 6 hours per heartbeat.

[Land et al., 2015] Another key benchmark study which aggregates and 11 submitted mechanics code for simple problems. It is the basis for validation of the mechanics calculation in this work (see Chapter 6).

[Zhang et al., 2016] A review of multi-scale simulation including fluid mechanics. It specifies the different problems but does not demonstrate any solution.

[Quarteroni et al., 2017] A large and comprehensive paper gives full details of models and assembly and time integration for electrophysiology, electromechanical coupling, solid dynamics and FSI. Studies reproduce benchmarks electrophysiology, and mechanics test problems. An idealised ventricle is shown but without fully-coupled FSI, using pre-determined structure and 0D fluid rather

than a two-way solution. It focuses on convergence rather than performance concluding that a full model at all scales is *'still far beyond reach'*.

[Niederer et al., 2018] A broad overview of research and clinical applications written for a wide audience with many useful references.

[Santiago et al., 2018] A fully-coupled heart model including fluid mechanics over a complete geometry including atria and aortae and sliding pericardium. There is no Purkinje network, valves or atrial contraction. The ALE formulation refers to the prior literature for electrophysiology, solid and fluid models. Discretisation via in-house FEM code, with a mixture of forward, backward Euler and Newmark schemes, Generalized Minimal Residual (GMRES) method Picard method for velocity-pressure, FSI stabilised with quasi-Newton iterations. MPI-parallel solver alternates between CFD and electrophysiology/solid mechanics with 800k DOFs. The authors show qualitative results and a scaling study up to 4800 cores with speedup of 0.85, but do not publish wall clock timings, noting that this is a validation of the first instance of their model.

[Kaboudian et al., 2019] A GPU Lattice Bolzmann electrophysiology simulation based on WebGL Javascript API, demonstrating 2D/3D spiral waves, and low resolution rabbit ventricle at 1/3 real-time, also examples of fluid dynamics, and crystal formation. This is the first published attempt to re-formulate the problem specifically for GPU processing.

[Viola et al., 2020] Studies an idealised left heart with atrium, ventricle, aorta and mitral valve. Navier-Stokes is treated with a Finite Difference Method (FDM) on a cartesian mesh with circulatory effects prescribed as boundary conditions. FSI is via interpolated IB onto valve leaflets modelled as 2D mass-spring membranes with Adams-Bashforth integration for strong coupling. There is a Fung mechanical model [Fung, 1981] with fibre anisotropy, full bidomain with FHN for atria and TenTusscher-Panfilov electrophysiology for ventrcle. Bachman and His effects are prescribed as timed stimuli. The simulation is timed at 1500 CPU Hours per heartbeat (850ms of simulation).

[Gerach et al., 2021] Model a four-chamber geometry with a 0D closed-loop circulation, with the effects of ablation scar, solving Bidomain with EC and anisotropic Guccione material. The FEM discretisation mixes of implicit and explicit schemes, with activation of atria and ventricles prescribed via external stimulus. They note that experimentally derived parameters from literature must be adjusted to give physiological results. The mechanics mesh has 136k DOFs (Authors note that it is too coarse), but is finer for Electrophysiology 0.6mm. Results include PV loops for atria and ventricles, using a 2019 Apple iMac with 8 MPI processes requires 20-24 hours per heart beat.

[Augustin et al., 2021] Simulate a whole canine heart with a 0D lumped ODE circulation over 25 heartbeats. Electrophysiology uses Reaction Eikonal [Neic et al., 2017] approach. Mechanics via FEM with Newton-Raphson using GM-

RES on PETSc framework. Results include PV loops, activation maps and EGC. The mesh has 156k nodes at 1.25mm resolution and is processed on 256 cores requiring 30 minutes per heartbeat.

[Viola et al., 2022] Updates prior work on an idealised heart with GPU acceleration. The study includes a complete Fluid-Structure-Electrophysiology Interaction (FSEI) simulation using an in-house FEM library. An MPI-parallel in-house FEM library is processed with OpenACC-type compiler directives for CUDA Fortran acceleration. Weak and strong scaling studies are performed on 8 NVIDIA A100 cards showing 1-2 orders of magnitude speedup with respect to prior serial code. A single heartbeat requires 3-10 hours of processing depending on mesh resolution.

[Del Corso et al., 2022] is a similar paper from the same group as [Viola et al., 2022] which presents electrophysiology only, discretised with Finite Volume Model (FVM) and CUDA Fortran GPU acceleration of the GNRES solver. A Bidomain simulation at 0.5mm mesh width using the approach of [Rush and Larsen, 1978], requires 8 hours per heartbeat, with an equivalent monodomain requiring 1.5 hours.

[Verzicco, 2022] The most recent publication in the field is a comprehensive review paper giving the history of the field, with a focus on the inclusion of strongly-coupled FSI and its relevance for pathology. It serves as an ideal and modern introduction to the subject giving a fair representation of the state-of-the-art. It concludes with the following remark:

*'The high computational cost of complex multiphysics heart models and the need of trained researchers to run them constitutes an insurmountable barrier to the use of digital twins in clinical practice. Furthermore, they rely on massive parallel supercomputers, have a time-to-solution of the order of days and produce terabytes of data which need extensive post-processing to extract relevant information; this is totally incompatible with clinical decisions that must be taken, at most, within hours and obtained by commodity computers.'*

As research has progressed, research groups have reached a broad consensus on the mathematical formulation of the models that represent the features of a whole heart simulation and on the computational approach taken to solve them. The general method is summarised below:

- Model parameters are estimated from experiments carried out on small tissue samples, usually ex-vivo, from a variety of species.

- Medical imaging data is segmented and used to generate a tetrahedral mesh of the heart and surrounding tissue.

- The various differential operators related to the mathematical formulation are assembled into large in-memory linear systems via Finite Element discretisation.

- Integration in time is accomplished via a mixture of explicit and implicit methods. The hardest and most nonlinear part of the problem, fluid-structure interaction, is usually solved as a saddle-point sub-problem using Newton-type iterations.

- The solution of the many linear systems required for integration is carried out using well-established High Performance Computing (HPC) libraries implementing algorithms such as preconditioned CG or GMRES, amongst others.

- The computation of these algorithms requires the use of large supercomputing resources, dividing and balancing the problem over many (thousands) of processors.

From the review above, a generous estimate of processing time for complete simulation of a single heartbeat may be about an hour. Another rough estimate of the financial cost of computation is 1 US Dollar per node hour. The HPC simulations described in the publications above are therefore consuming (approximately) tens-of-thousands of Dollars per second of cardiac simulation.

Modern computer chip design has already hit the 'Power Wall' in which power consumption and heat dissipation are the limiting factors for processing speed. Parallel computing is thus standard and the cost-of-communication between processors has become the bottleneck for performance. Attempts to accelerate per-core performance via the addition of GPUs has shown some speedup but ultimately does not overcome this limitation.

Fitting a heart simulation to an idealised reference subject would potentially require the simulation of thousands of heartbeats. To then reproduce simulations for single pathologies would require thousands more. To do this on a per-patient basis for diagnosis would incur thousands more and thousands again for treatment. The turnaround time between and admission and a decision about treatment could not be met. We therefore conclude that the economic and environmental cost of energy consumption associated with current approaches does not make them feasible for clinical use. We also do not expect to see improvements in processor design that will allow for their practical application in the foreseeable future.

## 1.3   Objectives

The motivation for this work is therefore to re-design tools for cardiac simulation that can be applied in clinical or research situations. We therefore aim to provide *proof-of-concept* of a computational approach that is sufficiently accurate to give insight at a clinical or academic level but which is not prohibitive in terms of computational cost.

This work will deliver a piece of software that is designed to meet the following objectives:

- Provide a complete description of a heartbeat, representing Electrophysiology, Solid and Fluid dynamics and their interaction via FSI, as described above.

- Run on a geometry that represents the whole heart, which can be assimilated directly from medical imaging data.

- Allow for computation on a single small device, which is low-cost in terms of acquisition and energy consumption.

- Make use of a novel dual grid discretisation that allows for matrix-free computation and makes optimal use of GPU memory access.

- Run within seconds or minutes, allowing for repeated use over many heartbeats, or as the objective function for parameter estimation or patient model fitting.

- Apply the well-established mathematical formulations as above, without requiring new or unproven mathematical approaches.

- Provide results and biological indicators that are qualitatively similar to those of the studies outlined above.

- Be simple enough in design to achieve the required performance, but extensible to allow the inclusion of more complex mathematical models and features.

- Have no software dependency and thus give freedom of design to future developers and users.

- Be simple enough to be configured and run by third parties such as clinicians and researchers, or incorporated into more complex software. This document is designed to serve both as a thesis and software documentation.

The approach therefore starts from first principles, taking the simplest and most well-established models of solid and fluid dynamics and re-formulating them for parallel computation on a single small device, namely a Graphics Processing Unit (GPU). The formulation itself is based upon the Finite Volume Method combining the Lagrangian solid and Eulerian fluid formulations via the Arbitrary Lagrangian-Eulerian method (ALE).

Geometry is encoded on a deforming structured grid via a Signed Distance Function (SDF), and as such the approach can be considered *mesh-free*. The algorithm also uses a *matrix-free* approach, with each GPU process assembling the various linear operators by row as required. Nonlinearity is handled via *operator-splitting* in which the differential operators that drive the state of the

system are solved separately and in turn. The removal large linear systems and operator splitting allow the use of a *self-dual* grid which is ideal for both the FVM discretisation and the fast memory access of the GPU.

Since the design objectives are guided by constraints and compromise, then it follows that it can not fully reproduce the accuracy or completeness of the existing studies in the literature. The results are general in their nature since this is the first prototype of the software. They demonstrate the validity of the algorithm rather than the model itself. Some features are not present:

- Electro-mechanical coupling is simplified, with contraction coupled directly to the action potential. The correct model for Calcium dynamics can be added in future in a straightforward way.

- There is no model for the circulation or proximal blood vessels. The effects can be modelled modelled directly by adding them to the geometry with similar results.

- The current schematic heart does not include fully functioning valves. Their action requires an extra model for structure-structure interaction which is not currently present. This would is the subject of future work.

Given the objectives above, the work hopes in future to deliver software for practical use, rather than for purely academic investigation.

## 1.4   Overview

This document is laid out as follows:

Chapter 2 gives details of the mathematical models that will be employed, all of which are based upon the principles of continuum mechanics. The electrophysiology simulation includes the Mitchell-Schaffer ionic membrane model and the diffusive Bidomain and Monodomain models for the movement of charge through the myocardium and torso. Solid mechanics, including the deformation of tissue and muscle contraction, is formulated via Finite Strain theory using various stress-strain relationships for hyperelastic materials. The fluid mechanics of blood flow is described by the incompressible Navier-Stokes equations. Fluid-stucture interaction is makes use of the Arbitrary Lagrangian-Eulerian formulation.

Chapter 3 contains a brief introduction to GPU computing. Its aim is to show how architecture specialized for parallel processing can give advantages in computational efficiency. It will also to make clear the restrictions upon code design that arise as a result. This in turn guides the design of the algorithm.

Chapter 4 shows how the geometry and continuous differential operators required by the mathematical models of Chapter 2 are expressed in the discrete setting of a computational mesh, leading to the assembly and solution of the linear systems resulting from these operators.

Chapter 5 describes how the continuum mechanics models of Chapter 2 and discrete operators of Chapter 4 are arranged for parallel computation, under consideration of the design constraints of Chapter 3. It details the relationships between computational kernels and the mathematical formulation as well as memory use and therefore serves as documentation for the computer code.

Chapter 6 details the numerical experiments which are used to validate the quality of the discretisation of the operators in Chapter 4 and their combination and integration in Chapter 5. These include numerical convergence studies with respect to analytic solutions for the differential operators and the reproduction of benchmark studies for the simulation of electrophysiology and solid mechanics.

Chapter 7 demonstrates the application of the software to cardiac simulation. It includes qualitative studies of whole heart function as well as quantitative results such as a synthetic Electrocardiogram (ECG) and Pressure-Polume (PV) loop.

Chapter 8 provides a review of the work, considering its strengths and weaknesses and provides an outlook for future improvement and applications.

# Chapter 2

# Cardiac Multiphysics

During the normal heartbeat a wave of membrane depolarisation propagates at the Sino-Atrial (SA) node in the Right Atrium (RA) and travels through the myocardium in an orderly fashion, first to the Left Atrium (LA) and then to the Right and Left Ventricles (RV, LV). The tissue has a property of anisotropic conductivity that is dependent upon both the orientation of the muscle fibres and the presence of tissues of varying conductivity such as the Pukinje Fibre network and the Bundle of His.

As this travelling wave of membrane depolarisation passes through the heart, changes in ion channels and receptor proteins within the membranes of the myocites mediate intracellular ionic changes that lead to a sequence of co-ordinated muscle contraction.

The deformation resulting from movement of the contractile tissue results in forces that change the pressure in the chambers of the heart. These pressure changes, together with viscous forces and advection cause the movement of blood through the heart and into surrounding vessels. The unidirectional flow and hence circulation of blood is maintained by the sequential contraction of the chambers and by valves within the heart and circulatory system that prevent the reversal of flow.

This chapter gives a mathematical description of the processes outlined above.

## 2.1  Electrophysiology

Membrane depolarisation initiated at the SA node is propagated as a travelling wave through the myocardium via two mechanisms:

Firstly, active processes maintain chemical potential gradients across the mem-

branes of myocardial cells. Changes in charge, and hence transmembrane potential, cause the opening and closing of voltage-dependent channels that allow ions ($Na^+$, $K^+$, $Ca^+$, $Cl^-$) to flow across the membrane, further depolarising it on a feedforward manner, until the ionic potential gradients have reached an equilibrium. Thereafter there is a refractory period during which the membrane can not be excited until the potential gradient has been restored by the pumping of ions back across the cell membrane.

Secondly, the changes in potential caused by membrane depolarisation cause the diffusion of electrical charge between an excited cell and its neighbours. This in turn stimulates ion channels as described above and causes ionic excitation to spread. The refractory period ensures that a recently excited cell is unresponsive to further depolarisation and thus the unidirectional progress of the travelling wave is maintained.

The Partial Differential Equation (PDE) that models the spreading cycle of depolarisation and repolarisation is a *reaction-diffusion* equation. We consider the reactive Mitchell-Schaeffer membrane model and the diffusive Bidomain and Monodomain equations.

### 2.1.1   Mitchell-Schaeffer Model

The model of [Mitchell and Schaeffer, 2003] describes a *uniformly polarized membrane patch* representing an action potential in a spatially clamped ventricular myocite. It considers a a pair of coupled time-dependent Ordinary Differential Equations (ODE)s with time $t \in \mathbb{R}^+$, for a dimensionless transmembrane voltage $v \in [0,1]$ and gating variable $w \in [0,1]$. The voltage $v$ is determined by the first ODE

$$\frac{dv}{dt} = J_{\text{in}}(v,w) + J_{\text{out}}(v) + J_{\text{stim}}(t) \tag{2.1.1}$$

where the three currents $J$ are defined as follows: The inward current $J_{\text{in}}$ combines ionic movements ($Na^+$, $Ca^+$) that raise the membrane voltage $v$. The feedforward nature of the ion channels gives it a cubic dependence on the membrane voltage $v$ and a linear dependence on the gating variable $w$, scaled by a constant time parameter $\tau_{\text{in}}$:

$$J_{\text{in}}(v,w) = \frac{1}{\tau_{\text{in}}} v^2 (1-v) w. \tag{2.1.2}$$

The outward current $J_{\text{out}}$ combines those ions that lower membrane voltage (primarily $K^+$) and depends negatively on $v$ but is not gated, scaled by a time constant $\tau_{\text{out}}$

$$J_{\text{out}}(v) = -\frac{1}{\tau_{\text{out}}} v. \tag{2.1.3}$$

The stimulus current $J_{\text{stim}}$ can applied by the experimenter.

The second ODE for the gating variable $w$ is as follows:

$$\frac{dw}{dt} = \begin{cases} \dfrac{1-w}{\tau_{\text{open}}} & \text{, if } v < v_{\text{gate}} \\ \dfrac{1-w}{\tau_{\text{close}}} & \text{, if } v \geq v_{\text{gate}} \end{cases} \tag{2.1.4}$$

where $\tau_{\text{open}}$ and $\tau_{\text{close}}$ are time constants and $v_{\text{gate}}$ is a threshold voltage for activation. The nonlinear and opposing nature of the equations for $v$ and $w$ place the system into the class of *activator-inhibitor* ODEs.

The authors note the relationships between their model and those of [Fenton and Karma, 1998] , [Luo and Rudy, 1994] and [FitzHugh, 1955] [Nagumo et al., 1962], and point out two improvements: Firstly it contains four physically meaningful constants, corresponding to the four phases of the action potential: initiation, plateau, decay and recovery. Secondly, that it avoids the non-physical voltage overshoot of its predecessors. It is also simpler than the more comprehensive [ten Tusscher et al., 2004]. The parameters used by the model have been experimentally fitted into a physical range [Ngoma et al., 2017].

An example of a numerical solution of the system is shown in Figure 2.1. It is worth noting that the equations are highly nonlinear or *stiff*, given the steep rise in onset membrane voltage. As such they illustrate a key point in the design of this work. The nonlinearity makes it too costly to solve the system implicitly. It is acceptable therefore to perform an explicit time integration and accept a maximum time step for stability.
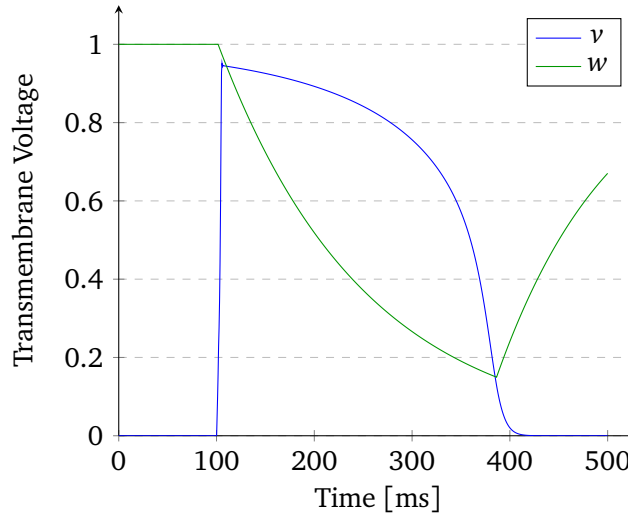


Figure 2.1: Mitchell-Schaeffer action potential, Explicit Euler $\Delta t = 0.5$ms.

### 2.1.2  Bidomain Model

Given a description of the action potential in time we next consider the movement of the traveling wave in space. The Bidomain model considers a small region of tissue as a single point $\mathbf{x} \in \mathbb{R}^d$ where $d \in \mathbb{N}$ is the dimension, that contains both intra- and extra-cellular spaces, along with variables for the potential differences and current flows between them. The propagation of electrical activity from one cell to another is represented as a diffusion of ionic charge. This construction leads to a system of PDEs which govern the movement of charge both through the <u>i</u>ntracellular and <u>e</u>xtracellular spaces of the heart and also into the surrounding tors<u>o</u> (underlined characters correspond to variable subscripts). For this reason the Bidomain model was instrumental in the development of defibrillation. The standard reference for this material is the book of [Keener and Sneyd, 2010].

The model is derived as follows, we consider a point in space $\mathbf{x} \in \mathbb{R}^d$, an element of the following sets:

| | |
|---|---|
| $\mathbb{H}$ | heart tissue with boundary $\partial\mathbb{H}$ |
| $\mathbb{T}$ | surrounding tissue of the body with boundary $\partial\mathbb{T}$ |

For a point in $\mathbb{H}$:

| | | |
|---|---|---|
| $v$ | $\in \mathbb{R}$ | transmembrane voltage |
| $v_i, v_e$ | $\in \mathbb{R}$ | intracellular and extracellular voltage |
| $\Sigma_i, \Sigma_e$ | $\in \mathbb{R}^{d \times d}$ | intracellular and extracellular conductivity tensors |
| $J_i, J_e, J_t$ | $\in \mathbb{R}^d$ | intracellular, extracellular and transmembrane current density |
| $C_m$ | $\in \mathbb{R}$ | membrane capacitance per unit area |
| $G_m$ | $\in \mathbb{R}$ | membrane conductivity per unit area |
| $\chi_m$ | $\in \mathbb{R}$ | membrane surface area to volume ratio |
| $I_{\mathrm{m}}$ | $\in \mathbb{R}$ | ionic current over membrane per unit area |

For a point in $\mathbb{T}$:

| | | |
|---|---|---|
| $J_o$ | $\in \mathbb{R}^d$ | tissue current density |
| $v_o$ | $\in \mathbb{R}$ | tissue voltage |
| $\Sigma_o$ | $\in \mathbb{R}^{d \times d}$ | tissue conductivity tensor |

#### 2.1.2.1  Derivation

First apply the current-voltage relationship over the intracellular and extracellular domains using Ohm's law:

$$J_i = -\Sigma_i \nabla v_i \tag{2.1.5}$$
$$J_e = -\Sigma_e \nabla v_e \tag{2.1.6}$$

Then require that there is no accumulation of charge in $\mathbb{H}$, setting the current densities equal and opposite in sign:

$$-\nabla \cdot J_i = \nabla \cdot J_e = \chi_m I_m \tag{2.1.7}$$

Substituting gives the first model equation, which states that all current exiting one domain must enter the other:

$$\nabla \cdot (\Sigma_i \nabla v_i) + \nabla \cdot (\Sigma_e \nabla v_e) = 0 \tag{2.1.8}$$

By convention, the transmembrane current $J_t$ and voltage $v$ are measured with respect to the intracellular domain:

$$J_t = \nabla \cdot (\Sigma_i \nabla v_i) = -\nabla \cdot (\Sigma_e \nabla v_e) \tag{2.1.9}$$
$$v = v_i - v_e \tag{2.1.10}$$

The model for transmembrane current depends on the dynamic behaviour of the membrane and is based on the cable equation:

$$J_t = \chi_m \left( C_m \frac{\partial v}{\partial t} + I_{\mathrm{m}} \right) \tag{2.1.11}$$

The ionic current $I_{\mathrm{ion}}$ is where the Mitchell-Schaeffer model enters the Bidomain equations. Scaled by membrane conductivity $\chi_m$ it models the dynamic properties of the transmembrane voltage. The expressions for $J_t$ are combined for the second model equation:

$$\nabla \cdot (\Sigma_i \nabla v_i) = \chi_m \left( C_m \frac{\partial v}{\partial t} + I_{\mathrm{m}} \right) \tag{2.1.12}$$

### 2.1.2.2 Boundary Conditions

We next repeat a similar process with the body tissue domain $\mathbb{T}$, first applying Ohm's law for the current-voltage relationship:

$$J_o = -\Sigma_o \nabla v_o \tag{2.1.13}$$

then preventing the accumulation of charge in $\mathbb{T}$:

$$\nabla \cdot (\Sigma_o \nabla v_o) = 0 \tag{2.1.14}$$

Next, we electrically isolate the body tissue $\mathbb{T}$ by setting current flow to zero in the unit normal direction $\mathbf{n} \in \mathbb{R}^d$ normal to $\partial \mathbb{T}$:

$$(\Sigma_o \nabla v_o) \cdot \mathbf{n} = 0, \quad \mathbf{x} \in \partial \mathbb{T} \tag{2.1.15}$$

Couple the extracellular voltage $v_e$ to the tissue voltage $v_o$ on the boundary of the heart $\partial \mathbb{H}$:

$$v_e = v_o, \quad \mathbf{x} \in \partial \mathbb{H} \tag{2.1.16}$$

Couple the current flow from the extracellular to the tissue domain on $\partial \mathbb{H}$:

$$(\Sigma_e \nabla v_e) \cdot \mathbf{n} = (\Sigma_o \nabla v_o) \cdot \mathbf{n}, \quad \mathbf{x} \in \partial \mathbb{H} \tag{2.1.17}$$

Isolate the intracellular domain from the body tissue:

$$(\Sigma_i \nabla v_i) \cdot \mathbf{n} = 0, \quad \mathbf{x} \in \partial \mathbb{H} \tag{2.1.18}$$

Grouping these equations gives the fully determined Bidomain system:

$$
\begin{aligned}
\nabla \cdot (\Sigma_i \nabla v_i) &= \chi_m \left( C_m \frac{\partial v}{\partial t} + I_\mathrm{m} \right) & \mathbf{x} \in \mathbb{H} \\
\nabla \cdot (\Sigma_i \nabla v_i) + \nabla \cdot (\Sigma_e \nabla v_e) &= 0 & \mathbf{x} \in \mathbb{H} \\
\nabla \cdot (\Sigma_o \nabla v_o) &= 0 & \mathbf{x} \in \mathbb{T} \\
\mathbf{n} \cdot (\Sigma_o \nabla v_o) &= 0 & \mathbf{x} \in \partial \mathbb{T} \\
\mathbf{n} \cdot (\Sigma_e \nabla v_e) - \mathbf{n} \cdot (\Sigma_o \nabla v_o) &= 0 & \mathbf{x} \in \partial \mathbb{H} \\
(\Sigma_i \nabla v_i) \cdot \mathbf{n} &= 0 & \mathbf{x} \in \partial \mathbb{H}
\end{aligned}
\tag{2.1.19}
$$

### 2.1.3   Monodomain Model

The Monodomain model is a simplification of the Bidomain model with a single anisotropy field for both the intra and extracellular spaces. Continuing from the definition of the Bidomain model (2.1.19) it makes the assumption that the intracellular $\Sigma_i$ and extracellular $\Sigma_e$ conductivity tensors are equal up to scaling. The two anisotropy fields are thus combined via a constant $\lambda \in \mathbb{R}$ which gives the intracellular to extracellular conductivity ratio:

$$\Sigma_e = \lambda \Sigma_i \tag{2.1.20}$$

From the derivation of the Bidomain model we take expressions (2.1.9) for transmembrane current $J_t$ and transmembrane voltage $v$:

$$
\begin{aligned}
J_t &= \nabla \cdot (\Sigma_i \nabla v_i) = -\nabla \cdot (\Sigma_e \nabla v_e) \tag{2.1.21} \\
v &= v_i - v_e \tag{2.1.22}
\end{aligned}
$$

Combining them via the conductivity ratio $\lambda$ and re-arranging gives the formulation of the Monodomain model:

$$\frac{\lambda}{1+\lambda} \nabla \cdot (\Sigma \nabla v) = \chi_m \left( C_m \frac{\partial v}{\partial t} + I_\mathrm{m} \right). \tag{2.1.23}$$

The equation (5.3.2) calculates the change in membrane potential difference $v$ with respect to time $t$. The reaction term on the right is the transmembrane

current $I_{\text{ion}}$ generated by the membrane potential difference. The diffusion term on the left is the flow of current into neighbouring regions, where $\boldsymbol{\Sigma}$ is the anisotropic diffusivity tensor that relates potential gradients $\nabla v$ to current flow. There are also constants $\chi_m$, a surface area to volume ratio for the cell membrane, and $C_m$ the membrane capacitance.

It remains to quantify the conductivity tensor $\boldsymbol{\Sigma}$. In practice our model defines a vector field $\mathbf{f} \in \mathbb{R}^d$ of unit length fibre directions as well as longitudinal and transversal conductivities $\sigma_L, \sigma_T \in \mathbb{R}$ respectively, from which the conductivity tensor is derived via an outer product as follows,

$$\boldsymbol{\Sigma} = \sigma_T \mathbb{I} + (\sigma_L - \sigma_T)\mathbf{f} \otimes \mathbf{f}. \tag{2.1.24}$$

The same field of fibre directions also determines the stresses arising from muscular contraction.

## 2.2 Continuum Mechanics

### 2.2.1 Deformation

The movement and interaction of both the solid and fluid materials of the heart are described within the formalism of continuum mechanics, effectively a set of scalar, vector and tensor fields defined over time in physical space. This brief introduction follows [Gonzalez and Stuart, 2008], [Reddy, 2013] and [Formaggia, 2009].
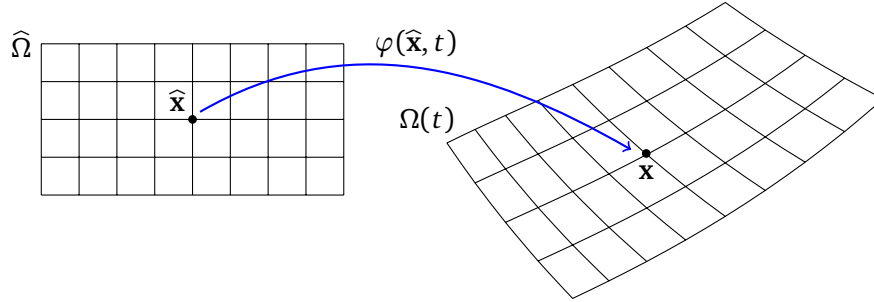


Figure 2.2: The reference $\widehat{\Omega}$ and deformed $\Omega(t)$ configurations, with a material point $\widehat{\mathbf{x}}$ and its current image $\mathbf{x}$ under the motion $\varphi(\widehat{\mathbf{x}}, t)$.

Figure 2.2 shows a body in its reference configuration $\widehat{\Omega} \subset \mathbb{R}^d$, described by material coordinates $\widehat{\mathbf{x}} \in \mathbb{R}^d$ which undergoes a smooth admissible deformation over time $t \in \mathbb{R}$, called a motion $\varphi : \mathbb{R}^d \times \mathbb{R} \mapsto \mathbb{R}^d$, into a current configuration $\Omega(t) = \Omega_t \subset \mathbb{R}^d$ described by spatial coordinates $\mathbf{x} \in \mathbb{R}^d$.

This allows the definition of the following fields:

$$\text{Motion} \qquad \mathbf{x} = \varphi(\widehat{\mathbf{x}}, t)$$

$$\text{Velocity} \qquad \mathbf{v} = \frac{\partial \varphi}{\partial t}(\widehat{\mathbf{x}}, t) = \frac{\partial \mathbf{x}}{\partial t}$$

$$\text{Deformation gradient} \quad \mathbf{F} = \frac{\partial \varphi}{\partial \widehat{\mathbf{x}}}(\widehat{\mathbf{x}}, t) = \frac{\partial \mathbf{x}}{\partial \widehat{\mathbf{x}}}$$

$$\text{Jacobian determinant} \quad J = \det \mathbf{F}(\widehat{\mathbf{x}}, t).$$

$$(2.2.1)$$

The Lagrangian description gives a field with respect to the material coordinates and the Eulerian description is made with respect to the spatial coordinates. The motion gives the relationship between the two descriptions, for instance for the density $\rho : \mathbb{R}^d \times \mathbb{R} \mapsto \mathbb{R}$,

$$\rho(\mathbf{x}, t) = \widehat{\rho}\left(\varphi^{-1}(\mathbf{x}, t), t\right). \tag{2.2.2}$$

Key to the formulation are the deformation gradient $\mathbf{F}$ and its determinant $J$. Due to its regularity we can express the motion as a Taylor series for some vector $d\widehat{\mathbf{x}}$:

$$\varphi(\widehat{\mathbf{x}} + d\widehat{\mathbf{x}}, t) = \widehat{\mathbf{x}} + \mathbf{F}(\widehat{\mathbf{x}}, t)d\widehat{\mathbf{x}} + \mathcal{O}(d\widehat{\mathbf{x}}^2). \tag{2.2.3}$$

The change in length of a vector under the motion is thus approximated by:

$$\|d\widehat{\mathbf{x}}\|_2 \approx \sqrt{d\widehat{\mathbf{x}}^\top \mathbf{F}(\widehat{\mathbf{x}}, t)^\top \mathbf{F}(\widehat{\mathbf{x}}, t)d\widehat{\mathbf{x}}}. \tag{2.2.4}$$

For a subvolume $\widehat{\Omega} \subset \mathbb{R}^d$ and its image $\Omega_t$ the determinant $J = \det(\mathbf{F}(\widehat{\mathbf{x}}, t))$ gives a relationship for volume between the reference and deformed configurations.

$$|\Omega| = \int_\Omega d\mathbf{x} = \int_{\widehat{\Omega}} J d\widehat{\mathbf{x}} \tag{2.2.5}$$

Nanson's formula relates surface area in the reference and deformed configurations, where $\widehat{\mathbf{n}}$ and $\mathbf{n}$ are the two unit normals respectively:

$$|\partial\Omega| = \int_{\partial\Omega} \mathbf{n} \, dA = \int_{\partial\widehat{\Omega}} J\mathbf{F}^{-\top}\widehat{\mathbf{n}} \, d\widehat{A}. \tag{2.2.6}$$

It combines the two previous results, effectively dividing volume by length for surface area. Transposition arises since normals are orthogonal to the surface and integration thus involves a dot product.

## 2.2.2   Conservation Laws

A complete description of a continuum body is based upon a set of conservation laws for mass, momentum and energy, as well as laws for intertia and thermodynamics. In order to continue we give expressions for the conserved quantities

of mass $m \in \mathbb{R}$ and linear momentum $\mathbf{l} \in \mathbb{R}^d$ for a body $\Omega \subset \mathbb{R}^d$.

$$m = \int_\Omega \rho(\mathbf{x}, t) \, dV \tag{2.2.7}$$

$$\mathbf{l} = \int_\Omega \rho(\mathbf{x}, t)\mathbf{v}(\mathbf{x}, t) \, dV. \tag{2.2.8}$$

### 2.2.3  Cauchy's Theorem

A body in a continuum can be subject to two types of force: body forces $\mathbf{b}(\mathbf{x}) \in \mathbb{R}^d$ such as gravity, act per unit volume, whereas traction forces $\mathbf{t}(\mathbf{n}, \mathbf{x}) \in \mathbb{R}^d$ act per unit area across surfaces that may form the exterior surface of the body itself or pass through its interior. Cauchy's Theorem shows the existence of a second order tensor field and the dependence of the traction force upon it:

$$\mathbf{t}(\mathbf{n}, \mathbf{x}) = \boldsymbol{\sigma}(\mathbf{x})\mathbf{n}(\mathbf{x}), \tag{2.2.9}$$

where $\boldsymbol{\sigma}(\mathbf{x})$, as defined in the Eulerian description, is known as the Cauchy Stress tensor. The resultant force $\mathbf{r} \in \mathbb{R}^d$ acting on a body $\Omega \subset \mathbb{R}^d$ is thus a combination of surface and volume integrals:

$$\mathbf{r}(\mathbf{x}, t) = \int_{\partial\Omega} \boldsymbol{\sigma}(\mathbf{x}, t)\mathbf{n}(\mathbf{x}, t) \, dA + \int_\Omega \rho(\mathbf{x}, t)\mathbf{b}(\mathbf{x}, t) \, dV, \tag{2.2.10}$$

where $\rho(\mathbf{x}, t)$ is the density at a given spatial point. It can be shown that the symmetry of the Cauchy stress tensor $\boldsymbol{\sigma}$ ensures conservation of angular momentum.

### 2.2.4  Newton's Second Law

Newton's second law [Newton, 1687] expresses a change in linear momentum (2.2.8) in terms of resultant force (2.3.7):

$$\frac{d}{dt} \int_\Omega \rho(\mathbf{x}, t)\mathbf{v}(\mathbf{x}, t) \, dV = \int_{\partial\Omega} \boldsymbol{\sigma}(\mathbf{x}, t)\mathbf{n}(\mathbf{x}, t) \, dA + \int_\Omega \rho(\mathbf{x}, t)\mathbf{b}(\mathbf{x}, t) \, dV,$$
$$\tag{2.2.11}$$

giving the basis for the description of both solid and fluid dynamics.

## 2.3  Solid Dynamics

In a solid body forces arise from deformation. The material or constitutive model relates a measure of deformation or *strain* to a potential function for

*strain energy density*. The derivative of strain energy with respect to deformation gives a tensor field for *stress*. Conservation of linear momentum and the *stress-strain* relation give the Elastodynamics equation, from which the time evolution of a deforming solid can be determined.

### 2.3.1  Constitutive Models

The expression for the change in length under deformation (2.2.4) leads to the Right Cauchy-Green strain tensor **C**. It contains information about stretching but removes rotation. The Green-Lagrange strain tensor **E** measures how much **C** varies from the identity tensor $\mathbb{I}$.

$$\mathbf{C} = \mathbf{F}^\top \mathbf{F} \tag{2.3.1}$$

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}^\top \mathbf{F} - \mathbb{I}) \tag{2.3.2}$$

A hyperelastic material is one in which a strain energy density function $W$ can be defined in terms of the three measures of strain $\mathbf{F}, \mathbf{C}$ and $\mathbf{E}$. This family of models include those of Saint Venant-Kirchhoff [Holzapfel, 2000], Mooney-Rivlin [Mooney, 1940], [Rivlin and Rideal, 1948], Ogden [Ogden and Hill, 1972].

We take as an example the St. Venant Kirchoff material model, which defines strain energy as quadratic in relation to the the Green-Lagrange strain:

$$W(\widehat{\mathbf{x}}, t) = \frac{\lambda}{2} \operatorname{tr}(\mathbf{E})^2 + \mu \operatorname{tr}(\mathbf{E}^2). \tag{2.3.3}$$

Lamé constants $\lambda$ and $\mu$, quantify the material's resistance to compression and shear respectively and are determined experimentally.

Taking derivatives with respect to strain gives expressions for the First and Second Piola-Kirchhoff stress tensors, and applying the inverse Piola Transform gives the Cauchy stress.

| | | | | |
|---|---|---|---|---|
| Second Piola-Kirchhoff stress | $\mathbf{S}$ | $=$ | $\frac{\partial W}{\partial \mathbf{E}}$ | $= \lambda \operatorname{tr}(\mathbf{E})\mathbb{I} + 2\mu \mathbf{E}$ |
| First Piola-Kirchhoff stress | $\mathbf{P}$ | $=$ | $\frac{\partial W}{\partial \mathbf{F}}$ | $= \mathbf{FS}$ |
| Cauchy stress | $\boldsymbol{\sigma}$ | | | $= J^{-1}\mathbf{PF}^\top$ |

$$\tag{2.3.4}$$

The material models are extended to allow for muscle contraction coupled to the monodomain equation (5.3.2) by the addition of contractile stress to the material model corresponding to fibre directions $\widehat{\mathbf{f}}$ in the reference configuration. Other models will be introduced as needed, such as [Guccione et al., 1995] and [Tang et al., 2007] which allow for passive isotropy due to the fibre orientation of the myocardium.

### 2.3.2  Elastodynamics Equation

Until now these results have been given in their Eulerian form, but we now combine them in their Lagrangian form.

Taking the reference configuration at $t = 0$ we have $\mathbf{x} = \varphi(\widehat{\mathbf{x}}, 0) = \widehat{\mathbf{x}}$, thus (2.2.7) and (2.2.5) imply that mass is conserved over time:

$$\int_\Omega \rho(\mathbf{x}, t)\, dV \;\;=\;\; \int_{\widehat{\Omega}} J(\widehat{\mathbf{x}}, t)\rho(\varphi(\widehat{\mathbf{x}}, t), t)\, d\widehat{V} \qquad (2.3.5)$$

$$=\;\; \int_{\widehat{\Omega}} \rho(\widehat{\mathbf{x}}, 0)\, d\widehat{V}. \qquad (2.3.6)$$

We convert the Eulerian description of resultant force (2.2.10) into its Lagrangian form using integral relations (2.2.6) and (2.2.5) and substitute mass conservaiton (2.3.6):

$$\mathbf{r}(\mathbf{x}, t) = \int_{\partial\widehat{\Omega}} \underbrace{J(\widehat{\mathbf{x}}, t)\boldsymbol{\sigma}(\varphi(\widehat{\mathbf{x}}, t), t)\mathbf{F}(\widehat{\mathbf{x}}, t)^{-\top}}_{\mathbf{P}(\widehat{\mathbf{x}}, t)} \widehat{\mathbf{n}}(\widehat{\mathbf{x}})\, d\widehat{A} + \int_{\widehat{\Omega}} \rho(\widehat{\mathbf{x}}, 0)\mathbf{b}(\varphi(\widehat{\mathbf{x}}, t), t) d\widehat{V}$$

$$(2.3.7)$$

The tensor field $\mathbf{P}(\widehat{\mathbf{x}}, t)$ is the First Piola-Kirchhoff stress tensor which encodes the *stress-strain* relation.

Substituting into Newton's Second Law (2.2.11) relates the change in linear momentum (2.2.8) to body and traction forces.

$$\frac{d}{dt} \int_\Omega \rho(\mathbf{x}, t)\mathbf{v}(\mathbf{x}, t)\, dV = \int_{\partial\widehat{\Omega}} \mathbf{P}(\widehat{\mathbf{x}}, t)\widehat{\mathbf{n}}(\widehat{\mathbf{x}})\, d\widehat{A} + \int_{\widehat{\Omega}} \rho(\widehat{\mathbf{x}}, 0)\mathbf{b}(\varphi(\widehat{\mathbf{x}}, t), t) d\widehat{V}$$

$$(2.3.8)$$

This is the Elastodynamics equation, which we will integrate in time for the solution of the solid body dynamics problem.

## 2.4  Fluid Dynamics

We model blood as an incompressible Newtonian fluid and apply conservation of mass and linear momentum as before. The fluid flow is naturally expressed in its Eulerian form and calculation of the time derivative leads to a *transport* term. Fluids do not support shear forces, and thus Cauchy stress depends upon *strain rate* and *pressure*. The combination of transport, stress and body forces result in the Navier-Stokes equations for incompressible flow.

This section is based upon the previous texts in continuum mechanics as well as [Versteeg and Malalasekra, 1996] and [Patankar, 1980].

### 2.4.1   Continuity Equation

The Eulerian description of a fluid flow is spatial velocity field $\mathbf{v}(\mathbf{x}, t)$. Change in density is equivalent to the accumulation of mass by velocity. Conservation of mass can thus be expressed in differential form as follows:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0. \tag{2.4.1}$$

For an incompressible fluid, density $\rho$ is constant and the expression reduces to the continuity equation:

$$\nabla \cdot \mathbf{v} = 0. \tag{2.4.2}$$

### 2.4.2   Advection

Consider a scalar function $q : \mathbb{R}^d \times \mathbb{R} \mapsto \mathbb{R}$, of a spatial point $q(\mathbf{x}, t)$ in motion such that $\mathbf{x} = \varphi(\widehat{\mathbf{x}}, t)$. The total time derivative of $q$ has a contribution from changes in both space and time. Applying the Chain Rule recovers the inner product of velocity $\mathbf{v}$ and spatial gradient $\nabla q$:

$$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \frac{\partial q}{\partial \varphi} \cdot \frac{\partial \varphi}{\partial t} \tag{2.4.3}$$

$$= \frac{\partial q}{\partial t} + \mathbf{v} \cdot \nabla q. \tag{2.4.4}$$

The value of $q$ is thus transported by velocity. For the fluid $q$ is replaced by $\mathbf{v}$, and the velocity field transports itself by advection:

$$\frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v}. \tag{2.4.5}$$

It is worth noting that if the velocity $\mathbf{v}$ of the fluid is known, then its evolution can be described without knowledge of its motion $\varphi$.

### 2.4.3   Cauchy Stress

In an incompressible fluid, forces arise from changes in velocity and pressure. The Cauchy stress tensor is thus a function of spherical pressure field $p(\mathbf{x}, t)$ and the *strain rate* tensor for viscous stress:

$$\boldsymbol{\sigma}(p, \mathbf{v}) = -p\mathbb{I} + \mu \left[ \nabla \mathbf{v} + \nabla \mathbf{v}^\top \right] \tag{2.4.6}$$

where $\mu$ is the dynamic viscosity, considered constant in the Newtonian case and $\mathbb{I}$ is an identity matrix.

### 2.4.4   Navier-Stokes Equations

The expression for Newton's Second Law (2.2.11) is written in its integral form over an arbitrary set $\Omega$. Since the balance of momentum holds for any $\Omega$ and density $\rho$ is constant, the law can be expressed in its differential form as follows:

$$\rho \frac{D}{Dt}\mathbf{v}(\mathbf{x}, t) = \nabla \cdot \boldsymbol{\sigma}(\mathbf{x}, t) + \rho \mathbf{b}(\mathbf{x}, t). \tag{2.4.7}$$

We substitute for advection (2.4.5) on the left and fluid Cauchy stress (2.4.6) on the right

$$\rho \left[ \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} \right] = \nabla \cdot \left( -p\mathbb{I} + \mu \left[ \nabla \mathbf{v} + \nabla \mathbf{v}^\top \right] \right) + \rho \mathbf{b}, \tag{2.4.8}$$

then make use of the following calculus identities,

$$\nabla \cdot p\mathbb{I} = (\nabla p)\mathbb{I} + p(\nabla \cdot \mathbb{I}) = \nabla p, \tag{2.4.9}$$

$$\nabla \cdot (\nabla \mathbf{v}) = \Delta \mathbf{v}, \tag{2.4.10}$$

$$\nabla \cdot (\nabla \mathbf{v}^\top) = \nabla(\nabla \cdot \mathbf{v}) = 0, \tag{2.4.11}$$

where, in the last case, $\nabla \cdot \mathbf{v} = 0$ by continuity (2.4.2) . The resulting momentum equation, along with continuity are the Navier-Stokes equations for incompressible flow:

$$\nabla \cdot \mathbf{v} = 0 \tag{2.4.12}$$

$$\rho \left[ \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} \right] = -\nabla p + \mu \Delta \mathbf{v} + \rho \mathbf{b}. \tag{2.4.13}$$

The integration of these equations in time will give the solution of the fluid dynamics problem.

## 2.5   Fluid-Structure Interaction

Given the continuum description of solid and fluid dynamics it now remains to define the dynamics at their interface. The combined description of the Lagrangian properties of the solid and the Eulerian properties of the fluid lead to the Arbitrary Lagrangian-Eulerian framework (ALE). This material follows [Formaggia, 2009] and [**?**].

### 2.5.1   ALE Formulation

Figure 2.3 shows solid $\widehat{\Omega}_s$ and fluid $\widehat{\Omega}_f$ domains in the reference configuration with their images $\Omega_s(t)$ and $\Omega_f(t)$ under deformation. Their boundaries
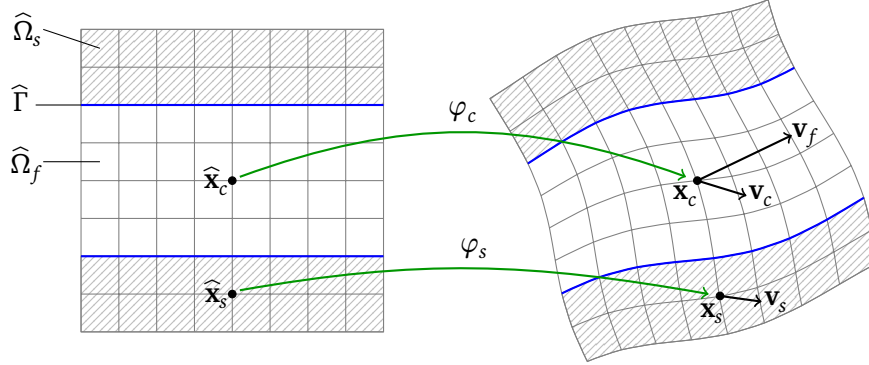
Figure 2.3: Solid, Fluid and Computational domains, velocities and motions in the ALE formulation.

$\partial\widehat{\Omega}_s$ and $\partial\widehat{\Omega}_f$ allow the definition of boundary conditions in the normal way. The fluid-structure interface or surface $\Gamma(t)$ is defined by their common boundary:

$$\Gamma(t) = \Omega_s(t) \cap \Omega_f(t) \qquad (2.5.1)$$

The current position $\mathbf{x}_s$ and velocity $\mathbf{v}_s$ of a point on the interior of the solid domain $\widehat{\mathbf{x}}_s$ can be described via the elastodynamics equation (2.3.8) via the solid motion $\varphi_s(\widehat{\mathbf{x}}_s, t)$.

As noted in Section 2.4.2, for a point on the interior of the fluid domain, flow velocity $\mathbf{v}_f$ is an Eulerian field described by the Navier-Stokes equations (2.4.13) without the need for a reference position or motion.

### 2.5.2 Computational Domain

In order to couple the conservation of mass and momentum on the solid and fluid domains the ALE formulation requires an extra definition. The computational domain $\Omega_c \subset \mathbb{R}^d$ represents both the solid and the void through which fluid flows.

$$\Omega_c = \Omega_s \cup \Omega_f \qquad (2.5.2)$$

It thus occupies the same space as $\Omega$ but carries Lagrangian information about the current position $\mathbf{x}_c$ and velocity $\mathbf{v}_c$ of any point in the reference configuration $\widehat{\mathbf{x}}$ via the computational motion $\varphi_c(\widehat{\mathbf{x}}_c, t)$.

In the solid domain the solid and computational motions are equal but on the fluid domain the computational motion $\varphi_c$ is independent of the fluid.

$$\varphi_c = \varphi_s, \ \ \forall \mathbf{x} \in \Omega_s \qquad (2.5.3)$$

The purpose of $\Omega_c$ is to encode the coupling conditions between the solid and fluid domains on the surface $\Gamma$. The computational motion does not have any physical meaning in itself and does not influence the solution. It is updated by any means that ensures the admissibility of the deformation.

Since it has velocity on the fluid domain, this velocity $\mathbf{v}_c$ must be subtracted from fluid velocity $\mathbf{v}_f$ so that only relative velocity is used in the calculation of advection (2.4.5), hence:

$$\rho\left[\frac{\partial \mathbf{v}_f}{\partial t} + \left[(\mathbf{v}_f - \mathbf{v}_c) \cdot \nabla\right]\mathbf{v}_f\right] = -\nabla p + \mu \Delta \mathbf{v}_f + \rho \mathbf{b} \qquad (2.5.4)$$

It is now possible to apply the coupling conditions to the solid, fluid and computational domains.

### 2.5.3   Coupling Conditions

Firstly, in the reference configuration the images of solid and computational deformation coincide for all points in the domain $\widehat{\Omega} = \widehat{\Omega}_s \cup \widehat{\Omega}_f$.

$$\varphi_s(\widehat{\mathbf{x}}, 0) = \varphi_c(\widehat{\mathbf{x}}, 0), \ \forall \widehat{\mathbf{x}} \in \widehat{\Omega} \qquad (2.5.5)$$

Secondly, the solid and computational motion of each point on the surface $\widehat{\Gamma}$ coincides at all times. This gives continuity of position.

$$\varphi_s(\widehat{\mathbf{x}}, t) = \varphi_c(\widehat{\mathbf{x}}, t), \ \forall \widehat{\mathbf{x}} \in \widehat{\Gamma} \qquad (2.5.6)$$

Thirdly, the velocity of each point on the surface $\Gamma$ coincides at all times. This applies a *no-slip* boundary condition to the fluid-structure interface.

$$\mathbf{v}_s(\mathbf{x}, t) = \mathbf{v}_f(\mathbf{x}, t) = \mathbf{v}_c(\mathbf{x}, t), \ \forall \mathbf{x} \in \Gamma \qquad (2.5.7)$$

Finally, traction forces are set equal and opposite on the surface $\Gamma$. This gives continuity of stress:

$$\boldsymbol{\sigma}_s \mathbf{n}_s + \boldsymbol{\sigma}_f \mathbf{n}_f = \mathbf{0}, \ \forall \mathbf{x} \in \Gamma \qquad (2.5.8)$$

In this way a point on the surface is subject to forces arising from the motion of both solid and fluid.

### 2.5.4   Fluid-Structure Summary

With the inclusion of Dirichlet and Neumann boundary conditions $\mathbf{f}, \mathbf{g}$ on the solid and fluid domains, ignoring body forces, the coupled FSI problem can be written as follows [Formaggia, 2009]:

Solid:

$$\rho_{s,0}\frac{d\mathbf{v}_s}{dt} - \nabla \cdot \boldsymbol{\sigma}_s = \mathbf{0}, \quad \mathbf{x} \in \Omega_s \tag{2.5.9}$$

$$\mathbf{v}_s = \mathbf{f}_s, \quad \mathbf{x} \in \partial\Omega_{s,D} \tag{2.5.10}$$

$$\boldsymbol{\sigma}_s\mathbf{n}_s = \mathbf{g}_s, \quad \mathbf{x} \in \partial\Omega_{s,N} \tag{2.5.11}$$

Fluid:

$$\nabla \cdot \mathbf{v}_f = \mathbf{0}, \quad \mathbf{x} \in \Omega_f \tag{2.5.12}$$

$$\rho_f\left[\frac{\partial\mathbf{v}_f}{\partial t} + [(\mathbf{v}_f - \mathbf{v}_c)\cdot\nabla]\mathbf{v}_f\right] + \nabla p - \mu\Delta\mathbf{v}_f = \mathbf{0}, \quad \mathbf{x} \in \Omega_f \tag{2.5.13}$$

$$\mathbf{v}_f = \mathbf{f}_f, \quad \mathbf{x} \in \partial\Omega_{f,D} \tag{2.5.14}$$

$$\boldsymbol{\sigma}_f\mathbf{n}_f = \mathbf{g}_f, \quad \mathbf{x} \in \partial\Omega_{f,N} \tag{2.5.15}$$

FSI:

$$\mathbf{x}_s = \mathbf{x}_f = \mathbf{x}_c, \quad \mathbf{x} \in \Gamma \tag{2.5.16}$$

$$\mathbf{v}_s = \mathbf{v}_f = \mathbf{v}_c, \quad \mathbf{x} \in \Gamma \tag{2.5.17}$$

$$\boldsymbol{\sigma}_f\mathbf{n} + \boldsymbol{\sigma}_s\mathbf{n} = \mathbf{0}, \quad \mathbf{x} \in \Gamma \tag{2.5.18}$$

We have thus defined a set of coupled problems in electrophysiology, solid and fluid dynamics. In order to solve them we next consider the computational tools at our disposal.

**Part II**

# Method

# Chapter 3

# GPU Computation

## 3.1 Introduction

This chapter will briefly introduce GPU architecture and show how parallel processing can increase computing performance. The cost of this improved performance is a set of constraints upon code design. If these constraints are not followed carefully then the benefits of parallel computing may be lost.

It is common for practitioners to *parallelize* or *accelerate* existing code by *porting* it to GPU, and then to show *speedup* with respect to their CPU implementation. There also exist frameworks and libraries which can be applied to standard problems, but without considering parallel processing in the design of a mathematical algorithm there is a limit to what these approaches can achieve.

The aim of this thesis is to re-consider existing mathematical and computational approaches and to show *proof-of-concept* for a multi-physics solver with parallel computing at the centre of its design. As such it is important to make clear the objectives and compromises that must be made, since they will guide the subsequent work.

This chapter is based largely upon the OpenCL language specification [Khronos OpenCL Working Group, 2022], programming [NVIDIA Corporation, 2012], and performance guides [NVIDIA Corporation, 2011]. There are many guide books including [Scarpino, 2012].

## 3.2 GPU Architecture

Originally, the Graphics Processing Unit (GPU) was a specialised piece of computer hardware designed to manipulate and alter memory for output to the

screen buffer for display. The specialism arose because frequently required tasks, such as interpolation, were often parallel in their nature, usually with respect to pixels, and required low latency, specifically the screen buffer should be refreshed at the frame rate of the display.

Once present, the use of the units was extended to include other computationally intensive and parallel tasks, such as linear operations on polygonal meshes and texture mapping in graphics rendering, as well as video and sound processing. Success in these tasks, drew interest for scientific applications and languages were developed that permitted General Purpose (GPGPU) computing.

One such language is OpenCL [Khronos OpenCL Working Group, 2022], which is a based upon set of agreed standards for functionality agreed by the GPU vendors and consumers, comprising the Khronos Working Group, and implemented separately by vendors. The group is also responsible for other standards including OpenGL shader language, widely used in computer graphics. The adoption of these standards was beneficial to the growth of GPU computing since it allowed elasticity of supply and demand. Since then however GPGPU has become a large and very significant market which now includes multimedia processing transforms, the training and forward evaluation of neural networks and cryptocurrency mining. Vendors have extended functionality into proprietary languages such as NVIDIA's CUDA, Apple's Metal Shader Language and Microsoft's DirectCompute Extensions, claiming improved programming flexibility and performance, largely for business purposes. Increasing competition has reduced the vendors' enthusiasm for open standards such as OpenCL, but it is still preinstalled and supported on almost all new and existing machines.

At the same time that GPU languages have evolved, so have the units themselves. Starting as specialised circuits they became larger off-chip devices with their own DRAM memory. Now that their benefits have been demonstrated both in terms of speed and power consumption, there has been some focus on improving the integration of CPU and GPU memory, which is a significant bottleneck, and is discussed later in this chapter.

OpenCL 3.0 includes instructions that allow simultaneous co-processing of memory by both CPU and GPU.

We start by introducing the main aspects of GPU computation then consider how this determines code design.

### 3.2.1  Hardware Multi-threading

Both CPU and GPU execution pipelines make use of threads. A four quad-core CPU can process 16 threads in parallel (or 32 with hyperthreading). There may
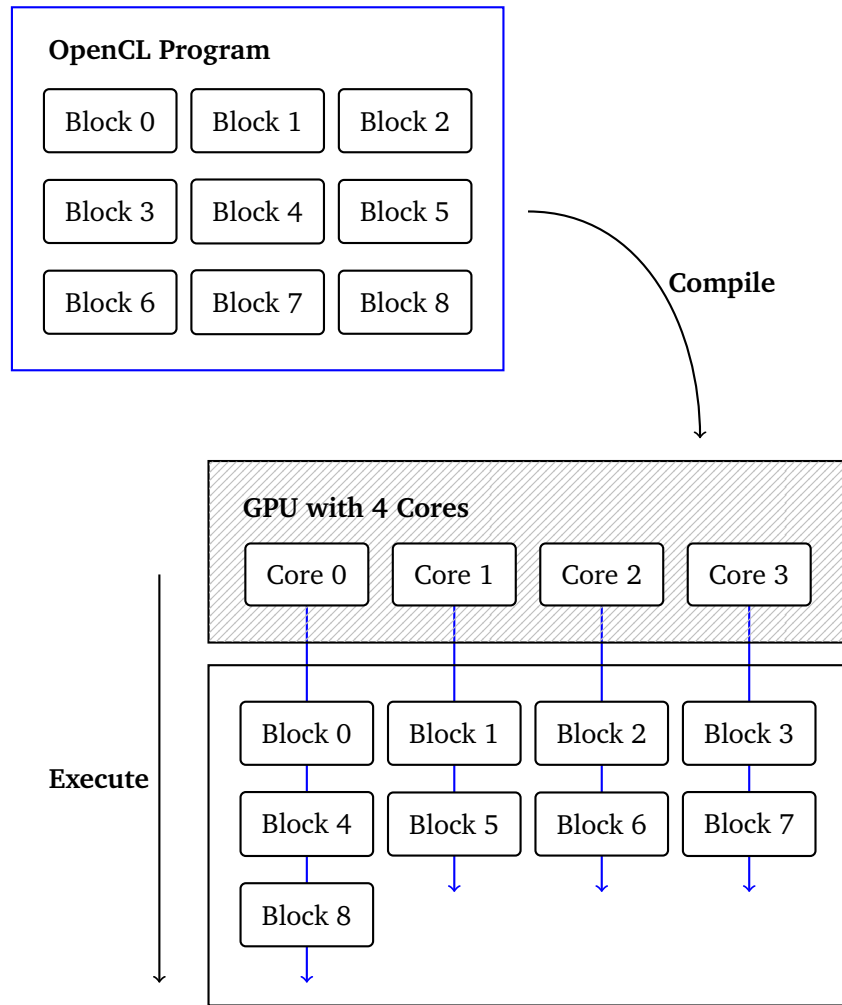
Figure 3.1: Thread blocks in a program are processed independently and in parallel by a multicore GPU.

be a delay while the results of a floating point operations (FLOPs) or memory access becomes available in register memory. For FLOPs this may be as much as 24 cycles while for off-chip memory access 400-800 cycles. If this latency causes the pipeline to stall, then the environment can switch to another to keep the processor busy. The pipeline is thus filled via *multi-threading*. The cost of switching threads on a CPU may be high, since each is fully independent and carries with it a large amount context and register information stored both on and off-chip.

Figure 3.1 gives a schematic representation of an OpenCL program. The program consists of a large number of threads, grouped into thread blocks (or work groups). Each block contains as many as 1024 threads and is allocated

to a compute unit (or core) at compile time. At execution the thread blocks are processed sequentially but independently by the compute units. It is worth noting that the order in which the blocks are processed may not have a close physical relation to their layout in the program, as shown in the figure.

The execution context, counters and register information are preserved on-chip for all threads for the entire life of the block. Within the block there are *thread groups* with a minimum size of 32 threads. If any thread group is stalled the processor can switch to another and continue execution at zero cost. As a result the register latency for floating point operations and memory access can, in theory, be hidden.

This architecture is known as Single Instruction Multiple Thread (SIMT). In addition the GPU may have vector processing units, which can perform concurrent floating point operations (usually four) in the same number of cycles as a single FLOP, thus implementing a Single Instruction Multiple Data (SIMD) architecture. GPU languages have vector and matrix data types for this purpose.

A good GPU may have 100 compute units and can thus support the concurrent processing of 100,000 threads, with billions scheduled at any one time. Thus the GPU gives an opportunity for a huge increase in computing performance. For this improvement to be achieved, the action of the program must be *parallel*, that is the design of the program should allow the threads to operate on a large number of data elements at the same time.

### 3.2.2   Memory

#### 3.2.2.1   Host/Device Memory Transfer

In general, CPU memory is not directly accessible to the GPU. The original model for computation involved populating a memory buffer on the CPU *host*, copying it via the Peripheral Component Interconnect (PCI) bus to GPU *device* for processing and rendering, then filling the screen buffer for the display. The one-way nature of this process allowed for read-only and write-only buffers which removes the need for cache management.

With the advent of GPGPU, users generally need to copy results back to the CPU to store results as files, requiring another pass through the PCI bus. If procedures require intermediate processing or memory access then there must be further transfers and cache management.

The bandwidth of the PCI bus between CPU and GPU memory is around 8GBps, compared to 128GBps between GPU memory and processor, or 100GBps between CPU memory and processor. The cost of memory transfer is therefore a key bottleneck for GPU computation.

### 3.2.2.2   Device Memory Spaces

Table 3.1 shows features of GPU device memory spaces. They are ordered roughly by preference, with Register and Shared memory being ideal but limited in size and scope. Constant memory is cached and therefore has zero latency on second read and can be considered as good as a register. Thereafter Texture memory, which is discussed later, benefits from caching and optimised memory memory access.

| Memory | On-chip | Cached | Access | Scope | Lifetime |
|--------|---------|--------|--------|-------|----------|
| Register | yes | - | read/write | thread | thread |
| Shared | yes | - | read/write | block | block |
| Constant | no | yes | read | all | all |
| Texture | no | yes | read/write* | all | all |
| Local | no | yes/no* | read/write | thread | thread |
| Global | no | no | read/write | all | all |

Table 3.1: Principle features of GPU memory spaces, reproduced from [NVIDIA Corporation, 2011]. *Texture memory write access and Local memory caching are version and vendor-dependent.

The vast majority of memory is Global, where large working buffers reside and latency is roughly 100 times that of registers. During execution the programmer can actively manage the movement of data from Global to Shared memory and as such it is a user-controlled cache. Some devices will cache data into Shared memory directly. When Global memory is accessed, however, it is not possible to avoid the cost of one read or write per kernel execution.

When memory requirements exceed the number of available registers, the processor will first make use of on-chip Shared then off-chip Global memory, with associated performance reduction called *register spill*. If a kernel requires more resources than are available then it will fail to launch.

### 3.2.2.3   Texture Memory

Texture mapping is the process in computer graphics rendering by which 2D bitmap images are applied to the surfaces of 3D polygonal meshes. It is a fundamental part of the graphics pipeline which makes multiple reads of many such bitmaps and various techniques such as interpolation, reflection and repetition, all of which are ideal for GPU processing. As such there is a special portion of cached memory available for these operations which has several hardware optimisations for fast access.

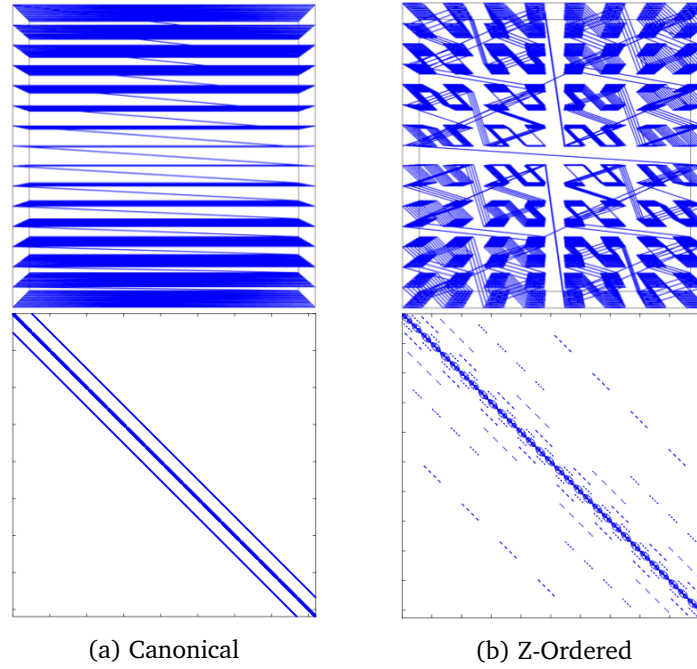|         (a) Canonical          |         (b) Z-Ordered          |

Figure 3.2: Canonical and Z-Ordered memory indexing in 3D, and corresponding sparsity patterns for 7-point stencil.

Space-filling curves map high dimensional spaces into one-dimension. Texture memory uses this technique to re-index memory addresses. When assembling a stencil operation, [Morton, 1966] showed that the technique is equivalent to reordering a sparse matrix such that bandwidth is optimally reduced. An example for a 3D cube with a 7-point stencil operation is shown in Figure 3.2.

The canonical scheme in 2D generates a scalar memory index from zero-based row and column indices by taking the row index multiplied by the row size plus the column index. Z-ordering takes the row and column indices as binary numbers and interleaves their digits in alternating fashion. The method extends naturally into higher dimensions resulting in the distinctive z-shaped curves that give the method its name.

In addition to this, texture memory has some other attractive features. Firstly, Row and column indices can be normalised into the range $[0, 1]^d$ and called using floating point coordinate values. There is specialised hardware that calculates memory addresses for the surrounding values and performs bilinear or trilinear interpolation. This is equivalent to storing a scalar, vector or even tensor field in a bi- or tri-linear Lagrangian basis. The GPU can be set to handle out-of-bounds either returning zero, fixed, repeated or reflected values as necessary.

Secondly, Bitmap data in RGBA format is stored as a vector of four floating point values which is ideal for storing vector fields. The values can be stored as half, single, or double precision values or unsigned integers. When queried the values can be returned as higher precision floating point or integer types.

These two features allow data fields stored at different spatial resolutions and precision to be combined seamlessly into the same calculation. This is important for multigrid methods, since a problem can be completely reconstructed at any scale.

## 3.3   Design Objectives

As discussed in the previous sections, the performance benefits of the SIMT and SIMD architectures, as well as optimised memory access of the GPU offer a potentially huge increase in performance over CPU processing. In order to benefit from these advantages, the computer program must have the following characteristics:

- The action of the program should be *parallel*, that is that action of the threads can be performed on a large number of data elements at the same time.

- Traffic over the PCI bus between the CPU host and the GPU device should be minimised.

- There should be *coherence* in data access, that is there should be spatial and temporal locality in both read and write memory operations which allows the device to *coalesce* memory operations.

The following sections describe some strategies by which these objectives can be achieved.

### 3.3.1   Parallel Execution

Maximising parallel execution starts with structuring the algorithm in a way that exposes as much data parallelism as possible.

The floating point dot product $\mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is a fundamental operation in linear algebra. It comprises a product which can be computed in parallel and a sum which is inherently serial. For the product both CPU and GPU can use independent threads. As discussed in previous sections, there would be a cost in PCI memory transfer from host to device but for large vectors this would be outweighed by the increased thread count of the GPU (10k threads) compared to the CPU (32 threads).

For the sum, the CPU compiler would use *loop unrolling* to accelerate the operation, allocating memory for subtotals which could be computed in parallel. The GPU would use a similar *parallel reduction* operation in which vectors of subtotals would hold results of parallel sums. In both cases, speedup would thus be a tradeoff between parallel computation and memory use, as well as the cost of memory transfer for the GPU. For operations more complex than the dot product, algorithms and strategies grow in complexity, and a large quantity of literature and software exists that attempts to solve these problems.

The alternative is to remove the dot product from the algorithm entirely, and this is the approach taken here. At all stages the operations used by the solvers are data parallel. The threads behave like stencil operations reading from their near neighbours, and writing to a single location in memory. There is no CPU calculation in the algorithm at all, and no data transfer between device or host during calculation except for the writing of output files.

### 3.3.2   Control Flow

Branching keywords (*if, for, do, while, switch*) cause threads to follow different execution paths. A CPU handles code branching in a straightforward way, executing only paths that are required. The SIMT architecture of the GPU relies on all multiprocessors carrying out the same instructions at any given time. As a result code divergence is serialised by the compiler. Where conditional logic is used within a kernel, both branches are calculated and one value is discarded. It is therefore important to design the algorithm such that this code branching contains a minimal amonut of processing.

For example, an if control condition will be allocated a per thread control predicate, which stores the result of the logical evaluation. The code block within the if statement will execute for all threads, but any resulting operations such as register writes will only take place where the control predicate is true. This is known as *bit-masking*.

When there is an if...else condition, the compiler will execute both code branches for all threads, applying bit-masking to the results, effectively serialising the divergence. This serialisation removes the advantage of parallel processing and greatly reduces performance. Therefore, the algorithm and code design in this work removes all branching statements.

### 3.3.3   Memory Throughput

For both CPU and GPU memory access represents the key limit to computational performance.

The first and most obvious optimisation is the use of single precision floating point variables. GPUs were not developed for high numerical precision and while some offer extensions for double precision calculations, most have single precision floating point registers and arithmetic units. As such single precision calculations halve instruction and memory throughput.

The range of values and accuracy of single precision floats is sufficient for the calculation physiological information that our model requires. In some cases (such as fibre fields) half precision data is sufficient.

### 3.3.3.1 PCI Bus

As mentioned earlier, the relatively low bandwidth of the PCI bus (8MBps) compared to GPU the Global memory (128MBps) space is a key performance bottleneck. There are several strategies which minimise transfer over the PCI bus:

Memory should be allocated and persisted on the GPU wherever possible. Routines that allocate device memory allow flags that control read and write access by both host and device allowing the compiler to optimise cache use. Memory transfers should *coalesced* into a few bulk operations rather than many smaller ones.

Data transfer can be avoided by re-calculation of values rather than storage. This is especially true with reference to CPU and GPU where a kernel that is slower on GPU than CPU may be preferable as it avoids transfer. It is also relevant to device-only storage where calculation may be faster than retreival.

Memory transfer can be achieved in a straightforward way by the allocation of resources on both host and device followed by a memory copy from one to the other. Another method that is available is the use of *pinned memory*. Implemented in different ways by different vendors it is effectively a region of cache that is in the address space of both device and host and not subject to paging. As such there are routines that will return a pointer to GPU memory that can be read or written by the CPU at a higher bandwidth than a traditional copy operation.

In addition, pinned memory can be read in a synchronous or asynchronous fashion. Once a pointer is allocated, the application can either perform a *blocking transfer* in which the GPU processing is paused until IO is complete, or a *non-blocking transfer* in which the GPU continues its execution. In the second case the latency of memory transfer is hidden but its contents may change during the operation. This must be considered as part of the program design.

As discussed above, the continuing integration of CPU and GPU memory has led to *Shared Virtual Memory* in which memory is accessible to both host and device

and can be operated upon in a synchronous and asynchronous way, with out the need for explicit transfers. For example, Apple's recent M1 processor has 64GB of shared memory for exactly this purpose and demonstrates that manufacturers are accepting data parallelism as a fundamental part of processor design.

As a result the design considerations outlined above, the code implementation in this work does not create any memory or perform any calculations on the on the CPU. All allocations and calculations are carried out directly on the GPU. The only memory transfer over PCI bus is the reading of buffers to generate output files. This is performed asynchronously via pinned memory and is thus optimal with respect to memory transfer.

### 3.3.3.2   Device Memory

There are several considerations which relate to the efficient use of the GPU device memory spaces:

The use of shared memory should be maximised. An example of blocked matrix multiplication is usually given, in which the block size allows multiple threads to share local memory access. In this case data is loaded into shared memory manually.

In general however, it can be assumed that the device will make one cached read and/or write per kernel execution. Thus it is important that the kernel does not require multiple access to global memory. Atomic writes are serialised, that is multiple writes to the same memory address will be carried out separately but the order of their execution is not defined, this results in a *race condition* for both read and write. It is important that code either avoids race conditions or is robust to them. For instance, when using Jacobi iteration to solve a linear system, a race condition would lead in some instances to a Gauss-Seidel iteration instead. In this case the race condition is actually beneficial to the speed of convergence and the solvers in this work will take advantage of this behaviour.

Originally it was a requirement that a memory location was either read-only or write-only per kernel access, thus removing the need for cache coherence. Modern languages and processors allow for read/write operations within a kernel but it is still beneficial to observe this design constraint. Memory can be flagged as read or write only per kernel for increased performance.

The features of texture memory indicate the way in which *coherent* memory access patterns allow for *coalesced* memory access. The SIMT architecture blocks threads on to multiprocessor cores which use and cache Shared memory. It is important that all threads in a block are accessing memory from the same or similar cache lines. For this reason random or strided memory access will

destroy the benefits of device caching. The next chapter describes the structured grid on which the problem is discretised and this design decision is made with coherent memory access in mind.

Register spill should be avoided by separating large complex kernel in to smaller units. If a kernel requires more memory than the available registers then it will be allocated from Global resources, which introduces hundreds of clock cycles of latency. There is a low cost in launching a kernel (estimated around $300\mu s$) and so decomposition may be a better option.

As already mentioned, the kernels in this work will make multiple local reads per kernel and only write to a single unique location per memory buffer. This coherent access pattern takes full advantage of spatial locality and cached global memory access.

### 3.3.4   Instruction Usage

The final design considerations take place at the register and processor level with the aim of maximising processor throughput:

As mentioned before, the floating point operations of a GPU are naturally single precision and the macroscopic nature of the work allows for their use.

As with any good code, slow functions such as division and numerical root finding should be avoided. Similarly most GPUs do not have dedicated integer processing units and integer division and modulo operations should be avoided.

The OpenCL language has compiler directives which coalesce floating multiply-add operations which can achieve significant speedup with a small cost in accuracy. There are also options for relaxed and fast mathematics calculations. These are implemented with geometric operations on meshes with only visual quality in mind and thus should be avoided

The following Chapters 4 and 5 describe respectively the discrete formulation and GPU parallel solution of the continuous problem outlined in Chapter 2.

# Chapter 4

# Discretization and Solver

In general, the various continuous spatial differential operators specified in Chapter 2 are discretized onto a computational mesh, and integrated in time via the method of lines. The use of the dual mesh combined with the Finite Volume Method for GPU assembly and processing is original to this work.

This Chapter has three sections, the first of which describes the discrete operators that will be required, first on the regular structured grid, and then in the deformed configuration. The next section explains how a signed distance function is used to encode geometry relating solid and fluid domains into the mesh. Finally, we show how a linear Laplacian solver is assembled with respect to both the geometry and deformation.

## 4.1  Operators

We consider the problem as described in Chapter 2 on a domain $\Omega \subset \mathbb{R}^d$ as a set of scalar $\mathbb{R}^d \to \mathbb{R}$, vector $\mathbb{R}^d \to \mathbb{R}^d$ and second-order tensor $\mathbb{R}^d \to \mathbb{R}^{d \times d}$ fields on a regular structured grid.

Some information is stored at mesh vertices $\mathbf{x}_i \subset \mathbb{R}^d$ while other values relate to the enclosed volumes $V_i$ and is stored at their centres. As such both the mesh and its dual are used in the solution of the problem and this is illustrated in Figure 4.1.

The fields will be stored for optimised memory access and processed in parallel in a per-vertex or per-volume manner. This is in line with the design considerations for GPU processing and memory access outlined in Chapter 3.

The mesh will deform according to the motions $\varphi_s, \varphi_c$ of the solid and computational subdomains $\Omega_s, \Omega_c$ as described in section 2.5.1. Throughout this Chapter
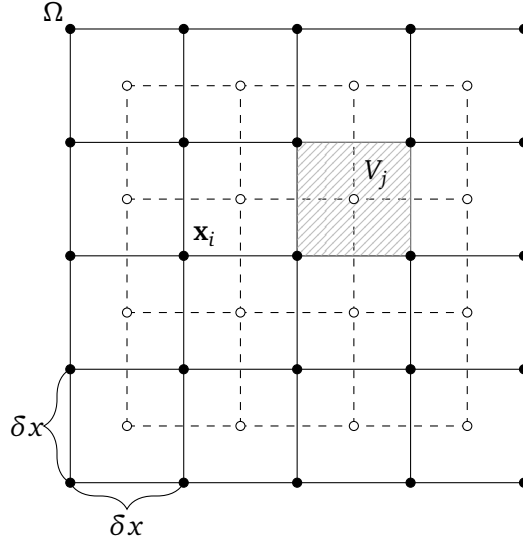
Figure 4.1: A regular structured grid over the domain $\Omega \subset \mathbb{R}^2$ with mesh width $\delta x$, vertices $\mathbf{x}_i$ and face dual at the centres of enclosed volumes $V_j$.

the calculations will be illustrated in $\mathbb{R}^2$ for simplicity, although everything described extends naturally into $\mathbb{R}^3$ and by the orthogonality of the coordinate system.

All of the operations in the following section are SIMT parallel in elements or vertices and SIMD parallel for vector and tensor fields. The first part of Chapter 6 will demonstrate their convergence properties.

### 4.1.1   Reference Volume

We consider a reference volume $\widehat{V} \subset \mathbb{R}^2$ as a unit square, with local coordinate system for a point $\widehat{\mathbf{x}} = [x_1, x_2]^\top \in [0, 1]^2$. The continuously differentiable vector field $\mathbf{u} : \mathbb{R}^2 \to \mathbb{R}^2$ is defined over $\widehat{V}$. Discrete differential operators will be derived for $\mathbf{u}$ but are also needed for scalar $u : \mathbb{R}^2 \to \mathbb{R}$ and tensor $\mathbf{U} : \mathbb{R}^2 \to \mathbb{R}^{2\times 2}$ fields. Where necessary definitions will be expanded accordingly. These definitions are equivalent to the reference element in FEM and are summarised in Figure 4.2.

#### 4.1.1.1   Interpolation

The value of $\mathbf{u}(\widehat{\mathbf{x}})$ can be approximated by bilinear interpolation. We use an abbreviated notation for the values at the vertices $\mathbf{u}(0, 1) = \mathbf{u}_{01}$ etc, as shown
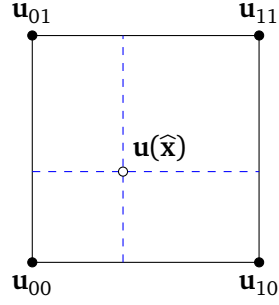
Figure 4.2: The reference volume $\widehat{V} \in \mathbb{R}^2$, showing function evaluation of $\mathbf{u}$ at at the vertices and a notional point $\widehat{\mathbf{x}}$.

in Figure 4.2:

$$\mathbf{u}(\widehat{\mathbf{x}}) \approx (1 - x_2)\left[(1 - x_1)\mathbf{u}_{00} + x_1\mathbf{u}_{10}\right] + x_2\left[(1 - x_1)\mathbf{u}_{01} + x_1\mathbf{u}_{11}\right] \qquad (4.1.1)$$

This is equivalent to applying a set of linear Lagrange basis functions to the vertices of $\widehat{V}$ and weighting $\mathbf{u}$ by $\widehat{\mathbf{x}}$. At the centre of the volume, where $\widehat{\mathbf{x}} = [0.5, 0.5]^\top$ the formula reduces as follows:

$$\mathbf{u}(\widehat{\mathbf{x}}) \approx \frac{1}{4}\left(\mathbf{u}_{00} + \mathbf{u}_{10} + \mathbf{u}_{01} + \mathbf{u}_{11}\right). \qquad (4.1.2)$$

The interpolation operation is the same for the scalar $u$ and tensor $\mathbf{U}$ fields.

### 4.1.1.2  Gradient

Taking the gradient of the Lagrange basis functions yields a difference formula for the discrete gradient of $\mathbf{u}$, equivalent to interpolating differences along the edges of the volume. Vertex values and difference weights are assembled into matrix form giving an expression for the Jacobian matrix:

$$\frac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} \approx \begin{bmatrix} \mathbf{u}_{00} & \mathbf{u}_{10} & \mathbf{u}_{01} & \mathbf{u}_{11} \end{bmatrix} \begin{bmatrix} -1 + \widehat{x}_2 & -1 + \widehat{x}_1 \\ +1 - \widehat{x}_2 & -\widehat{x}_1 \\ -\widehat{x}_2 & +1 - \widehat{x}_1 \\ +\widehat{x}_2 & +\widehat{x}_1 \end{bmatrix} \in \mathbb{R}^{2\times2} \qquad (4.1.3)$$

It is worth noting that the gradients are piecewise constant in their respective dimensions. At the centre of $\widehat{V}$ the formula reduces as follows:

$$\frac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} \approx \frac{1}{2} \begin{bmatrix} \mathbf{u}_{00} & \mathbf{u}_{10} & \mathbf{u}_{01} & \mathbf{u}_{11} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ +1 & -1 \\ -1 & +1 \\ +1 & +1 \end{bmatrix} \qquad (4.1.4)$$

For a scalar field $u$ the expression gives a row vector approximating the gradient transposed $\nabla u(\widehat{\mathbf{x}})^\top$. For the tensor field $\mathbf{U}$, the operator yields a second order tensor in $\mathbb{R}^{2\times 2\times 2}$:

$$\frac{\partial \mathbf{U}}{\partial \widehat{\mathbf{x}}} \approx \left[ \begin{array}{cc} \dfrac{\partial \mathbf{U}}{\partial \widehat{x}_1} & \dfrac{\partial \mathbf{U}}{\partial \widehat{x}_2} \end{array} \right]. \tag{4.1.5}$$

### 4.1.1.3  Divergence

The divergence of a vector or tensor field is defined as the limit of outward flux per unit volume.

$$\nabla \cdot \mathbf{u} = \lim_{\widehat{V} \to 0} \frac{1}{|\widehat{V}|} \oint_{\partial \widehat{V}} \mathbf{u} \cdot \widehat{\mathbf{n}} \, dA \tag{4.1.6}$$

Using the FVM approach, the discrete integral is the sum over $j$ faces of the face midpoint values $\mathbf{u}_j$, interpolated via (4.1.1), dotted with the exterior normals $\widehat{\mathbf{n}}_j$. The values at the face centres are the averages of the values at their incident vertices and thus the inner products represent average flux across each face. Since side lengths (face areas in $\mathbb{R}^3$) and volume $\widehat{V}$ have unit value in the reference configuration the divergence can be stated simply as:

$$\nabla \cdot \mathbf{u} \approx \sum_j^4 \mathbf{u}_j \cdot \widehat{\mathbf{n}}_j. \tag{4.1.7}$$

For a scalar field $u$, this operation is not defined, although in FVM the discrete application of Green's theorem to a scalar field gives the Green-Gauss gradient. On the reference volume this is equal to both the standard FDM and FEM gradient approximations.

For a tensor field $\mathbf{U}$ in the reference volume, the inner product is replaced by a matrix-vector product and the expression for discrete divergence is as follows:

$$\nabla \cdot \mathbf{u} \approx \sum_j^4 \mathbf{U}_j \cdot \widehat{\mathbf{n}}_j. \tag{4.1.8}$$

### 4.1.1.4  Laplacian

Due to the piecewise constant nature of the discrete gradient on the reference volume, its divergence is zero by definition. For the Laplacian it is necessary to consider both the primal and dual in two steps, meshes as shown in Figure 4.3. We make use of the fact that a regular structured grid is a *self-dual* mesh.

The field $\mathbf{u}$ is stored at each of the primal mesh vertices (black dots). In the first step, (4.1.4) is used to calculate a discrete gradient at the centre of each of the four volumes (white dots) adjacent to the central vertex $\widehat{\mathbf{x}}$.
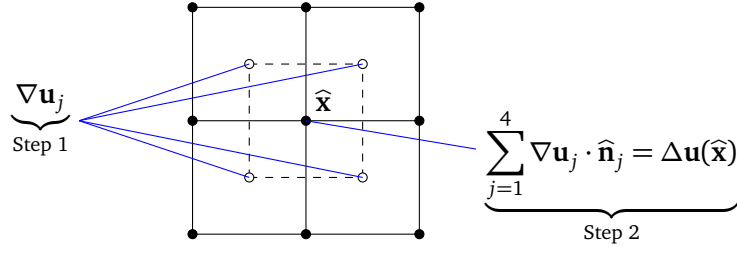
Figure 4.3: The Discrete Laplacian operator uses nine values of **u** at the vertices (black dots) to calculate four values of the gradient field $\nabla\mathbf{u}_j$ at the volume centres (white dots). It then calculates the divergence of the gradients at the centre of the dual, which is the Laplacian at vertex $\widehat{\mathbf{x}}$ itself.

The volume centres are the vertices of the dual mesh. In the second step, the divergence of the gradients is calculated via (4.1.8). Since the centre of the dual volume is a vertex of the primal mesh, this gives the Laplacian at the central vertex $\widehat{\mathbf{x}}$.

It is important to note that an identical procedure can be used to find the Laplacian for a field on the dual mesh, stored at the centre of the volumes, via gradients the vertices of the primal mesh. The algorithm will make use of both the primal and dual Laplacian operators.

Again, in the reference configuration these definitions are equivalent to both those of FEM and FDM.

### 4.1.2   Deformed Volume

In the deformed configuration, a Lagrangian description of position and velocity is provided by the solid $\varphi_s$ and computational $\varphi_c$ motions, as defined in section 2.5.1. We will derive expressions for discrete operators at the centre of a volume under an admissible deformation $\varphi : \mathbb{R}^2 \to \mathbb{R}^2$. Figure 4.4 shows the reference volume $\widehat{V}$ and its image $V$ under deformation.

#### 4.1.2.1   Interpolation

The interpolation operator at the centre of the deformed volume is the same as for the undeformed volume (4.1.1).
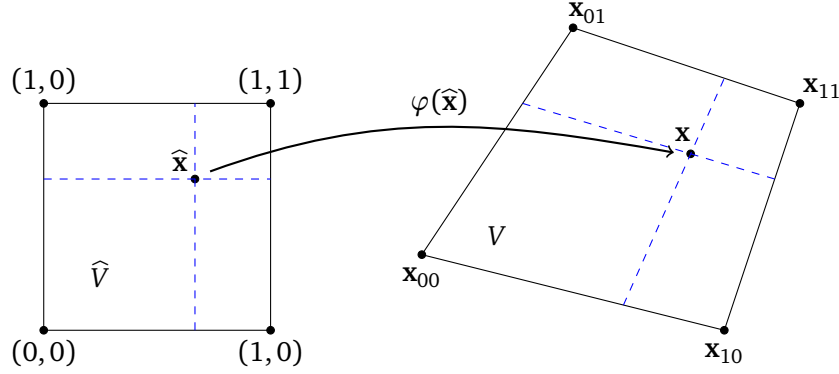
Figure 4.4: The reference and deformed volume showing vertices, an interior point $\widehat{\mathbf{x}}$ and its image $\mathbf{x}$.

#### 4.1.2.2 Deformation Gradient

We now consider the deformed image $\mathbf{x} = \varphi(\widehat{\mathbf{x}})$ of a point $\widehat{\mathbf{x}}$ in the reference volume $\widehat{V}$ and apply the discrete gradient operator (4.1.3) to the deformation itself. The resulting discrete approximation to the deformation gradient $\mathbf{F} \in \mathbb{R}^{2\times2}$ and its determinant $J$ are key to the continuum mechanics definitions of (2.2.1). At the centre of the volume (4.1.4) gives the following expression:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \widehat{\mathbf{x}}} = \frac{\partial x_i}{\partial \widehat{x}_j} \approx \frac{1}{2} \begin{bmatrix} \mathbf{x}_{00} & \mathbf{x}_{10} & \mathbf{x}_{01} & \mathbf{x}_{11} \end{bmatrix} \begin{bmatrix} -1 & -1 \\ +1 & -1 \\ -1 & +1 \\ +1 & +1 \end{bmatrix} \qquad (4.1.9)$$

It will frequently be necessary to calculate the inverse of the deformation gradient in various, such as $J\mathbf{F}^{-\top}$ in Nanson's relation (2.2.6).

In practice we work in $\mathbb{R}^3$ and calculate a transposed inverse and determinant using the vector cross product. For a matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$ with columns $\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \mathbf{a}_3]$ we have:

$$\mathbf{A}^{-\top} = \frac{1}{\det \mathbf{A}} \begin{bmatrix} \mathbf{a}_2 \times \mathbf{a}_3 & \mathbf{a}_3 \times \mathbf{a}_1 & \mathbf{a}_1 \times \mathbf{a}_2 \end{bmatrix} \qquad (4.1.10)$$

$$\det \mathbf{A} = \mathbf{a}_1 \cdot (\mathbf{a}_2 \times \mathbf{a}_3) \qquad (4.1.11)$$

The cost is three cross products, for which OpenCL has a built-in function.

#### 4.1.2.3 Gradient

For the gradient operator in the deformed configuration, we consider the function $\mathbf{u}$ under the coordinate transformation $\varphi$:

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}(\varphi(\widehat{\mathbf{x}})). \qquad (4.1.12)$$

Applying the chain rule and recognising the deformation gradient $\mathbf{F}$:

$$\frac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \widehat{\mathbf{x}}} \approx \frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{F}. \tag{4.1.13}$$

The gradient in the deformed configuration can thus be calculated via the reference gradient (4.1.4) and the deformation gradient (4.1.9), inverted as per (4.1.10).

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}}\frac{\partial \widehat{\mathbf{x}}}{\partial \mathbf{x}} \approx \frac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}}\mathbf{F}^{-1} \tag{4.1.14}$$

### 4.1.2.4 Divergence

Starting from the discrete integral for divergence in the reference volume (4.1.7), the expression for deformed divergence takes into account the effect of the deformation on the volume (2.2.5) and the face normals (2.2.6).

$$\nabla \cdot \mathbf{u} \approx \frac{1}{J}\sum_j^4 \mathbf{u}_j \cdot J_j \mathbf{F}_j^{-\top}\widehat{\mathbf{n}}_j \tag{4.1.15}$$

where $J$ is the determinant at the centre of the volume. The relevant deformation gradients can be calculated from vertex positions and since the multiplication $J_j\mathbf{F}_j^{-\top}\widehat{\mathbf{n}}_j$ simply selects columns, it is not necessary to perform full matrix inversion. The relevant column can be calculated directly via (4.1.10) at the cost of one cross product.

For a tensor field $\mathbf{U}$ the process is similar, applying (4.1.8) in the deformed configuration gives:

$$\nabla \cdot \mathbf{U} \approx \frac{1}{J}\sum_j^4 J_j \mathbf{U}_j \mathbf{F}^{-\top}\widehat{\mathbf{n}}_j, \tag{4.1.16}$$

### 4.1.2.5 Laplacian

Calculation of the Laplacian follows the same steps as for the reference volume, making use of the self-dual grid as shown in Figure 4.5. First, the vertex coordinates and values (black dots) are used to calculate the coordinates (4.1.1) and gradient (4.1.14) at the centres of the volumes (white dots) adjacent to the vertex $\mathbf{x}$. The values at the volume centres are then used to calculate the deformed divergence of the resulting tensor field at the vertex $\mathbf{x}$ itself via (4.1.16).

Under deformation however, the centre of the dual volume may not coincide with the vertex $\mathbf{x}$. There are various approaches to this problem: Firstly, to calculate the position of the vertex in the reference volume (by solving a quadratic equation) and calculate the gradient and deformation at this point via (4.1.3).
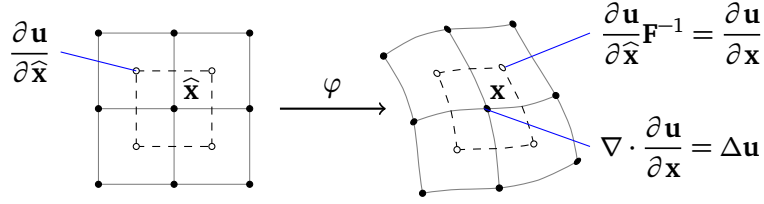
Figure 4.5: Calculation of the deformed Laplacian operator. First vertex values and coordinates (black dots) are used to calculate deformed gradients at the volume centres (white dots), then these are used to calculate the deformed divergence at vertex **x**.

Secondly, the position of the vertex can be moved to centre of the dual volume. This is effectively a form of damping, which is in line with the continuum assumption. Thirdly, the deformed Laplacian can be applied without adjustment, which is in line with the assumption that the deformation is constant over the volume. In practice, all three approaches are reasonable.

## 4.2   Linear Solver

The algorithm described in the following Chapter requires the solution of various linear systems which arise from the discretisation of differential operators. We thus give a brief description of the matrix-free assembly and solution of a deformed vector Laplacian operator as defined in Section 4.1.2.5.

This solver serves as a prototype for all of the linear solvers used in the algorithm, and is applied both to the primal mesh and its dual. That is the Laplacian may be solved both with respect to vertex properties, such as velocity in the application of viscosity, or with repect to volumes as in the case of pressure or electrical charge.

### 4.2.1   Jacobi Method

The linear solver is based on the standard Jacobi method, which is parallel in its execution. As discussed in Chapter 3, race conditions in GPU memory will cause the solver to degenerate into a Gauss-Seidel smoother, which has faster convergence, but when this occurs depends upon the scheduler of the GPU and can not be controlled. Many discussions of iterative solvers include [Golub, 2013], [Briggs et al., 2000], and [Trottenberg et al., 2000].

The Jacobi method solves the linear system $\mathbf{Au} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{u}, \mathbf{b} \in \mathbb{R}^n$. The method considers the lower triangular, diagonal and upper triangular

parts of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}. \tag{4.2.1}$$

Given some $k$-th approximation $\mathbf{u}^k$ the following update step

$$\mathbf{u}^{k+1} = \mathbf{D}^{-1}\left[\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{u}^k\right], \tag{4.2.2}$$

converges to the solution under the following condition on the spectral radius $\rho$ of the iteration matrix:

$$\rho(\mathbf{D}^{-1}[\mathbf{L} + \mathbf{U}]) < 1. \tag{4.2.3}$$

It is sometimes preferable to apply a damped Jacobi update according to the parameter $\alpha \in \mathbb{R}$, usually chosen such that $\alpha \in [0, 1]$ generates a convex combination of the current and previous iterates:

$$\mathbf{u}^{k+1} = (1 - \alpha)\mathbf{u}^k + (\alpha)\mathbf{D}^{-1}\left[\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{u}^k\right]. \tag{4.2.4}$$

### 4.2.1.1 Implicit Euler

When the Jacobi iteration is employed with respect to the Implicit Euler time integration method there is an outer iteration over time $\mathbf{u}^t \to \mathbf{u}^{t+1}$, and in inner iterate for the solution of the system $\mathbf{u}^k \to \mathbf{u}^{k+1}$. We derive the Jacobi step as follows, with operator $\mathbf{A}$ and generic constant $\alpha = \nu \frac{\delta t}{\delta x^2}$:

$$\begin{aligned} \mathbf{u}^{t+1} &= \mathbf{u}^t + \alpha \mathbf{A}\mathbf{u}^{t+1} & (4.2.5) \\ (\mathbb{I} - \alpha \mathbf{A})\mathbf{u}^{t+1} &= \mathbf{u}^t & (4.2.6) \\ [\mathbb{I} - \alpha \mathbf{D} - \alpha(\mathbf{L} + \mathbf{U})]\mathbf{u}^{t+1} &= \mathbf{u}^t & (4.2.7) \end{aligned}$$

Separating the diagonal gives an expression for the update to $\mathbf{u}^k$, the iterative the solution of $\mathbf{u}^{t+1}$ which is used frequently in the algorithm:

$$\mathbf{u}^{k+1} = [\mathbb{I} - \alpha \mathbf{D}]^{-1}\left[\mathbf{u}^t + \alpha(\mathbf{L} + \mathbf{U})\mathbf{u}^k\right]. \tag{4.2.8}$$

It is worth noting that the operator includes the identity matrix $\mathbb{I}$ and the value $\alpha$ decreases with decreasing time step length. As a result, the operator is diagonally dominant and thus well-conditioned for small time steps and the Jacobi iteration converges relatively quickly.

### 4.2.2 Matrix-Free Assembly

The iterative solver is applied in a matrix-free form that is parallel in matrix rows. Thus, each GPU thread updates a single element of the solution vector, in accordance with good practice for memory access. It must therefore generate
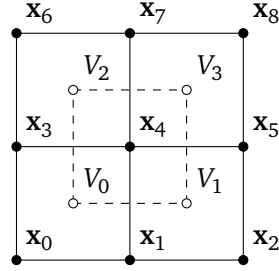
Figure 4.6: Node numbering for assembly of the Laplacian operator.

the coefficients corresponding to a single row of the Laplacian matrix. We begin by considering the assembly of the Laplacian operator *on paper* and show that its implementation reduces to a computationally efficient calculation of stencil weights which can be expressed in closed form.

As per the Figure 4.5 the discrete formulation of the Laplacian operator in $\Omega \subset \mathbb{R}^2$ makes use of nine vertices with a point $\mathbf{x}$ at their centre. The implementation in $\mathbb{R}^3$ follows the same basic construction using 27 mesh points.

For a vector function $\mathbf{u} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ as defined earlier, the vertices and volume centres are numbered according to Figure 4.6. The the coefficients corresponding to the row of a matrix can be synthesised by concatenating the 9 function evaluations $\{\mathbf{u}_0 \ldots \mathbf{u}_8\}$ at vertices $\{\mathbf{x}_0 \ldots \mathbf{x}_8\}$ into a matrix $\mathbf{U} \in \mathbb{R}^{2 \times 9}$. The elements of $\mathbf{U}$ are the only values that change from one Jacobi iteration to the next:

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \ldots & \mathbf{u}_8 \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{19} \\ u_{21} & \cdots & u_{29} \end{bmatrix}. \tag{4.2.9}$$

The undeformed gradient operator as per (4.1.4) is applied as a difference matrix $\mathbf{D}_1 \in \mathbb{R}^{9 \times 8}$ to the vertex values, resulting in a matrix of four blocks corresponding to the four reference Jacobians at the centre of each volume $\{V_0, \ldots, V_3\}$.

$$\begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_8 \end{bmatrix} \frac{1}{2} \underbrace{\begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ +1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & 0 & 0 & 0 & 0 \\ -1 & +1 & 0 & 0 & -1 & -1 & 0 & 0 \\ +1 & +1 & -1 & +1 & +1 & -1 & -1 & -1 \\ 0 & 0 & +1 & +1 & 0 & 0 & +1 & -1 \\ 0 & 0 & 0 & 0 & -1 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & +1 & -1 & +1 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & +1 \end{bmatrix}}_{\mathbf{D}_1} \tag{4.2.10}$$

$$= \begin{bmatrix} \left. \dfrac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} \right|_{V_0} & \left. \dfrac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} \right|_{V_1} & \left. \dfrac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} \right|_{V_2} & \left. \dfrac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}} \right|_{V_3} \end{bmatrix}$$

Next, the four inverse deformation gradients at each volume centre $\mathbf{F}_V^{-1}$, calculated via (4.1.9) are applied in block form $\mathbf{F}_1 \in \mathbb{R}^{8\times 8}$ for the deformed Jacobians at the volume centres:

$$
\left[\begin{array}{ccc} \left.\dfrac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}}\right|_{V_0} & \cdots & \left.\dfrac{\partial \mathbf{u}}{\partial \widehat{\mathbf{x}}}\right|_{V_3} \end{array}\right] \underbrace{\left[\begin{array}{cccc} \mathbf{F}_{V_0}^{-1} & 0 & 0 & 0 \\ 0 & \mathbf{F}_{V_1}^{-1} & 0 & 0 \\ 0 & 0 & \mathbf{F}_{V_2}^{-1} & 0 \\ 0 & 0 & 0 & \mathbf{F}_{V_3}^{-1} \end{array}\right]}_{\mathbf{F}_1} = \left[\begin{array}{ccc} \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{V_0} & \cdots & \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{V_3} \end{array}\right]
$$

$$(4.2.11)$$

An interpolation operator $\mathbf{D}_2 \in \mathbb{R}^{8\times 8}$ as per (4.1.1) is then applied which gives four gradients interpolated at the face midpoints $\mathbf{x}_j = \{\mathbf{x}_0 \ldots \mathbf{x}_4\}$. Here, $\mathbb{I} \in \mathbb{R}^{2\times 2}$ is the identity matrix.

$$
\left[\begin{array}{ccc} \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{V_0} & \cdots & \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{V_3} \end{array}\right] \frac{1}{2} \underbrace{\left[\begin{array}{cccc} \mathbb{I} & 0 & \mathbb{I} & 0 \\ 0 & \mathbb{I} & \mathbb{I} & 0 \\ \mathbb{I} & 0 & 0 & \mathbb{I} \\ 0 & \mathbb{I} & 0 & \mathbb{I} \end{array}\right]}_{\mathbf{D}_2} = \left[\begin{array}{ccc} \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} & \cdots & \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{\mathbf{x}_4} \end{array}\right]
$$

$$(4.2.12)$$

Finally, the products of the gradients and the deformed normals are summed to give the divergence of the gradient. This is achieved by arranging the deformed face normals into a column vector, given by the product of the four deformation gradients at the face midpoints $\mathbf{F}_2 \in \mathbb{R}^{8\times 8}$ divided by the volume determinant $J$, and the reference outer normal vectors $\mathbf{N}_1 \in \mathbb{R}^{8\times 1}$ :

$$
\left[\begin{array}{ccc} \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} & \cdots & \left.\dfrac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{\mathbf{x}_4} \end{array}\right] \frac{1}{J} \underbrace{\left[\begin{array}{cccc} J_0 \mathbf{F}_0^{-\top} & 0 & 0 & 0 \\ 0 & J_1 \mathbf{F}_1^{-\top} & 0 & 0 \\ 0 & 0 & J_2 \mathbf{F}_2^{-\top} & 0 \\ 0 & 0 & 0 & J_3 \mathbf{F}_3^{-\top} \end{array}\right]}_{\mathbf{F}_2} \underbrace{\left[\begin{array}{c} \widehat{\mathbf{n}}_0 \\ \widehat{\mathbf{n}}_1 \\ \widehat{\mathbf{n}}_2 \\ \widehat{\mathbf{n}}_3 \end{array}\right]}_{\mathbf{N}_1} \approx \Delta \mathbf{u}(\widehat{\mathbf{x}})
$$

$$(4.2.13)$$

In summary, the forward evaluation of the discrete deformed Laplacian is a composition of matrix products:

$$\Delta \mathbf{u} \approx \mathbf{U}\mathbf{D}_1\mathbf{F}_1\mathbf{D}_2\mathbf{F}_2\mathbf{N}_1. \tag{4.2.14}$$

We recall that the only values that change per Jacobi iteration are the elements of $\mathbf{U}$, while $\mathbf{D}_1$ and $\mathbf{D}_2$ are static difference and interpolation coefficients. The elements of $\mathbf{F}_1$ and $\mathbf{N}_1$ depend upon deformation gradients which only change per time step. Taking the product of all these static matrices gives a vector $\mathbf{c} \in \mathbb{R}^{9\times 1}$ of coefficients:

$$\underbrace{\mathbf{D}_1}_{9\times 8} \underbrace{\mathbf{F}_1}_{8\times 8} \underbrace{\mathbf{D}_2}_{8\times 8} \underbrace{\mathbf{F}_2}_{8\times 8} \underbrace{\mathbf{N}_1}_{8\times 1} = \underbrace{\mathbf{c}}_{9\times 1} \tag{4.2.15}$$

These are the entries of the row of the discrete Laplacian matrix, and are applied as weights to the columns of $\mathbf{U}$.

The assembly and multiplication of these matrices can be largely avoided if system is evaluated by composition from right to left, rather than left to right.

$$\underbrace{\mathbf{U}}_{2\times 9} \mathbf{D}_1 \mathbf{F}_1 \mathbf{D}_2 \underbrace{\underbrace{\underbrace{\underbrace{\mathbf{F}_2 \mathbf{N}_1}_{8\times 1}}_{8\times 1}}_{8\times 1}}_{9\times 1} = \underbrace{\Delta\mathbf{u}}_{2\times 1} \qquad (4.2.16)$$

As a result it is reduced to a small number of signed sums of difference quotients and entries of deformation matrices, which are easily hard-coded into computational kernels. This is both computationally intensive and low in memory bandwidth, ideal for GPU computation.

## 4.3   Geometry

### 4.3.1   Signed Distance Function

The geometry in the model is encoded via Signed Distance Functions (SDF). This includes not only the solid and fluid domains, but regions of differing fibre direction and electrical conductivity, amongst others. Although the SDF is used here, the voxels that are used in the solution algorithm can be obtained directly from segmented medical imaging data.

The SDF is a scalar function $s : \mathbb{R}^d \to \mathbb{R}$ over the domain that evaluates to a negative value on the interior of some subdomain and a positive value on its exterior. The boundary or surface is the set of points for which the function evaluates to zero. Figure 4.7 gives an example of a signed distance function for an intersecting circle and square, showing the positive and negative values of the function, as well as the boundary.

The use of SDFs originated in computer graphics since they have the important property that the gradient of the function on the boundary of the object is the exterior normal to the surface $\nabla s(\mathbf{x})|_{s(\mathbf{x})=0} = \mathbf{n}$. This is extremely useful when rendering scenes because it allows for the calculation of the paths of reflected light rays, during the application of lighting and surface effects, but is also ideal for the calculation of surface flux.

Mathematically the function derives from a norm. Consider a point $\mathbf{c} \in \mathbb{R}^d$. The Euclidean distance from $\mathbf{c}$ to any point $\mathbf{x} \in \mathbb{R}^d$ is given by the 2-norm $\|\mathbf{x} - \mathbf{c}\|_2$. If a scalar radius $r \in \mathbb{R}$ is subtracted from this distance the resulting
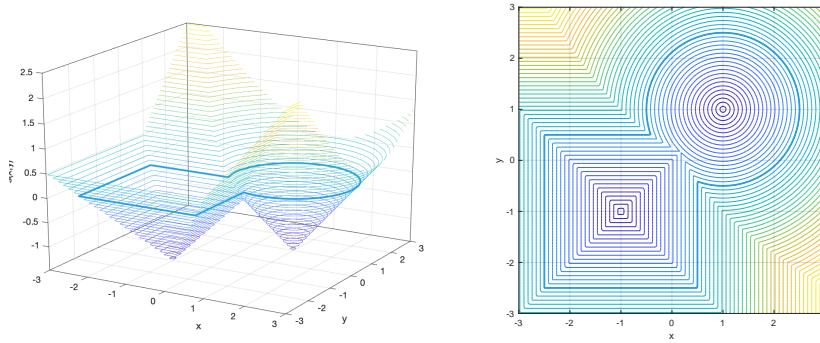
Figure 4.7: 2D and 3D contour plots of the same Signed Distance Function $s(x, y)$. The function takes negative values on the interior of an object and positive values on its exterior. The surface (blue line) is defined as the set of points for which the value of the function is zero.

function $s(\mathbf{x}, \mathbf{c}, r) = \|\mathbf{x} - \mathbf{c}\|_2 - r$, defines a $d$-sphere with centre $\mathbf{c}$ and radius $r$. It is possible to define other functions based upon norms that define different shapes, for instance the 1-norm $\|\mathbf{x} - \mathbf{c}\|_1$ defines a $d$-cube.

SDFs have several properties which are valuable in computation:

They can be evaluated at any point and thus any resolution. This allows the GPU to combine information stored at different resolutions in a straightforward manner, as outlined in section 3.2.2. It is also ideal for multigrid solvers, since the entire problem can be fully reconstructed automatically on coarse grids.

They can be easily derived from medical imaging data, either by thresholding or by standard algorithms applied to voxel data files which are the output of MRI or CT scanners.

SDFs degrade to circles at long distances. This allows the approximation of shapes which is useful for clipping (the exclusion of distant objects) or for collision detection and multi-body problems.

### 4.3.2  Primitives

Signed distance fields can be derived for various shapes and combined algebraically, allowing for the construction of complex shapes from simple objects. Much of this section is derived from the website https://iquilezles.org, although the functions have been adapted for simplicity.

Some examples of simple shapes, or primitives are shown in Figure 4.8 with code in Listing 4.3.2. In each case the vector argument p is the coordinate of the point to be tested, and the function returns a scalar value (positive, negative
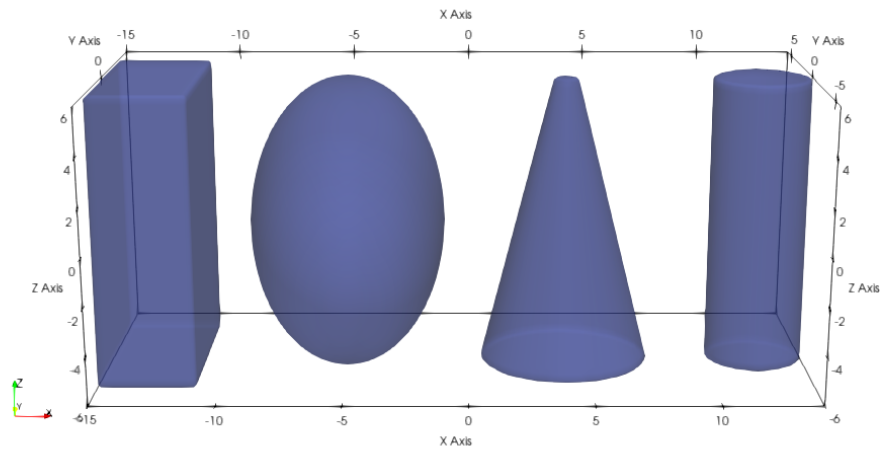
Figure 4.8: Cuboid, Ellipsoid, Cone and Cylinder primitives.

or zero) dependent upon the location of the point with respect to the shape. The functions are mathematically simple and SIMT/SIMD parallel.

```
//cuboid, p=point, r=radius (per axis)
float sdf_cub(float3 p, float3 r)
{
    float3 q = fabs(p/r) - 1.0f;

    return max(q.x,max(q.y,q.z));
}

//ellipsoid, p=point, r=radius (per axis)
float sdf_ell(float3 p, float3 r)
{
    return length(p/r) - 1.0f;
}

//cylinder, p=point, r=radius, h=height
float sdf_cyl(float3 p, float r, float h)
{
    return max(length(p.xy) - r, fabs(p.z) - h);
}

//cone, p=point, r=radius, h=height, a=angle
float sdf_con(float3 p, float r, float h, float a)
{
    return max(length(p.xy) - a*p.z - r, fabs(p.z) - h);
}
```

Any affine or nonlinear transformation can be applied to the point p and thus the shape can be translated and deformed as required. It is also possible to apply modulo functions to the coordinate to allow tiling.

The combination of primitive shapes is achieved by the implementation of sim-

ple set operations. We consider the two shapes $\mathcal{S}_1, \mathcal{S}_2$ and their respective SDFs $s_1, s_2 \in \mathbb{R}$ and implement the following set operations as follows:

$$
\begin{array}{llll}
\text{Union:} & \mathcal{S}_1 \cup \mathcal{S}_2 & \equiv & \min(s_1, s_2) \\
\text{Intersection:} & \mathcal{S}_1 \cap \mathcal{S}_2 & \equiv & \max(s_1, s_2) \\
\text{Complement:} & \mathcal{S}_1 \backslash \mathcal{S}_2 & \equiv & \max(s_1, -s_2)
\end{array} \tag{4.3.1}
$$

It is also possible to define hollow shapes thresholding the absolute value of the SDF. Figure 4.9 shows set operations acting on a hollow cube $\mathcal{S}_1$ and sphere $\mathcal{S}_2$. The code required to generate complex shapes and subregions is extremely simple.
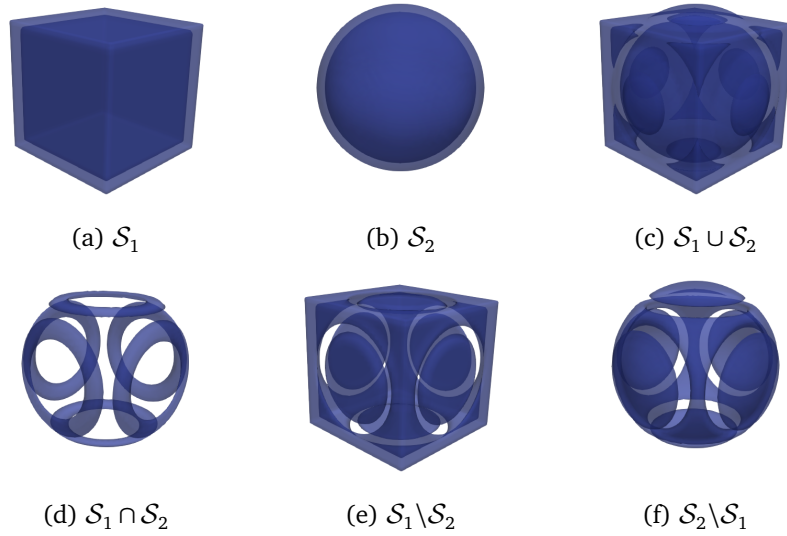


(a) $\mathcal{S}_1$                    (b) $\mathcal{S}_2$                    (c) $\mathcal{S}_1 \cup \mathcal{S}_2$

(d) $\mathcal{S}_1 \cap \mathcal{S}_2$                    (e) $\mathcal{S}_1 \backslash \mathcal{S}_2$                    (f) $\mathcal{S}_2 \backslash \mathcal{S}_1$

Figure 4.9: Set operations on Signed Distance Functions for a hollow cube $\mathcal{S}_1$ and sphere $\mathcal{S}_2$.

A smoothed minimum function can be used to blend together two SDFs leading to more natural surfaces that better represent organic objects. An example is shown in Figure 4.10, with a varying smoothness parameter $k$.
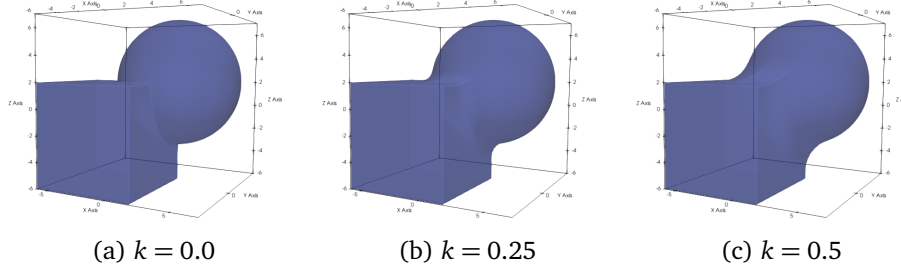
(a) $k = 0.0$                         (b) $k = 0.25$                         (c) $k = 0.5$

Figure 4.10: Smoothed minimum function for increasing values of parameter $k$.

### 4.3.3  Encoding

Having generated or captured a geometry it is now possible to encode it onto the mesh as detailed at the beginning of this chapter. Since the computational model makes use of both vertices and volumes, it important that the geometry has integrity with respect to these objects. As such a mesh volume $V$ can only belong to either the solid $\Omega_s$ or fluid $\Omega_f$ domains. A vertex, however, has no volume and can belong to one or the other, or both if it is on the fluid-structure interface.



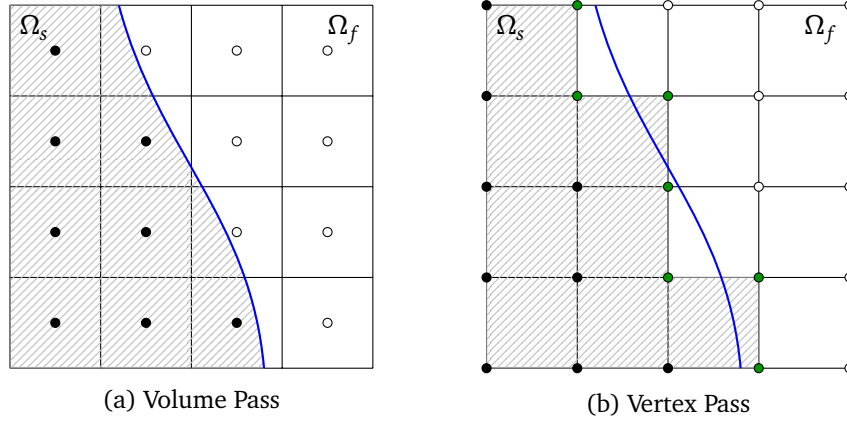(a) Volume Pass                                   (b) Vertex Pass

Figure 4.11: The SDF is encoded into the mesh in two stages. In the first pass the SDF is evaluated at the volume centres. In the second, vertices count incident volumes and are assigned as solid (black), fluid (white) or surface (green).

The algorithm that applies the SDF to the mesh thus makes two passes, in parallel over the mesh as follows:

1.  A kernel runs per volume, evaluating the SDF at the centre point of each. As such the volumes $V$ are assigned to either the solid or fluid subdomains.

2. The next kernel runs per vertex **x** and iterates through all subvolumes *V* incident to it. If all the adjacent subvolumes are in the solid subdomain the vertex is assigned as solid, similarly if all are fluid then the vertex is assigned as fluid. If the vertex is incident upon both solid and fluid subvolumes it is assigned as surface.

Figure 4.11 illustrates the relationship between the SDF and the mesh.

A discussed in Chapter 3, GPU memory access is SIMD parallel and it is efficient to store a 3-dimensional vertex coordinate $\mathbf{x} \in R^3$ in a `float4` vector variable[1]. The fourth float in the vector is populated with the value of the SDF and it is thus available to test during any kernel execution that calls the vertex coordinates into register without cost.

The geometry kernels also populate any other spatially-dependent data fields, such as the fibre directions, conductivity and varying tissue properties needed by the calculation.

Having assigned volumes and vertices, the geometry is encoded as voxel data in a regular structured grid. This is illustrated in 3D in Figure 4.12. When the input data is voxel data from a medical scan, this reflects the incoming data exactly. It would also be possible at this stage to fit the mesh precisely to the SDF. This would be achieved by deforming the mesh, such that surface vertices were moved to the root of the SDF. This would generate a smooth surface. Then the application of a Laplacian to the non-surface vertex coordinates would regularise the volumes. This has been successfully tested but is not yet implemented, since subsequent calculations would require extra storage and reference to the pre-deformed mesh.
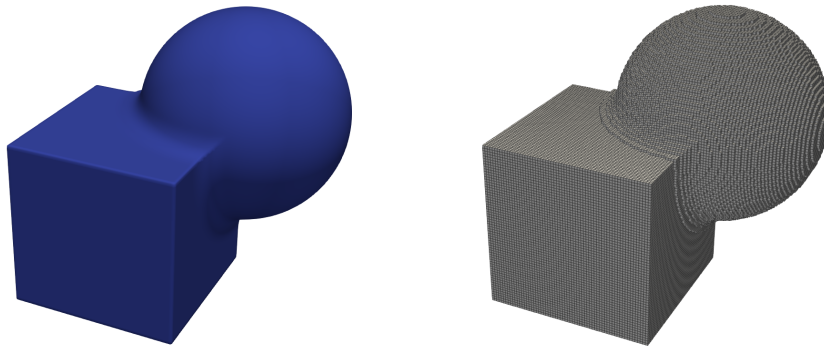


Figure 4.12: A Signed Distance Function and its encoding into the mesh.

---

[1]In OpenCL there is a `float3` variable, but it is an alias to `float4`.

# Chapter 5

# Algorithm Details

## 5.1 Overview

As described in Chapter 4, the discrete problem is processed in space on a regular structured grid with mesh width $\delta x \in \mathbb{R}$ and iterated over time $t \in \mathbb{R}^+$. The memory for the calculation is a set of three-dimensional buffers of the OpenCL data type `float4`, which are vectors of four single-precision floating point values. Some of the buffers correspond to values at the vertices while others contain information relating to volumes.

The C program instantiates OpenCL objects, including the memory buffers on the GPU and their contents at time $t = 0$ are initialised according to input data, which includes the geometry and initial conditions. A main loop then iterates over a time step $\delta t \in \mathbb{R}$ and applies a sequence of computational *kernels* which operate in parallel on the buffers, updating values relevant to the different steps of the algorithm. Kernels such as iterative solvers are applied in sub-loops to the main loop.

At prescribed intervals, the contents of the buffers are copied from the CPU to GPU and written into output files. These files are post-processed and visualised as frames of video or plots of time series data.

## 5.2 Electophysiology

The Monodomain equation (5.3.2) is a PDE consisting of both reactive and diffusive terms, which are handled separately via *operator-splitting*. The ODE corresponding to the Mitchell-Scheaffer Model (2.1.1) for transmembrane potential is first updated via a step of Explicit Euler time integration, and this value is

then used to update the diffusive part of the Monodomain equation via a step of Implicit Euler time integration. The regime is detailed below.

### 5.2.1 Reaction

First, a single kernel integrates the Mitchell-Scheaffer ODE with a step of Explicit Euler time integration, acting on the coupled scalar variables for transmembrane voltage $v$ in (2.1.1) and the gating variable $w$ in (2.1.4), which are stored as average values per volume:

$$v_i^{t+1} \;\; = \;\; v_i^t + \delta t \frac{dv_i}{dt} \tag{5.2.1}$$

$$w_i^{t+1} \;\; = \;\; v_i^t + \delta t \frac{dw_i}{dt}, \tag{5.2.2}$$

where in general, the discrete evaluation $v_i$ of a continuous variable $v$ is given a suffix $i \in \mathbb{N}$ which enumerates a vertex or, in this case Volume. The memory buffer contains a flag which indicates whether a volume is electrically active or not, in which case its value is preserved.

### 5.2.2 Diffusion

Next a kernel applies an Implicit Euler step to the left hand side of (5.3.2) corresponding to the diffusion of charge through the intracellular and extracellular domains. The resulting linear system is solved with a Jacobi iteration as described in Section 4.2.1.1, again acting on the average value of $v_i$ per volume.

$$v_i^{t+1} = v_i^t + \delta t \mathbf{A} v_i^{t+1} \tag{5.2.3}$$

The solver must take into account the conductivity tensor $\Sigma$ calculated from the fibre directions $\mathbf{f}$ as in (2.1.24). Since the conductivity tensor acts on the eight discrete gradients $\nabla v$ surrounding $v_i$, it is derived from interpolated values of $\mathbf{f}$ at the corresponding points and appears in the assembly described in Section 4.2.2 as an additional block diagonal matrix $\mathbf{S}_1$ of eight conductivity tensors:

$$\nabla \cdot (\Sigma \nabla v_i) \approx \mathbf{v}_i \mathbf{D}_1 \mathbf{S}_1 \mathbf{F}_1 \mathbf{D}_2 \mathbf{F}_2 \mathbf{N}_1, \tag{5.2.4}$$

where $\mathbf{v}_i$ is a row vector of the 27 values surrounding $v_i$ that are used by the stencil. The buffer of fibre directions uses all four elements of the `float4` variable such that the unit direction vector $\mathbf{f} \in \mathbb{R}^3$ and the longditudinal $\sigma_L$ and transverse $\sigma_T$ conductivities can be stored uniquely per volume $V_i$ of the domain.

The model assumes that the fibres $\mathbf{f}$, and thus the conductivity field $\Sigma$, deform with the mesh. The deformation gradients $\mathbf{F}_1, \mathbf{F}_2$ therefore need only to take

into account the scaling and are thus the product of mesh width $\delta x$ and an identity matrix $\mathbb{I} \in \mathbb{R}^{3\times3}$. Since they are applied twice during assembly they can be removed and appear as the familiar scalar term $\frac{1}{\delta x^2}$. For the generation of the ECG, the elliptic part of the problem in the second line of (2.1.19) is solved on the heart and torso via Jacobi iteration.

## 5.3   Solid Dynamics

### 5.3.1   Stress

In Section 5.3.1 it was shown that the principles of conservation of mass and momentum, along with a constitutive model for a solid continuum lead to the Elastodynamics equation (2.3.8). The resultant force acting on an enclosed volume $V$, is thus the sum of traction and body forces, given by surface and volume integrals respectively.

In the discrete case, these integrals are calculated in parallel per vertex as shown in Figure 5.1.
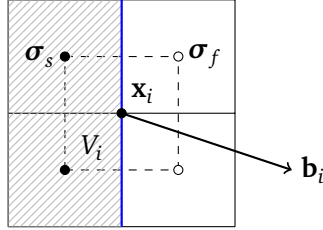


Figure 5.1: Spatial integration of the Elastodynamics equation at surface point $\mathbf{x}_i$ lying on the fluid structure interface $\Gamma$ (blue line), showing the volume of integration $V_i$ (enclosed by dotted lines), and examples of the domain dependent stress tensors $\boldsymbol{\sigma}_s$ and $\boldsymbol{\sigma}_f$ (black and white dots respectively) and body forces $\mathbf{b}_i$.

Body forces $\mathbf{b}_i$ may be prescribed as averages over $V_i$ and thus sum directly into the integral. Traction forces are given by the negative divergence of Cauchy stress. The stress tensors surrounding a vertex $\mathbf{x}_i$ may be derived in one of two different ways:

1. If the adjacent volume is part of the solid domain $\Omega_s$ then the tensor $\boldsymbol{\sigma}_s$ derives from the deformation gradient $\mathbf{F}$ in (4.1.9) and solid material model such as Saint Venant-Kirchhoff (2.3.4).

2. If the volume is in the fluid domain $\Omega_f$ then Cauchy stress $\boldsymbol{\sigma}_f$ is given by the pressure $p$ and strain-rate tensor as in the Navier-Stokes fluid (2.4.6).

In practice the kernels loop the volumes adjacent to the vertex and read the geometry information from the coordinate buffer as outlined in Section 4.3.3. The numerical calculation of the stress tensors in the deformed configuration is performed as per (4.1.16). As a result, the coupling conditions for continuity of stress in the ALE formulation (2.5.8) are satisfied.

### 5.3.2   Muscle Contraction

If tissue is mechanically active the Cauchy Stress tensor must include an additional component to represent muscle contraction. This is calculated by taking the unit vector for fibre direction $\mathbf{f}$ and multiplying it by a constant to represent the strength of contraction and the dimensionless value of transmembrane potential as given by (). The components of the resulting vector are added to the diagonal of the Second Piola-Kirchhoff stress in the reference configuration (2.3.4). In this way the fibres and resulting contraction forces deform with the tissue as expected.

This is the simplest possible implementation of Excitation-Contraction coupling and can be refined in future work.

### 5.3.3   Leapfrog Integration

Once the forces in the Elastodynamics equation are resolved, conservation of momentum allows the change in velocity to be calculated by consideration of mass, the integral of density over volume $V_i$. Integration in time is then achieved with the Leapfrog method. This explicit method has been derived separately across many disciplines and is, most importantly, symplectic in phase plane, that is it conserves angular momentum. It is therefore suitable for the wave-like solutions of the Elastodynamics equation.

For position, velocity and acceleration $\mathbf{x}, \mathbf{v} = \dot{\mathbf{x}}, \mathbf{a} = \ddot{\mathbf{x}} \in \mathbb{R}^d$, in a classical dynamic system of differential equations, Leapfrog integration at time $t \in \mathbb{R}$ is as follows:

$$\begin{aligned}
\mathbf{v}_{t+\frac{1}{2}} &= \mathbf{v}_{t-\frac{1}{2}} + \delta t\, \mathbf{a}_t & (5.3.1) \\
\mathbf{x}_{t+1} &= \mathbf{x}_t + \delta t\, \mathbf{v}_{t+\frac{1}{2}} & (5.3.2)
\end{aligned}$$

It is worth noting that the position and velocity are given at different times, which gives the method its name. Also that the system is started from rest, which does not require any spcific management of the velocity half-step. The first step (5.3.1) only is applied at this stage.

### 5.3.4   Damping

Solid damping can be applied as the diffusion of momentum via an Implicit Euler iteration on momentum as per Section 4.2.1.1. Damping and fluid viscosity can be applied to the whole domain separately or via the same kernel, taking care to use the correct coefficients for dynamic viscosity, damping and density. The fluid viscosity can also be applied in the mechanics kernel as the strain rate in the Cauchy stress tensor $\boldsymbol{\sigma}_f$ of (2.4.6), and natural damping is achieved via the interpolation of vertex position, as described in Section 5.3.1.

## 5.4   Fluid Dynamics

### 5.4.1   Helmholz Projection

The projection method that is at the core of most modern fluid dynamics solvers was pioneered by Chorin [Chorin and Marsden, 2013] and is the basis for the well-known SIMPLE algorithm [Patankar, 1980] and its subsequent variations. This section outlines the technique, following a useful introductions by [Harris, 2005] and [Griebel, 1997], and those following show how it must be adapted to incorporate the deforming fluid-structure interface.

Helmholz-Hodge decomposition theorem states that a vector field $\mathbf{w}$ on domain $\Omega$, can be uniquely decomposed into the sum of two orthogonal fields:

$$\mathbf{w} = \mathbf{u} + \nabla p, \tag{5.4.1}$$

where $\nabla p$ is the gradient of a scalar field, and $\mathbf{u}$ is parallel to $\partial \Omega$ and has zero divergence, that is:

$$\nabla \cdot \mathbf{u} = 0, \text{ on } \Omega \tag{5.4.2}$$

$$\mathbf{u} \cdot \mathbf{n} = 0, \text{ on } \partial \Omega. \tag{5.4.3}$$

The unknown fields $\mathbf{u}$ and $p$ can be obtained from $\mathbf{w}$ by taking the divergence of (5.4.1) and applying condition (5.4.2) to give a Poisson problem for pressure:

$$\nabla \cdot \mathbf{w} = \underbrace{\nabla \cdot \mathbf{u}}_{\nabla \cdot \mathbf{u} = 0} + \nabla \cdot \nabla p \tag{5.4.4}$$

$$\Rightarrow \Delta p = \nabla \cdot \mathbf{w} \tag{5.4.5}$$

which can be solved up to the addition of a constant with the zero-Neumann boundary condition (5.4.3). Taking the gradient of the solution $p$ and substituting into (5.4.1) completes the solution:

$$\mathbf{u} = \mathbf{w} - \nabla p. \tag{5.4.6}$$

The continuity equation (2.4.2) requires that fluid velocity must be divergence-free for conservation of mass. The decomposition method is applied to the discrete time formulation of the Navier-Stokes equations (2.4.13) as follows.

We start by considering a discrete time step $\delta t$, where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity and body forces $\mathbf{b}$ are ignored:

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \delta t \left[ (\mathbf{v} \cdot \nabla)\mathbf{v} + \nu \Delta \mathbf{v} - \frac{1}{\rho}\nabla p \right]. \qquad (5.4.7)$$

The discrete advection and viscosity operators are applied in an explicit fashion to give an intermediate velocity field $\mathbf{w}$ with non-zero divergence:

$$\mathbf{w} = \mathbf{v}^t + \delta t \left[ (\mathbf{v}^t \cdot \nabla)\mathbf{v}^t + \nu \Delta \mathbf{v}^t \right]. \qquad (5.4.8)$$

The velocity at the next time step, which must be divergence-free by continuity, is expressed as a sum representing the Helmholz decomposition of $\mathbf{w}$, equivalent to (5.4.1):

$$\mathbf{w} = \mathbf{v}^{t+1} + \frac{\delta t}{\rho}\nabla p^{t+1} \qquad (5.4.9)$$

Taking the divergence as above and using $\nabla \cdot \mathbf{v}^{t+1} = 0$ as in (5.4.4) yields a Poisson equation for pressure, with divergence of intermediate velocity as data:

$$\Delta p^{t+1} = \frac{\rho}{\delta t}\nabla \cdot \mathbf{w}. \qquad (5.4.10)$$

Finally, the intermediate field $\mathbf{w}$ can be corrected by the subtraction of the gradient of the calculated pressure field $p^{t+1}$ as in (5.4.6):

$$\mathbf{v}^{t+1} = \mathbf{w} - \frac{\delta t}{\rho}\nabla p^{t+1}. \qquad (5.4.11)$$

The explicit integration of advection and viscosity with the implicit solution of the Poisson pressure equation constitutes another *operator-splitting* approach, and the update of the intermediate field into its divergence-free form is a *predictor-corrector* method.

### 5.4.2  Advection

The discrete advection operator must also consider the computational mesh of the ALE formulation as described in (2.5.4). It follows the conservation form of the advection operator and applies the deformed divergence as in (4.1.16).

$$\frac{\partial \mathbf{v}_f}{\partial t} + \nabla \cdot (\mathbf{v}_f \otimes (\mathbf{v}_f - \mathbf{v}_c)) = 0, \qquad (5.4.12)$$

where $\otimes$ indicates an outer product. This is a sum of matrix vector products with outer normals.

$$\mathbf{v}_f^{t+1} = \mathbf{v}_f^t - \frac{1}{J} \sum_j^4 J_j (\mathbf{v}_f^t \otimes (\mathbf{v}_f^t - \mathbf{v}_c^t)) \mathbf{F}^{-\top} \widehat{\mathbf{n}}_j, \qquad (5.4.13)$$

At the same time as the computation we test the sign of the product $\mathbf{v}_f^t \cdot \mathbf{n}_j > 0$ and remove from the sum those instances where it is true. This results in an upwind scheme for the advection operator.

### 5.4.3   Viscosity

Similarly to damping, viscosity can be applied as via an implicit Euler step 4.2.1.1 for momentum on the fluid and surface domains. In situations where the Reynolds number is relatively low, viscosity can be applied explicitly in the fluid strain rate tensor as part of the fluid Cauchy stress $\boldsymbol{\sigma}_f$ as given in (2.4.6). The discrete deformed gradients of fluid velocity are calculated as per (4.1.14) and weighted by the constant of dynamic viscosity. The stress tensor is applied conditionally as outlined in Section 5.3.1.

### 5.4.4   Pressure

The Helmholz projection method of Section 5.4.1, requires only the gradient of pressure to correct fluid velocity. The definition of fluid Cauchy stress $\boldsymbol{\sigma}_f$ as given in (2.4.6) requires an absolute value for pressure. The Dirichlet boundary conditions necessary for the correct solution of the Poisson pressure equation. as thus contained in ghost cells at the perimeter of the domain. They are set at the beginning of the time iteration and can represent actual values or simply a ground state against which relative pressure is measured.

The application of the Helmholz projection method proceeds in four stages:

1.  Given an initial velocity $\mathbf{v}_f$, the advection and viscosity operators are applied as described above, to gain an intermediate velocity field.

2.  The discrete divergence of velocity is calculated as per (4.1.15), taking values at the mesh vertices and returning an average divergence at the volume centres.

3.  The divergence is given as data to the right hand side of a Jacobi iteration (4.2.2) for the solution of the poisson pressure (5.4.11) on the fluid domain.

4.  The resulting value of $p$ is used in the fluid Cauchy stress tensor (4.2.2) and applied to solid or fluid as necessary. In this way the continuity of

stress between the solid and fluid (2.5.8) is enforced as well as the fluid pressure correction (5.4.6).

In the derivation of the Helmholz projection method, the zero-Neumann boundary condition on $\mathbf{u} \cdot \mathbf{n} = 0$ on $\partial\Omega$ in 5.4.3 is applied to the fluid velocity in the Poisson pressure equation. In the ALE formulation the velocity may not be zero, since the *no-slip* condition (2.5.7) allows for the coupled movement of solid and fluid on $\partial\Omega$. As a result a non-zero Neumann boundary condition is applied to the Poisson pressure equation as per [Chorin and Marsden, 2013]:

$$\frac{\partial p}{\partial n} = \mathbf{w} \cdot \mathbf{n}, \text{ on } \partial\Omega. \qquad (5.4.14)$$

In practice the pressure field from the previous time step is used as the initial guess for the Jacobi iteration. The resulting pressure field is also under-relaxed aiding stability on the fluid-structure interface.

## 5.5   Computational Mesh

The computational mesh does not play any physical role in the computation but it must deform with the solid structures in order to maintain the admissibility if the motion as mentioned in Section 2.2.1. Effectively the determinant $J$ of the deformation gradient $\mathbf{F}$ should be greater than zero on $\Omega_c$.

The motion of the computational mesh is resolved by applying Jacobi iteration (4.2.2) of a Laplace equation to the computational velocity $\mathbf{v}_c$, using the solid velocity at the surface and ghost cells as Dirichlet boundary conditions. This is in line with the coupling condition for velocity on the fluid-structure interface (2.5.7).

At this point in the algorithm the second step of the Leapfrog iteration is applied (5.3.2), updating the position of the solid and computational mesh, and the main loop returns for its next iteration.

# Part III

# Results

# Chapter 6

# Numerical Experiments

## 6.1 Convergence Studies

### 6.1.1 Overview

We start by validating the discretization as introduced in Chapter 4. Convergence studies in the $L_2$ error norm with respect to mesh width $\delta x$ are carried out for the forward action of the various discrete differential operators, as well as the iterative solution of a discretized Laplace operator. All of the operations are found to be quadratically convergent as expected.

The single precision arithmetic introduces a lower bound to the mesh width, below which results become numerically unstable. This is because the definition of the divergence operator (4.1.15) includes a division by the measure of the control volume, $J \approx \frac{1}{\delta x^3}$, which quickly tends to machine zero in single precision arithmetic.

We consider the vector-valued objective function $\mathbf{u} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, where $\mathbf{x} = [x, y, z]^\top$ as follows:

$$\mathbf{u}(\mathbf{x}) = \begin{bmatrix} \sin(x)\cos(y) \\ \cos(y)e^z \\ \mathrm{atan}(z) \end{bmatrix} \tag{6.1.1}$$

To reference domain $\widehat{\Omega} = [-1, +1]^3 \subset \mathbb{R}^3$, the following deformation $\varphi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ was applied:

$$\mathbf{x} = \varphi(\widehat{\mathbf{x}}) = \begin{bmatrix} x + \cos(\pi z) \\ y + \sin(\pi z) \\ z + \sin(z) \end{bmatrix} \tag{6.1.2}$$

The objective function is shown in reference $\mathbf{u}(\widehat{\mathbf{x}})$ and deformed $\mathbf{u}(\varphi(\widehat{\mathbf{x}}))$configurations in Figure 6.1. For each of the operators the error $\mathbf{e} \in \mathbb{R}^d$ is the difference be-

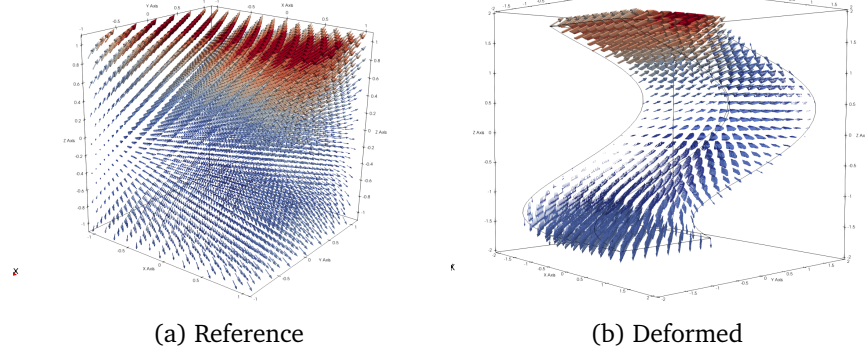(a) Reference                          (b) Deformed

Figure 6.1: Objective function $\mathbf{u}(\mathbf{x})$ for the convergence study in reference and deformed configurations, $n = 32$.

tween analytic and numeric solutions $\mathbf{u}_{\text{num}}, \mathbf{u}_{\text{ana}} \in \mathbb{R}^d$, which may be scalar or vector. The $L_2$ norm is calculated in the ususal way as the sum of the integral of squared errors over each of the control volumes $V_i$ in the the domain, where $J$ is the determinant of the deformation gradient as in (2.2.1):

$$\mathbf{e} = \mathbf{u}_{\text{num}} - \mathbf{u}_{\text{ana}} \tag{6.1.3}$$

$$\|\mathbf{e}\|_{L_2} = \left( \sum_{i=1}^{V_{\text{tot}}} J_i (\mathbf{e}^\top \mathbf{e}) \right)^{\frac{1}{2}} \tag{6.1.4}$$

It is important to note that since $J_i$ is only an approximation of the actual volume measure $|V_i|$, this introduces an error in to the norm itself. Despite this error it is still possible to draw the most important conclusions from the convergence studies below. Examples of the error norm in the reference and deformed configurations are shown in Figure 6.2.
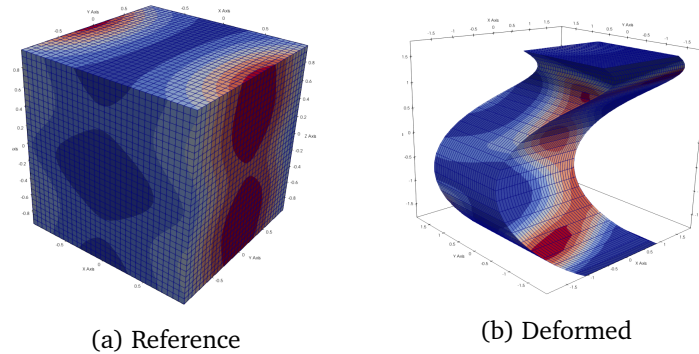


(a) Reference                          (b) Deformed

Figure 6.2: Sample $L_2$ error norm for the convergence study in reference and deformed configurations, $n = 32$.

### 6.1.2   Discrete Operators

#### 6.1.2.1   Interpolation

The discrete interpolation operator as given in (4.1.1) shows quadratic convergence with respect to mesh width. The calculation is not susceptible to any numerical instability, indeed this task represents much of the work for which the GPU is designed. Figure 6.3 shows the results in reference and deformed configurations.
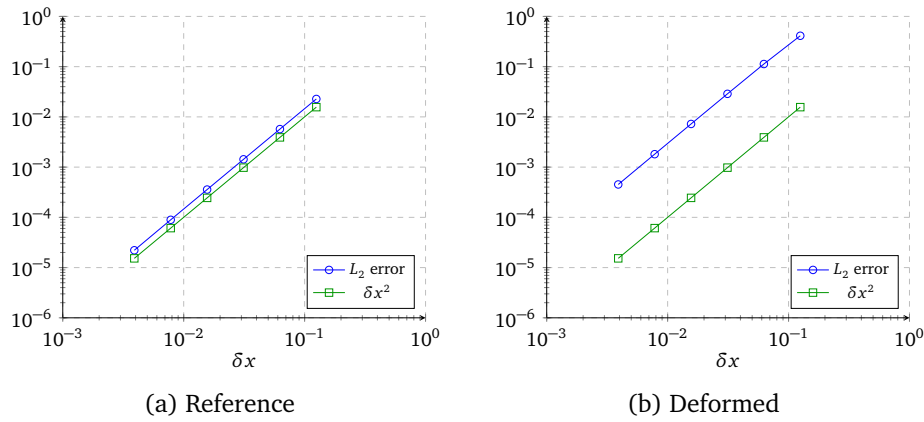


(a) Reference                                    (b) Deformed

Figure 6.3: $L_2$ error vs. mesh width $\delta x$ for the discrete interpolation operator.

#### 6.1.2.2   Gradient

The gradient operator as given in (4.1.14) again shows quadratic convergence with respect to mesh width. The results of the convergence study are shown in Figure 6.4.

In this case, since the result $\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \in \mathbb{R}^{3 \times 3}$ is a matrix, we use the Frobenius norm $e = \mathrm{tr}(\mathbf{A}^\top \mathbf{A})$ to calculate a scalar error, which is then integrated as per (6.1.4) for the $L_2$ norm.

In the reference configuration we see the beginning of some numerical instability. The calculation (4.1.14) involves multiplication by the inverse of the deformation gradient $\mathbf{F}^{-1}$, and hence division by its determinant $J$. As the mesh width $\delta x$ becomes small, the determinant tends to machine zero. The resulting division is close to zero and as a result we see an increase in the error as mesh width becomes small. In the deformed configuration there is sufficient error in the representation of the deformation itself to mask this effect.

In general the convergence of the discrete operator is quadratic, and as long as the mesh width is not too small the calculation is valid and numerically stable.
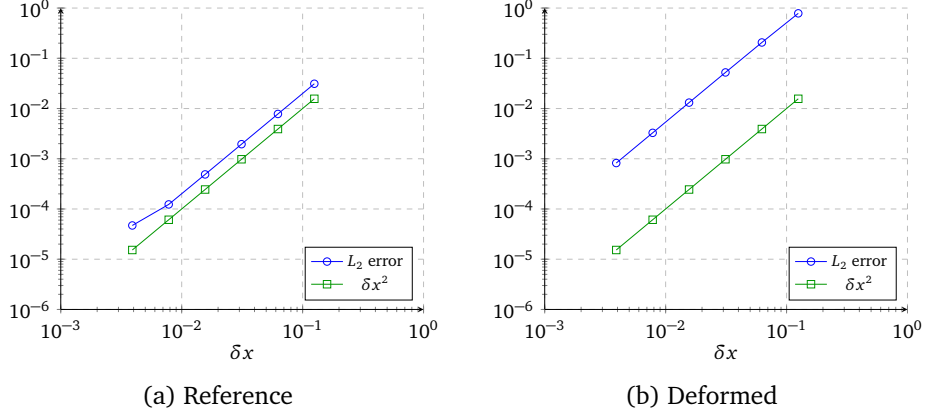


(a) Reference                                    (b) Deformed

Figure 6.4: $L_2$ error vs. mesh width $\delta x$ for the discrete gradient operator.

### 6.1.2.3  Divergence

The discrete divergence operator is given in (4.1.16) and the results of the convergence study are shown in Figure 6.5. Again, the approximation shows quadratic converge with mesh width and for similar reasons (the multiplication by $\mathbf{F}^{-\top}$) it is susceptible to numerical instability for small $\delta x$.



(a) Reference                                    (b) Deformed
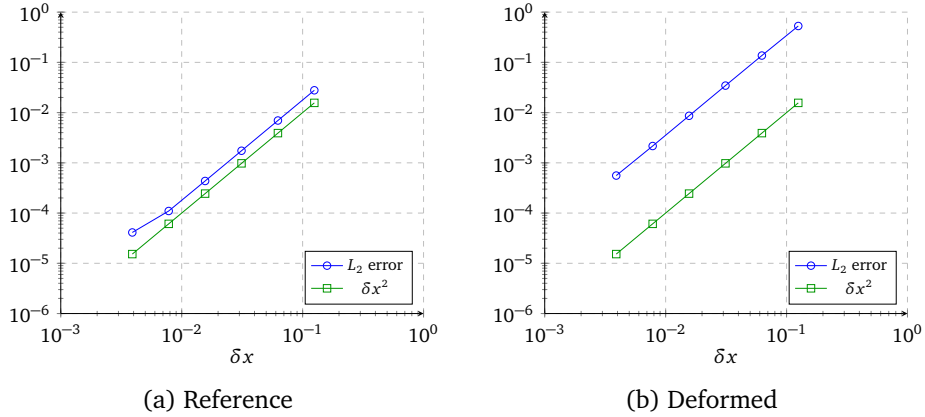
Figure 6.5: $L_2$ error vs. mesh width $\delta x$ for the discrete divergence operator.

### 6.1.2.4  Laplacian

The derivation of the discrete Laplacian operator is given in Section 4.1.2.5 and it shows the expected quadratic convergence with respect to mesh width. The results of the convergence study are shown in Figure 6.6.

Again the approximation is subject to numerical instability for small mesh widths, but now it is more pronounced. This is because the Laplacian requires the application of both gradient and divergence operators, and thus division by $J^2 \approx \delta x^6$. As a result it is important to design actual simulations with units that result in a mesh width within stable bounds. For this reason, in the simulations that follow, normal SI units are converted such that values stay within reasonable range, distances are in millimetres, rather than metres, for instance. This is an important insight from the study.
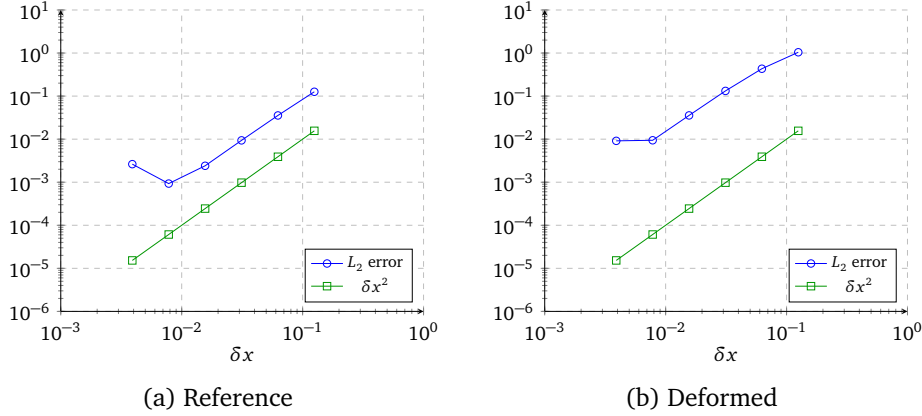


(a) Reference                                (b) Deformed

Figure 6.6: $L_2$ error norm vs. mesh width $\delta x$ for the discrete Laplacian operator.

### 6.1.3  Jacobi Solver

The details of the assembly and iterative solution of the discrete Laplace operator are given in Section 4.2. In this case we give the analytic values as data on the right hand side and solve for the objective function.

Figure 6.7 shows the convergence of the Jacobi solver with respect to iterations. This well-known result shows how the number of iterations required to reach convergence increases with. the number of degrees of freedom. The accuracy of the converged result with respect to mesh width is shown in Figure 6.8. As before the convergence is quadratic but there is numerical instability for small $\delta x$ due to the division by $J^2$ as for the forward operator. The effect is masked by the error introduced by the deformation but is still present, and the Jacobi solver must also be damped as per 4.2.4 which requires more iterations.

Again it is important to recognise the lower bound on mesh width, which can be mitigated by careful choice of units.

The Jacobi iterative solver is used four times during the algorithm described in Chapter 5. In the case of Electrophysiology, Damping and Viscosity it is used to solve for a step of implicit Euler time integration as per 4.2.8 and, as discussed in Section 4.2.1.1, it is well conditioned converges in relatively few iterations for small time steps. When used for the Poisson Pressure equation as discussed in Section 5.4.4, it is important to use the pressure solution from the last time step as the initial guess in order to minimise the number of iterations.
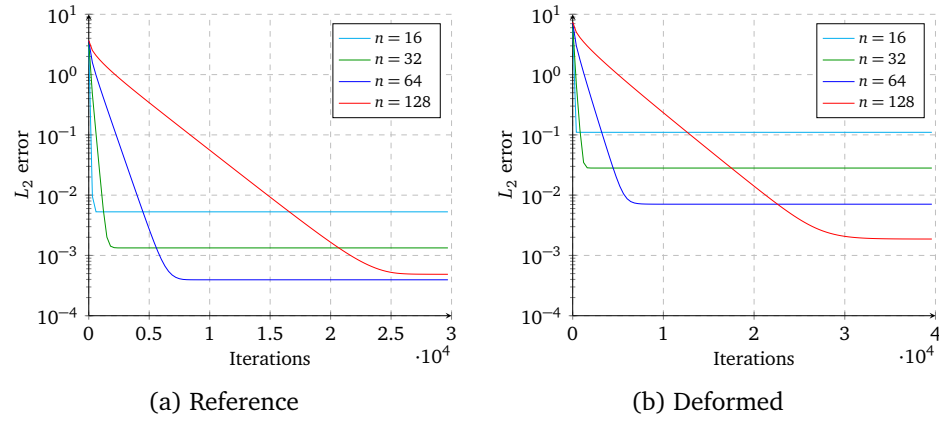


(a) Reference                                             (b) Deformed

Figure 6.7: $L_2$ error norm vs. iterations for the Laplacian Jacobi solver.



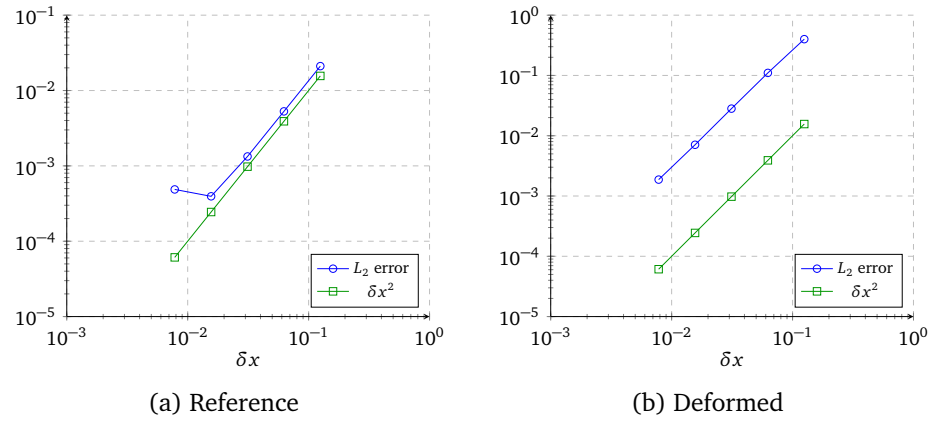(a) Reference                                             (b) Deformed

Figure 6.8: $L_2$ error norm vs. mesh width $\delta x$ for the Laplacian Jacobi solver.

## 6.2 Benchmark Studies

### 6.2.1 Electophysiology

The implementation of the Electrophysiology simulation was validated against the benchmark study of [Niederer et al., 2011a]. The study defines a problem in anisotropic electrophysiology and compares the results of 11 different codes for its solution, each submitted by different research groups.

The problem considers a cuboid of 20mm × 7mm × 3mm of myocardial tissue with an active electrophysiology in accordance with the monodomain equation (5.3.2), repeated here:

$$\frac{\lambda}{1+\lambda} \nabla \cdot (\Sigma \nabla v) = \chi_m \left( C_m \frac{\partial v}{\partial t} + I_m \right). \tag{6.2.1}$$

Instead of the Mitchell Schaeffer model (2.1.1) for the transmembrane potential $I_m$, the study uses the more detailed model of [ten Tusscher et al., 2004], which consists of 11 coupled ODEs and describes all major ion channels as well as intracellular calcium dynamics.

The cuboid is anisotropic, having greater conductivity along its long axis, and receives an initial stimulus in a cubic region of 1.5mm × 1.5mm × 1.5mm at its corner. With all parameters of the model fully specified, the experiment records the activation time in milliseconds throughout the cuboid generated by each of the submitted codes. An example result is shown in Figure 6.9.
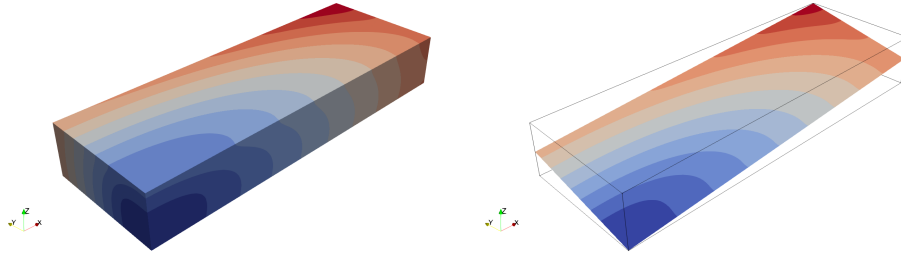


Figure 6.9: Sample results for the eletrophysiology benchmark experiment, showing activation times in *ms* through the cuboid. The stimulus is applied in the lower left corner and the latest activation is in the top right corner.

The results were compared for different spatial $\delta x \in \{0.5\text{mm}, 0.2\text{mm}, 0.1\text{mm}\}$ and temporal $\delta t \in \{0.05\text{ms}, 0.01\text{ms}, 0.005\text{ms}\}$ resolutions by plotting activation time (ms) vs. distance (mm) along the long diagonal of the cuboid from the stimulus to the opposite corner. Figure 6.10 reproduces the results from the paper for the 11 submitted codes.
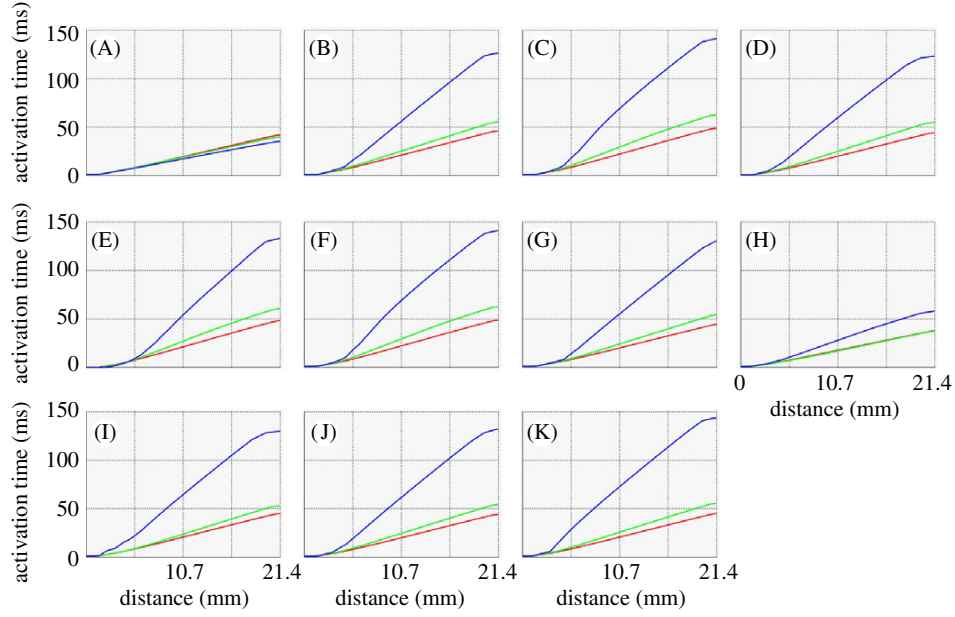
Figure 6.10: Reproduction from [Niederer et al., 2011a], showing activation time (ms) vs. distance (mm) along the diagonal of the cuboid, with $\delta t = 0.005$ms and $\delta x = 0.1$mm (red), $\delta x = 0.2$mm (green) and $\delta x = 0.5$mm (blue).

The discussion in the paper notes that the results are broadly similar, but subject to a variation of around 10% even at high-resolution convergence. Some of the results also exhibit mesh-dependent variation.

The equivalent plot using the algorithm as defined in this work is shown in Figure 6.11. It is worth noting that the y-axis range is zero to 50ms, and so the plot most closely resembles subplot (A) in Figure 6.10. As such the result of our experiment is broadly in line with the result at the finest resolution for the benchmarking results and shows little mesh-dependent variation. The paper gives the high-resolution convergence result at the top corner as between 42.5-43ms, and our result is within that range for the finer resolutions. This is despite the fact that our implementation uses the simpler model of [Mitchell and Schaeffer, 2003].

Furthermore, we consider the results of the simulation at coarser resolutions in both space and time. The results are shown in Figure 6.12(a) where $\delta x = 0.2$mm and 6.12(b) where $\delta x = 0.5$mm. In both cases there is a reduction in overall accuracy for the activation time with respect to mesh width. This can be seen by a higher activation time overall, but the solution is not sensitive to time resolution. This is very useful since it allows for a longer timestep and fewer iterations.
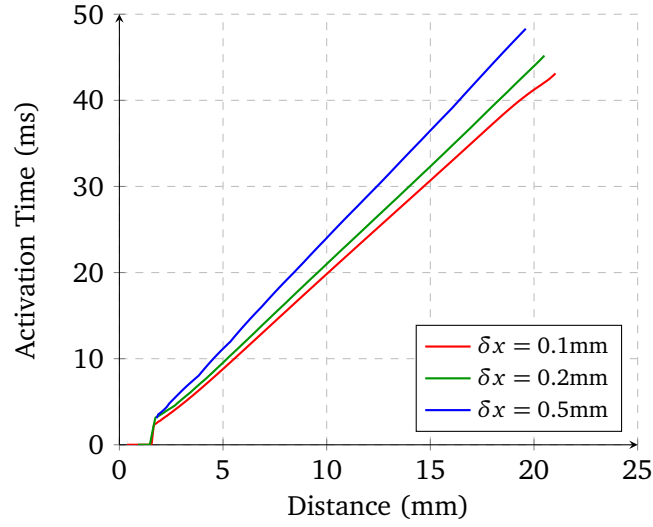
Figure 6.11: Results for the experiment in [Niederer et al., 2011a], using our model, showing activation time (ms) vs. distance (mm) along the diagonal of the cuboid, with $\delta t = 0.005$ms and $\delta x$ as shown.

In conclusion, the CFL condition determines a maximum time step $\delta t$. Since the size, and therefore resolution $\delta x$, of the domain is limited by the memory capacity of the processor, careful consideration can allow for an efficient compromise between spatial and temporal resolution and time to solution.
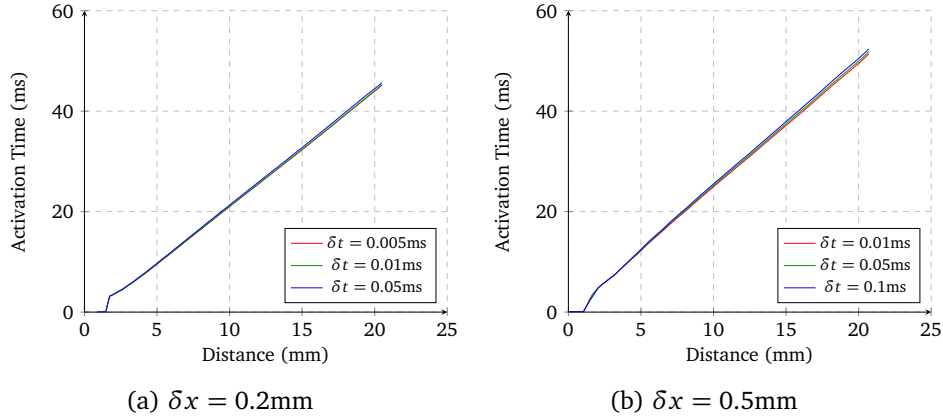


(a) $\delta x = 0.2$mm                        (b) $\delta x = 0.5$mm

Figure 6.12: A comparison or activation time vs. distance for coarser meshes. The overall accuracy is reduced with increasing $\delta x$ but is less sensitive to $\delta t$ and thus requires far fewer iterations.

### 6.2.2  Solid Mechanics

For the verification of the solid mechanics simulation we reproduce some results from the benchmark study of [Land et al., 2015]. The paper sets benchmark problems for the simulation of deformation which occurs in a solid body with varying spatial characteristics under a fluid pressure, either as a passive response or with active stress corresponding to muscle contraction.

The problems were solved by code submitted by 11 different research groups and arrived at consensus solutions using differing approaches to formulation, discretisation and processing. We will reproduce the two experiments corresponding to the response of an idealised ventricle under passive expansion and active contraction.

#### 6.2.2.1  Geometry

The two experiments both make use of the same geometry, given in its reference configuration and shown in Figure 6.13. An idealised ventricle is modelled as a truncated ellipsoid, fixed on its upper plane. The solid part of the geometry contain a vector field corresponding to fibre directions which vary spatially through the tissue in a spiral fashion. The dimensions and properties of the ventricle are given as algebraic expressions which can be easily reproduced in our model via signed distance functions and stored vector fields.
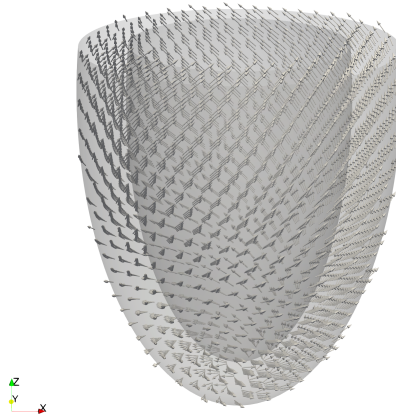


Figure 6.13: Model geometry of [Land et al., 2015], representing an idealised ventricle in its reference configuration showing fibre directions.

A static spherical pressure field was set on the interior of the ventricle, corresponding to the fluid pressure of the blood. On the exterior of the ventricle no boundary conditions were applied, except at the top plane, which was fixed in space.

### 6.2.2.2  Material Model

For the material model, the study makes use of the transversely isotropic constitutive law of [Guccione et al., 1995]. The strain energy function includes the parameterisation of an anisotropic material response determined by the fibre field within the tissue. This model has been widely used both in the determination of model parameters and in the simulation of tissue behaviour, see [Tang et al., 2007],[Hassaballah et al., 2013] amongst others.

We reproduce the results using a different material formulation, the Hyperelastic solid model of [Mooney, 1940] and [Rivlin and Rideal, 1948] for the stress strain relationship, which is passively isotropic and incompressible, and add an anisotropic stress determined by the vector field of fibre directions. We briefly summarise the model as it will be used again in cardiac simulation.

The strain energy density function of a Mooney-Rivlin material is given as:

$$W = c_1(\bar{I}_1 - 3) + c_2(\bar{I}_2 - 3), \tag{6.2.2}$$

where $c_1$ and $c_2$ are empirically determined material constants and $I_1$ and $I_2$ are the first and second invariants of $\bar{\mathbf{B}}$ the unimodular part of the left Cauchy-Green deformation tensor $\mathbf{B}$:

$$\mathbf{B} = \mathbf{F}\mathbf{F}^\top \tag{6.2.3}$$
$$\bar{\mathbf{B}} = \det(\mathbf{B})^{-\frac{1}{3}}\mathbf{B} \tag{6.2.4}$$

The unimodular part of $\mathbf{B}$ has a determinant $\det(\bar{\mathbf{B}}) = 1$. The material model is thus formulated directly in the deformed configuration and separates deformation from change in volume. The invariants are defined as follows and can be calculated quickly from the invariants $I_1, I_2$ of $\mathbf{B}$.

$$\bar{I}_1 = J^{-\frac{2}{3}} I_1 \tag{6.2.5}$$
$$\bar{I}_2 = J^{-\frac{4}{3}} I_2, \tag{6.2.6}$$

where the standard tensor invariants are defined as follows:

$$I_1 = \text{tr}(\mathbf{B}) \tag{6.2.7}$$
$$I_2 = \frac{1}{2}\left[\text{tr}(\mathbf{B})^2 - \text{tr}(\mathbf{B}^2)\right], \tag{6.2.8}$$

and $J = \det(\mathbf{F})$ is the determinant of the deformation gradient as usual.

For an incompressible Mooney-Rivlin material $J = 1$ and thus $\bar{\mathbf{B}} = \mathbf{B}$. The Cauchy stress $\boldsymbol{\sigma}$ can therefore be expressed as:

$$\boldsymbol{\sigma} = -p^*\mathbb{I} + 2(c_1\mathbf{B} - 2c_2\mathbf{B}^{-1}) \tag{6.2.9}$$

Where $p^*$ represents a corrective pressure given by:

$$p^* = \frac{2}{3}(c_1 I_1 - c_2 I_2) \qquad (6.2.10)$$

As a result the highly nonlinear behaviour of an incompressible solid can be modelled using a stable, explicit iteration. This the reason for the choice of the Mooney-Rivlin material and a key part of the design of the cardiac simulation.

For the anisotropic contraction a synthetic deformation is introduced, equivalent to an extension in the longditudinal fibre direction $\mathbf{f} \otimes \mathbf{f}$, as per (2.1.24). The extension is added to the existing deformation gradient $\mathbf{F}$ and flows into the calculation of stress, resulting in a contractile response that is equivalent to muscle contraction, but without affecting the incompressible response of the material model. Again this is an important part of the design, in maintaining the integrity and stability of the mechanical simulation.

### 6.2.2.3   Passive Expansion

In the first experiment a pressure of 10kPa was applied to interior surface of the solid in its reference configuration and the ventricle is allowed to inflate. The contributors to the paper solve a static problem but we run the dynamic solution until it reaches equilibrium, as shown in Figure 6.14.



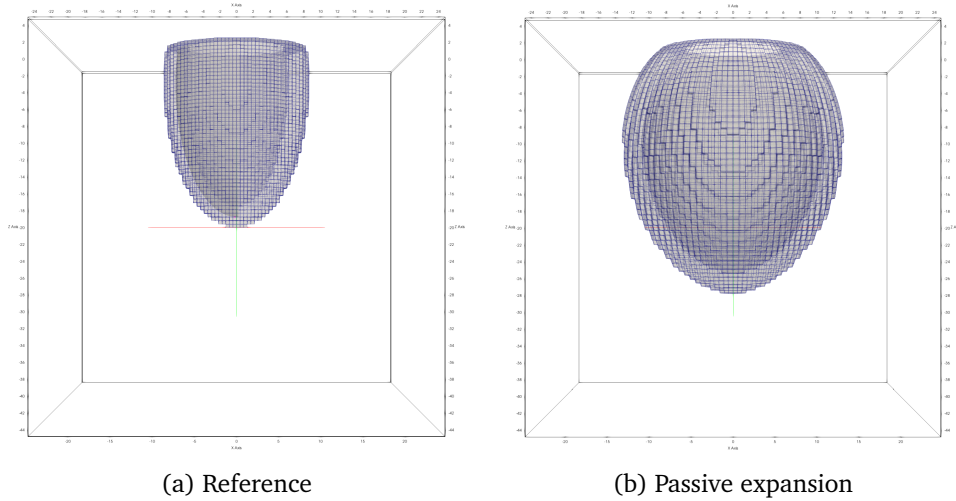|          (a) Reference          |       (b) Passive expansion       |

Figure 6.14: The reference configuration (a) and passive expansion (b) of an idealised ventricle.

The simulation was run on a three-dimensional regular structured grid of $100^3$ cells. The mesh width $\delta x = 0.5$mm, and time step $\delta t = 0.005$ms. The computation output a file completely describing the state of the model every 500

iterations, producing 100 files in total. The simulation time was thus $100 \times 500 \times 0.005 = 250$ms, which was sufficient for the solid to reach a stationary equilibrium. The GPU processor used was an NVIDIA A100 PCI with 40GB memory and the processing time was around 160 seconds.

The paper reports the final position of the midline of the solid. The experiment was repeated and the Mooney-Rivlin parameters updated until the results were in agreement with the consensus solution, as shown in Figure 6.15. The values of $c_1$ and $c_2$ were thus both set to 0.5. These values were retained for subsequent experiments.
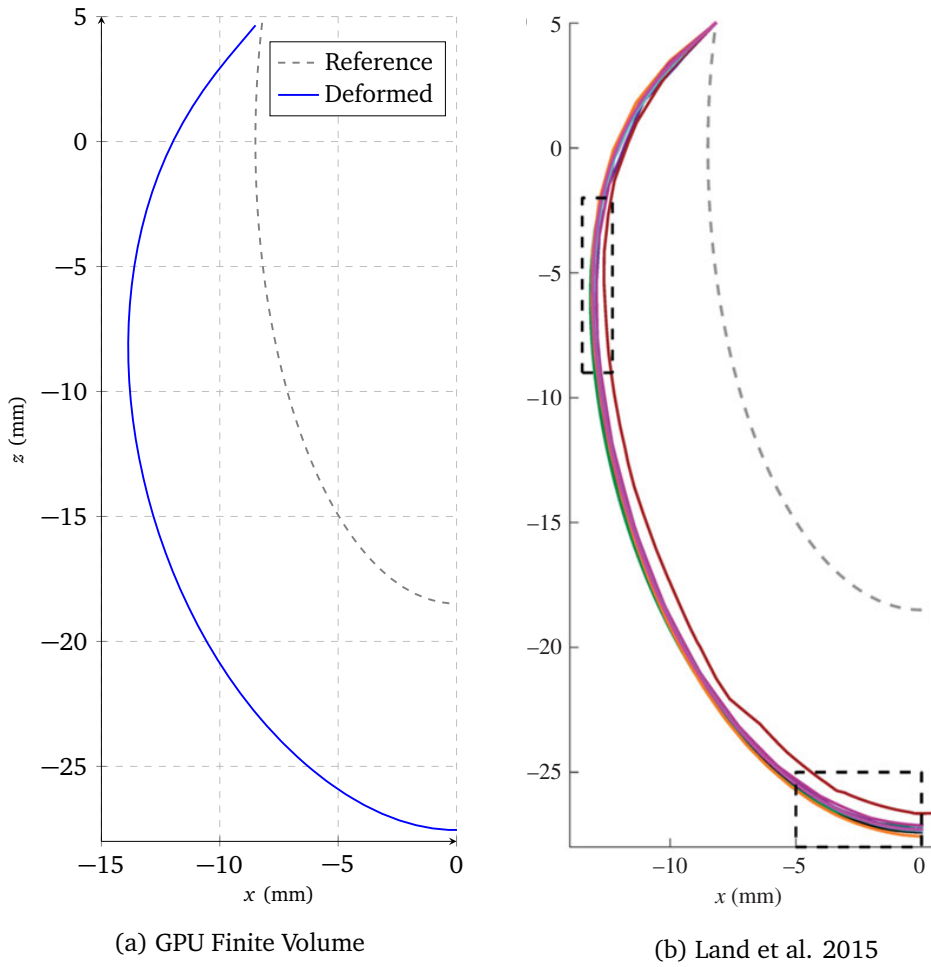


(a) GPU Finite Volume

(b) Land et al. 2015

Figure 6.15: The reference configuration and passive deformation of an idealised ventricle with (a) generated by this work and (b) an extract from [Land et al., 2015], showing comparable results.

### 6.2.2.4   Active Contraction

In the second experiment a pressure of 15KPa was applied to interior surface of the solid in its reference configuration as well as a contractile Piola Kirkhoff stress of 60kPa orientated along the fibre directions in the solid. Again, the contributors solve a static problem and we run a dynamic solution to equilibrium, as shown in Figure 6.16.



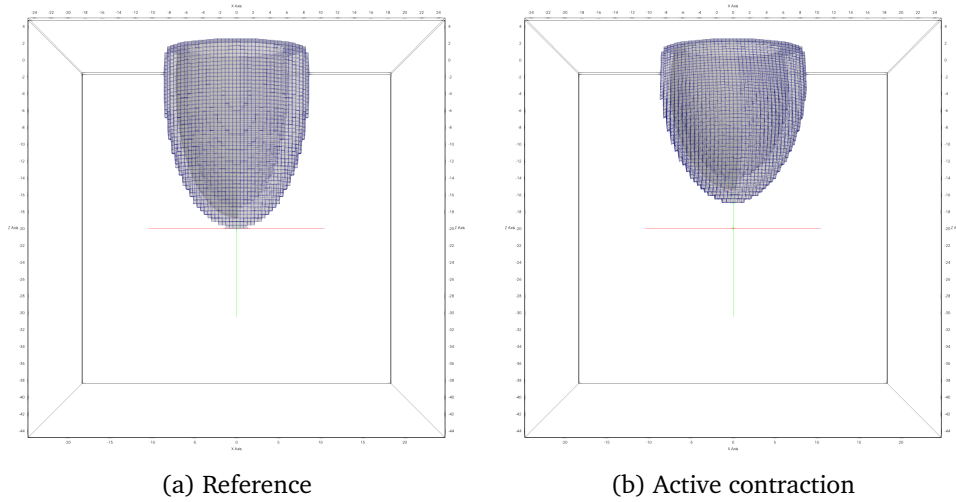(a) Reference                                            (b) Active contraction

Figure 6.16: The reference configuration (a) and active deformation (b) of an idealised ventricle under muscle contraction.

The simulation was run on the same three-dimensional regular structured grid of $100^3$ cells with mesh width $\delta x = 0.5$mm, but the larger forces required a shorter time step $\delta t = 0.001$ms. The computation output a file every 1000 iterations, producing 100 files in total for a simulation time of $100 \times 1000 \times 0.01 = 100$ms, again reaching a static equilibrium. The same NVIDIA A100 processor completed the calculation in 160 seconds. This illustrates an important point, that despite the model making twice as many iterations (50,000 in the first case and 100,000 in the second) there is no difference in the computation time. This is because the time to solution is completely dominated by traffic across the PCI bus and IO on the CPU. This is in line with the observations in Chapter 3.

The experiment was repeated until the synthetic expansion parameter for muscle contraction was identified as 1.0, and it is retained for future experiments. The results for the final position of the midpoint are shown in Figure 6.17.

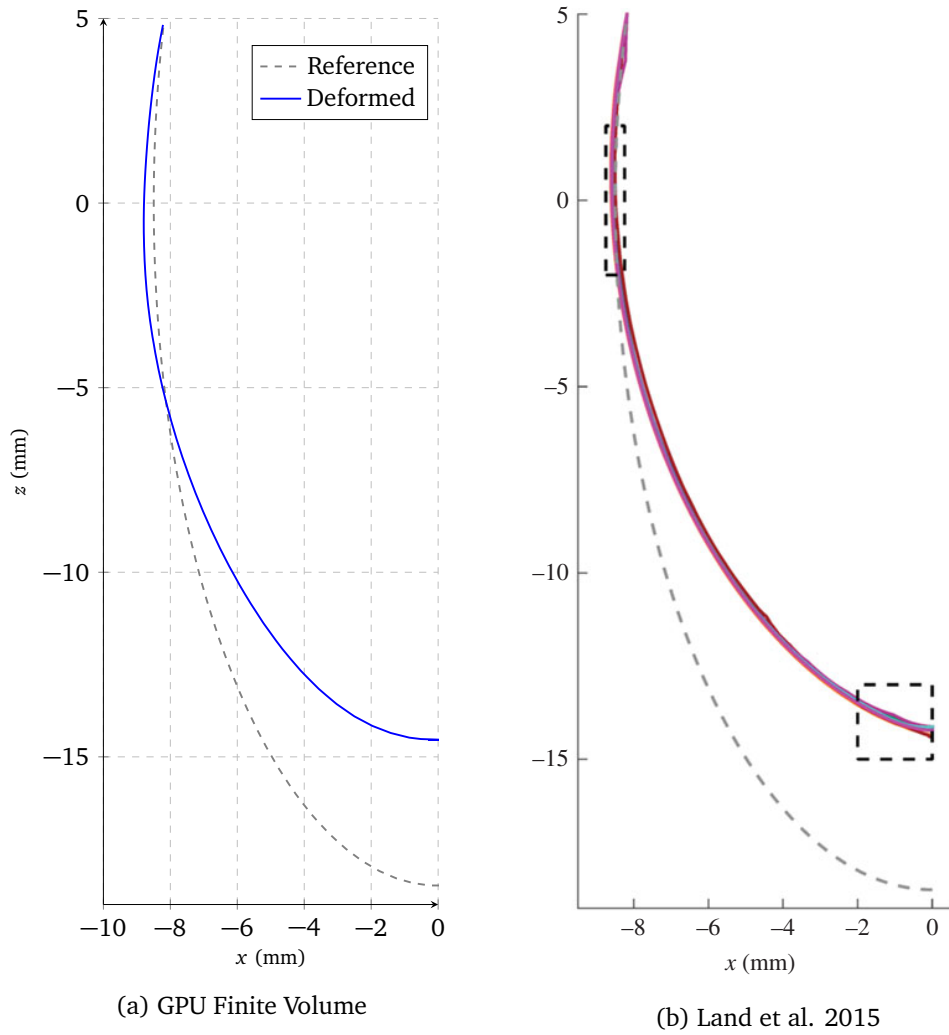(a) GPU Finite Volume

(b) Land et al. 2015

Figure 6.17: The reference configuration and active deformation of an idealised ventricle under muscle contraction, with (a) generated from this work and (b) an extract from [Land et al., 2015], showing comparable results, $\delta x = 0.5$mm.

# Chapter 7

# Cardiac Simulation

## 7.1 Overview

In this section we combine all of the elements of the work into a single whole heart simulation. The objective is to provide a complete set of descriptive data, including position $\mathbf{x}$, solid $\mathbf{v}_s$ and fluid velocity $\mathbf{v}_f$, membrane depolarisation $u$ and fluid pressure $p$ over the course of a single heartbeat. The simulation time is thus one second ($t = 1000\text{ms}$) with 100 data files, each corresponding to a frame of video output at 10ms intervals. The results are generated at at spatial resolutions of $\delta x = 1.0\text{mm}$ and $\delta t \in [0.05, 0.001]\text{ms}$ respectively. This allows the algorithm to make many iterations between each use of the PCI Bus, hiding latency as discussed in Chapter 3.

The resulting data can be sampled over subsets of the domain and post-processed to generate synthetic analogues to physiological recordings. The transmembrane voltage is sampled at three positions and used to simulate the output of a three-lead Electrocardiogram (ECG). Position data and pressure within a ventricle will be used to generate a synthetic Pressure-Volume (PV) loop.

It is important to note that the code is a first prototype and this simulation serves as a proof-of-concept for the algorithm itself, rather than presenting the experiment and its results as a state-of-the-art simulation. Validation and the addition of richer models is left as future work.

The following sections will outline the construction of the cardiac geometry using the signed distance functions of Section 4.3, and the adaptation of the solvers and algorithm of Chapters 4 and 5 respectively. Finally, we display and discuss some of results.
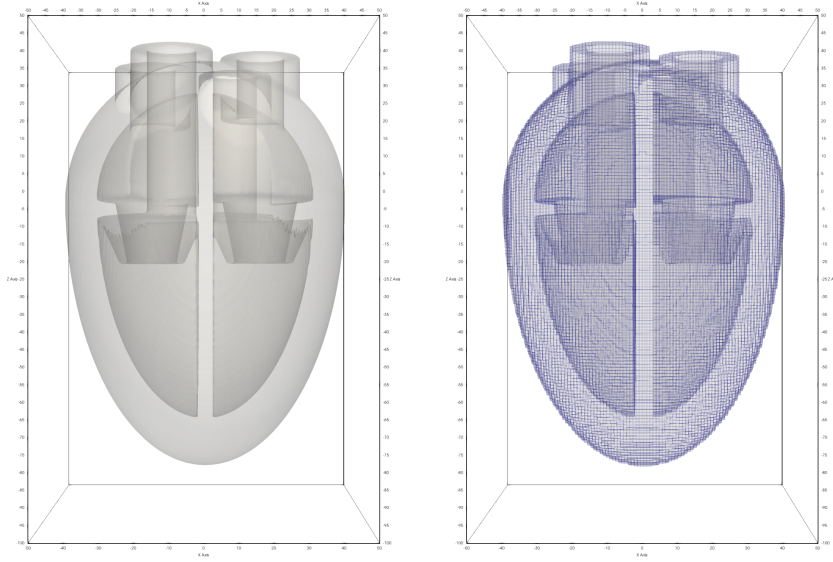
## 7.2   Geometry

### 7.2.1   Myocardium



Figure 7.1: The schematic whole heart in signed distance function and voxel representation.

The geometry of a schematic heart is shown in Figure 7.1. It is constructed around a central origin and is comprised of several elements defined by their own SDFs, which are combined according to the set algebra of Section 4.3. The myocardium is a combination of four SDFs, two half-spheres above the origin and two ellipsoids below it, as shown in Figure 7.2a. The intra-ventricular septum is and a horizontal septum are defined as elliptic discs as in Figure 7.2b. Hollow vertical cylinders representing the veins and aorta entering the atria and leaving the ventricles respectively are shown in Figure 7.2c. Finally the mitral and tricuspid valves are represented as hollow elliptic cones which penetrate the horizontal septum as shown in Figure 7.2d. The geometry does not include any other blood vessels or valves at the entrance of the atria or ventricles, and as such their action is significantly absent from the results of the simulation.

### 7.2.2   Fibre Directions

Fibres are present in the the tissue of the myocardium (Figure 7.2a) which determine both the conductivity of the myocardial tissue as well as the axis along which it contracts. The fibre field is stored a set of vectors, shown in Figure 7.3. The conductivity is determined via (2.1.24), where $\mathbf{f} \in \mathbb{R}^3$ is the stored
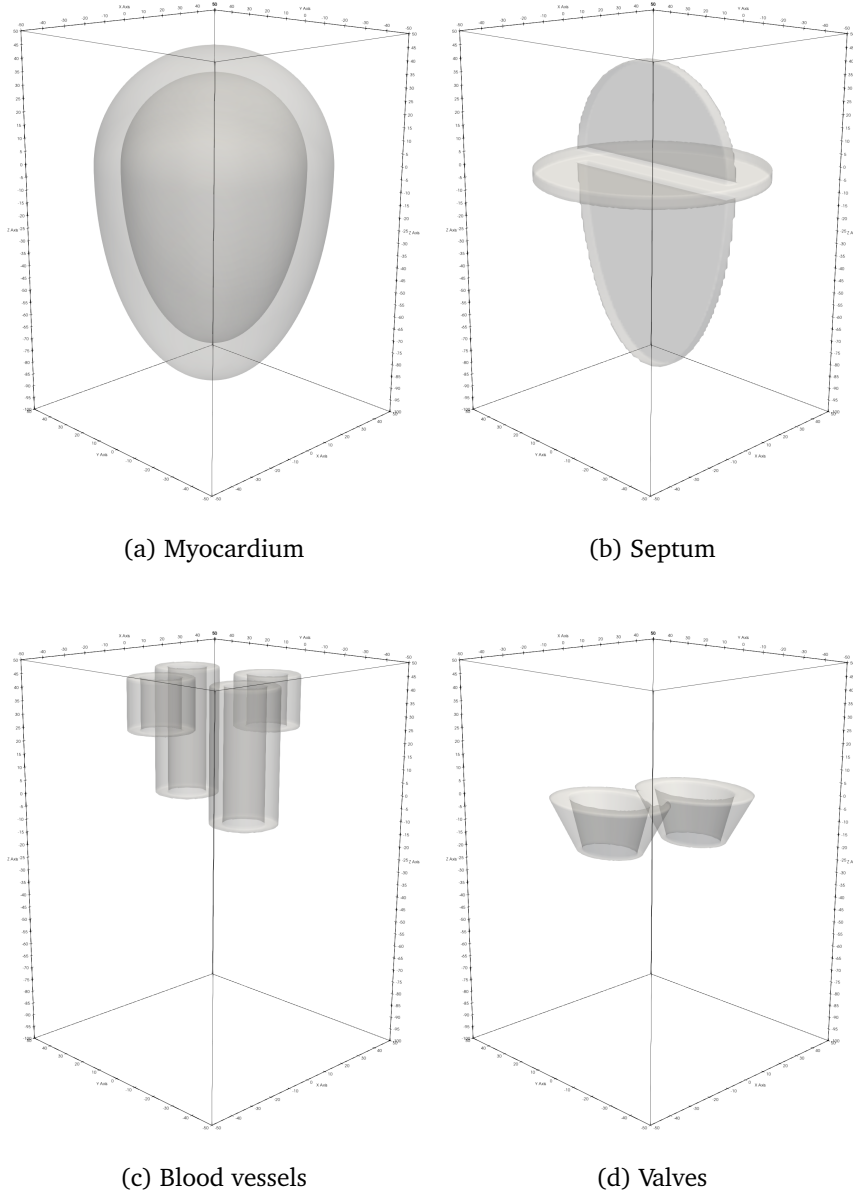
(a) Myocardium

(b) Septum

(c) Blood vessels

(d) Valves

Figure 7.2: Geometric elements comprising the whole heart geometry

fibre direction. The values of londitudinal $\sigma_L$ and transverse $\sigma_T$ conductivity are given as follows:

$$\sigma_L = 0.20 \text{ mS mm}^{-1}$$
$$\sigma_T = 0.05 \text{ mS mm}^{-1}$$

Where S indicates Siemens, the SI unit of conductivity, given as $1S = \mu A \, mV^{-1}$ $mm^{-1}$. Where no fibres are present the conductivity in all directions is set equal to $\sigma_T$. The form of the fibre direction is constructed as a unit vector lying initially in the horizontal (xy) plane, initially orthogonal to the position vector of the point in the myocardium at which the fibre is set. The vector is then rotated by $0.4\pi$ in the axis of its position vector to give the generally accepted helical form for the fibres [Land et al., 2015] [Potse et al., 2006] [Tang et al., 2007], among others. It is possible to set the angle of fibre rotation with a transmural parameter, but this did not affect the results for either electrophysiology or mechanics and it is not used at this stage.
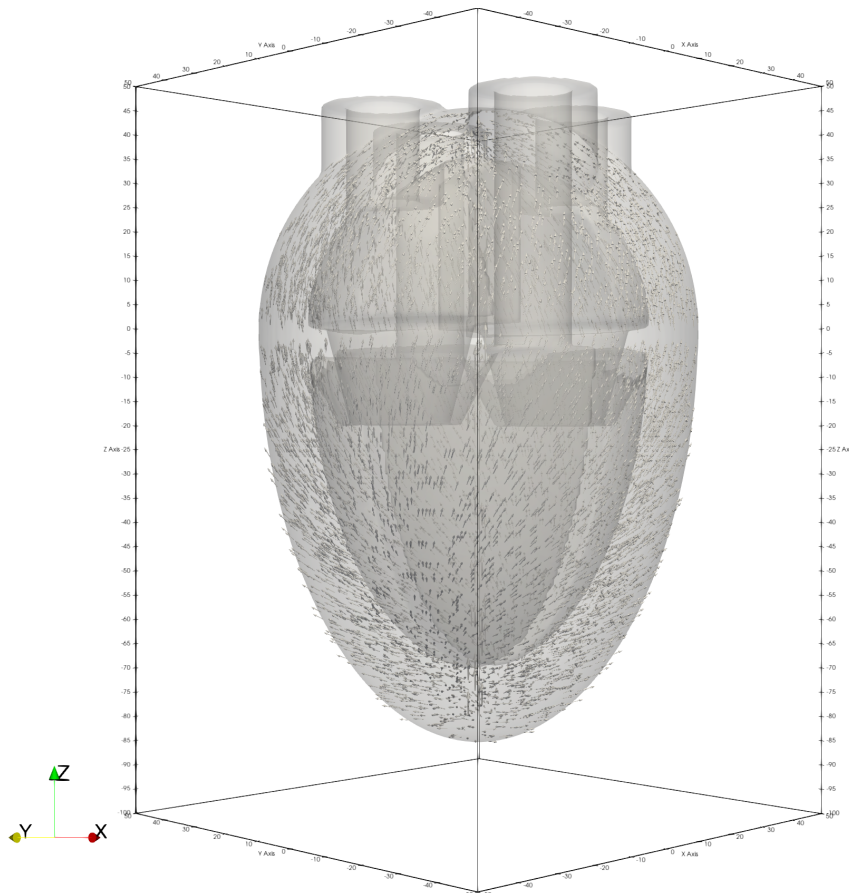


Figure 7.3: Cardiac geometry showing the vector field of fibre directions.

In addition to the fibres of the myocardium there is a fibre running vertically through the centre of the heart which represents the higher conductivity of the Purkinje network. The fibre has twice the longditudinal conductivity of the regular fibres. It works in cunjuction with the horizontal septum (Figure 7.2b) which has its conductivity set to zero. Since the fast fibre is the only point at which electrical stimulation can pass from the atria to ventricle the movement of

depolarisation through the centre of the heat reproduces the action of the atrio-ventricular node, separating the contraction of the upper and lower sections of the heart into two distinct events.

## 7.3 Algorithm

### 7.3.1 Electrophysiology

The heart receives a stimulus of 20mV at a position near its apex located in the left atrium, equivalent to the SA node. The Mitchell Schaeffer and Monodomain models are used to model calculate the electrophysiology, as defined earlier in this work and implemented in the reproduction of the benchmark paper of [Niederer et al., 2011a] in Chapter 6. The diffusive behaviour of the Monodomain model includes a Laplacian operator which can be iterated in time either with an Implicit Euler integration, using a Jacobi or Richardson iteration. The time step $\delta t = 0.001$ms is sufficiently short to allow for an Explicit Euler integration at a spatial resolution $\delta x = 1.0$mm. In this case the implicit and explicit results are equivalent. If an ECG is required the elliptic problem on the second line of (2.1.19) is solved via a Jacobi or Richardson iteration.

### 7.3.2 Solid Mechanics

For the solid mechanics the incompressible Mooney Rivlin model is used with the same parameters as in the reproduction of the benchmark study of [Land et al., 2015] in Chapter 6. As described in Section 6.2.2.2, a synthetic extension of the tissue is coupled to the transmembrane voltage. This induces a contractile stress in the tissue which does not interfere with the incompressible dynamics of the material model. As a result the contraction is stable under explicit iteration.

### 7.3.3 Fluid Dynamics

The behaviour of the fluid is as described in 5. For the solution of the Poisson pressure equation resulting from the Helmholz projection of Section 5.4.1, either a Jacobi or Richardson iteration is used. The current pressure field is used as the initial guess for the iterative solve and a small number of iterations are performed such that the solution is incomplete. This is equivalent to the semi-implicit solution given in [Formaggia, 2009] but takes advantage of the Finite Volume discretisation. Since the solution is incomplete, the corrected fluid velocity is not entirely divergence-free and the gradient of the pressure field is

under-estimated. This in turn enters the fluid structure interaction as Cauchy stress as shown in Figure 7.4.
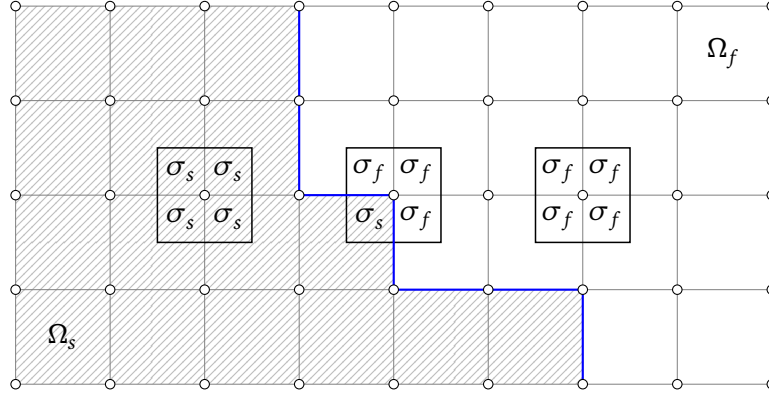


Figure 7.4: The contribution of solid and fluid Cauchy stress tensors to the velocity of solid, fluid and surface vertices.

Due to the conservative nature of FVM this error is retained in the solution and remains at the next solution of the Poisson equation. As such the pressure correction is applied over more than one time step. The resulting error in the solution is equivalent to a fluid that is both compressible and damped. This is not entirely physically incorrect and contributes to the stability of the fluid-structure interaction, which otherwise is the main source of error in the model.

## 7.4 Results

### 7.4.1 Electrophysiology

The simulation of the electrophysiology alone can be performed with a slightly longer time step of $\delta t = 0.05$ms. The results are shown in Figure 7.5 and reproduce an approximation on the cardiac action potential. The transport of the depolarisation through the fast fibre on the central axis of the geometry can be seen. The computational cost is quite low, and the computation can be performed quickly on a desktop computer. In this case we used an iMac with its native Radeon Pro 580 8GB GPU and the simulation with 1 million degrees of freedom took 66 seconds. Many repeated simulations are therefore possible, and as such the model can be considered as the objective function of an optimisation problem. This allows for richer iterative experiments such as parameter estimation, neural network training and the testing of surgical or drug therapies. A more complex ionic model can also be substituted into the calculations.
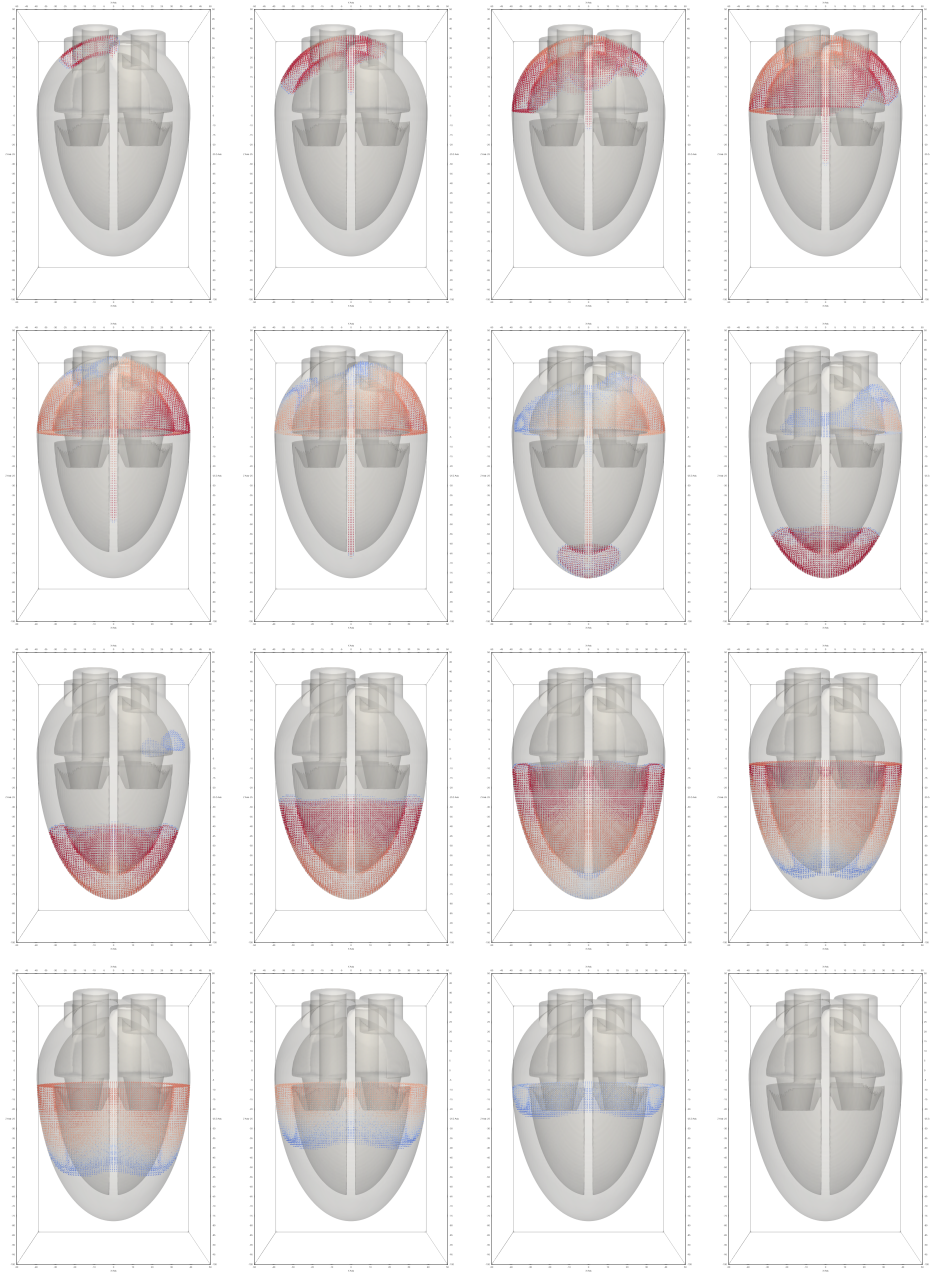
Figure 7.5: The evolution of the cardiac action potential at intervals of 50ms ($\delta x = 1.0$mm).

### 7.4.1.1   ECG

In order to reconstruct the output of a synthetic three-lead Electrocardiogram (ECG), the transmembrane voltage was allowed to diffuse out of the heart and
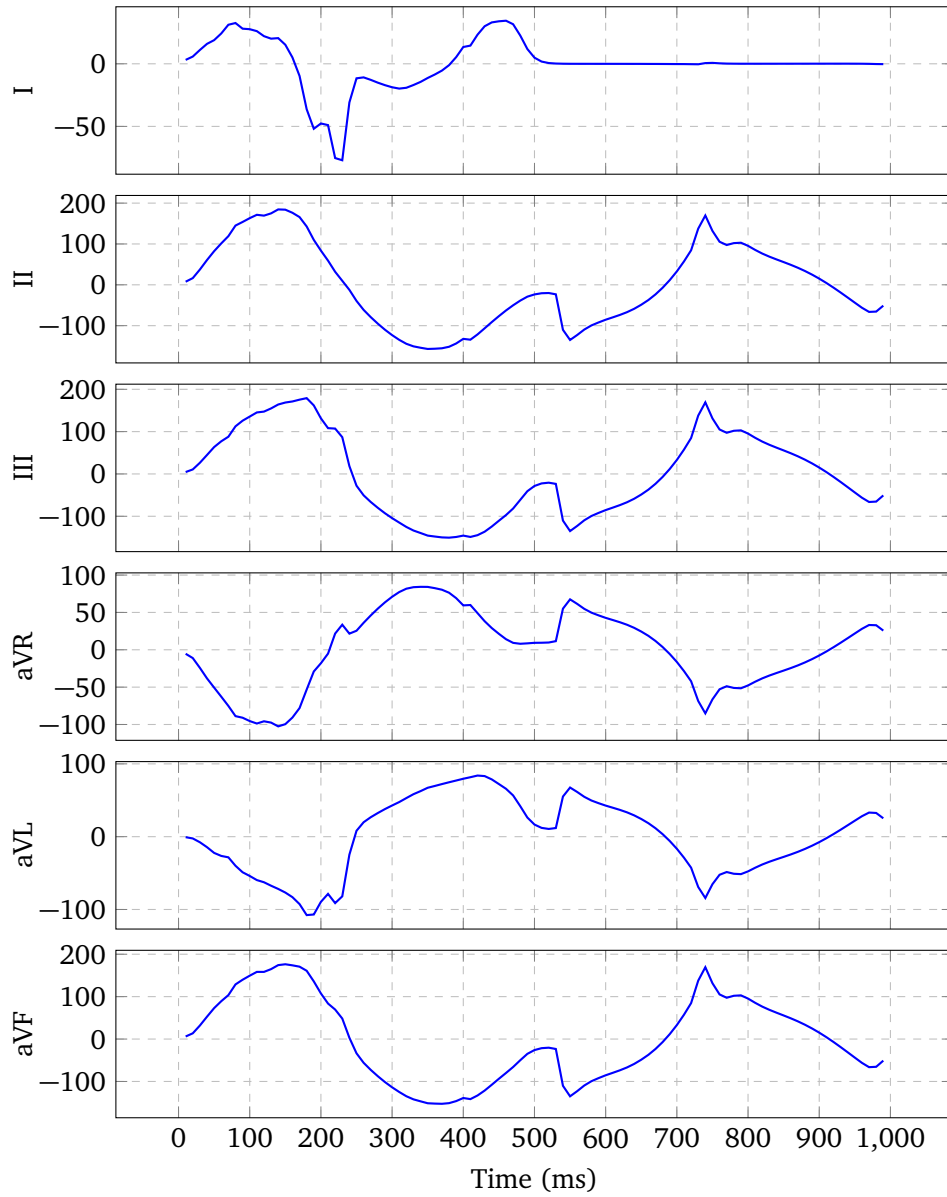
Figure 7.6: The six standard outputs of a 3-lead Electrocardiogram.

into the surrounding tissue. The conductivity of the tissue was set to a high value and thus mimicked the movement of charge through the torso.

The values at diametrically opposite corners of the domain were extracted from the data to represent the left arm (LA), right arm (RA) and left leg (LL) electrodes. The standard calculations were applied to generate the six outputs, as

per the following formulae:

$$
\begin{aligned}
I &= LA - RA \\
II &= LL - RA \\
III &= LL - LA \\
aVR &= RA - \frac{1}{2}(LA + LL) \\
aVL &= LA - \frac{1}{2}(RA + LL) \\
aVF &= LL - \frac{1}{2}(RA + LA)
\end{aligned}
$$

The results are shown in Figure 7.6. They do not appear to contain much detail and are certainly open to interpretation. They do however contain peaks corresponding to the P and T waves, and the QRS complex is present but not pronounced. The output depends upon both the choice of conductivity through the torso and the position of the recording points and is greatly affected by the small domain on which the problem is solved. These parameters can be chosen by experimenters, and since the ECG can be generated within around 3 minutes per heartbeat, there is much room for refinement.

### 7.4.1.2   Simulation Times

The following tables summarise electrophysiology simulation times for various resolutions and processors. In each case the total simulation time was 1000ms (1 second) and the software output a set of files containing a complete description of the heart at 10ms intervals (100 sets of files in total). The simulation is stable at the longer time step of 0.05ms, but the runs with shorter time step 0.01ms are shown for comparison with the whole heart timing data. File input/output (IO) and PCI bus operations dominate the calculation time for smaller simulations. When the ECG is required the iterative solution of the elliptic problem increases the total time depending on the resolution required. For example on the iMac the solution time for the heart at a resolution of 100 increases to around 3.5 minutes.

| Dimensions | Elements | $\delta x$(mm) | $\delta t$(ms) | Time(mm:ss) |
|---|---|---|---|---|
| 50×50×75 | 187,500 | 2.0 | 0.01 | 00:19 |
| 100×100×150 | 1,500,000 | 1.0 | 0.01 | 01:58 |
| 200×200×300 | 12,000,000 | 0.5 | 0.01 | 21:32 |
| 50×50×75 | 187,500 | 2.0 | 0.05 | 00:07 |
| 100×100×150 | 1,500,000 | 1.0 | 0.05 | 01:54 |
| 200×200×300 | 12,000,000 | 0.5 | 0.05 | 12:00 |

Table 7.1: AMD Radeon Pro 580 (2017 iMac).

| Dimensions | Elements | $\delta x$(mm) | $\delta t$(ms) | Time(mm:ss) |
|---|---|---|---|---|
| 50×50×75 | 187,500 | 2.0 | 0.01 | 00:10 |
| 100×100×150 | 1,500,000 | 1.0 | 0.01 | 00:52 |
| 200×200×300 | 12,000,000 | 0.5 | 0.01 | 05:15 |
| 50×50×75 | 187,500 | 2.0 | 0.05 | 00:06 |
| 100×100×150 | 1,500,000 | 1.0 | 0.05 | 00:30 |
| 200×200×300 | 12,000,000 | 0.5 | 0.05 | 03:39 |

Table 7.2: NVIDIA A100 PCIE 40GB (USI ICS Cluster).

When the output is ECG only there is no need to write files, only to copy the electrophysiology buffer to the CPU for post-processing. These timings are shown below:

| Dimensions  | Elements   | $\delta x$(mm) | $\delta t$(ms) | Time(mm:ss) |
|-------------|-----------:|:--------------:|:--------------:|:-----------:|
| 50×50×75    | 187,500    | 2.0            | 0.01           | 00:16       |
| 100×100×150 | 1,500,000  | 1.0            | 0.01           | 01:30       |
| 200×200×300 | 12,000,000 | 0.5            | 0.01           | 16:22       |
| 50×50×75    | 187,500    | 2.0            | 0.05           | 00:04       |
| 100×100×150 | 1,500,000  | 1.0            | 0.05           | 00:20       |
| 200×200×300 | 12,000,000 | 0.5            | 0.05           | 03:19       |

Table 7.3: ECG only: AMD Radeon Pro 580 (2017 iMac).

| Dimensions  | Elements   | $\delta x$(mm) | $\delta t$(ms) | Time(mm:ss) |
|-------------|-----------:|:--------------:|:--------------:|:-----------:|
| 50×50×75    | 187,500    | 2.0            | 0.01           | 00:04       |
| 100×100×150 | 1,500,000  | 1.0            | 0.01           | 00:12       |
| 200×200×300 | 12,000,000 | 0.5            | 0.01           | 01:33       |
| 50×50×75    | 187,500    | 2.0            | 0.05           | <00:01      |
| 100×100×150 | 1,500,000  | 1.0            | 0.05           | 00:04       |
| 200×200×300 | 12,000,000 | 0.5            | 0.05           | 00:22       |

Table 7.4: ECG only: NVIDIA A100 PCIE 40GB (USI ICS Cluster).

### 7.4.2 Whole Heart

The whole heart simulation includes a fully-coupled representation of electro-physiology, solid and fluid dynamics. The algorithm was executed on a single NVIDIA A100 PCIE 40GB GPU at resolution $\delta x = 1.0$mm and $\delta t = 0.001$ms. The simulation time was 6081 seconds or approximately 100 minutes. The results are shown in Figure 7.7, with streamlines indicating the fluid velocity field.

The output files contain a complete description of position, velocity, membrane voltage and pressure throughout the rectangular domain. They capture the basic elements of the heartbeat, specifically a two-stage contraction of atria, followed by ventricles, and the resulting unidirectional flow of blood through the heart. Due to the absence of valve functionality there is a considerable reverse flow of blood from ventricle to atrium during ventricular systole.
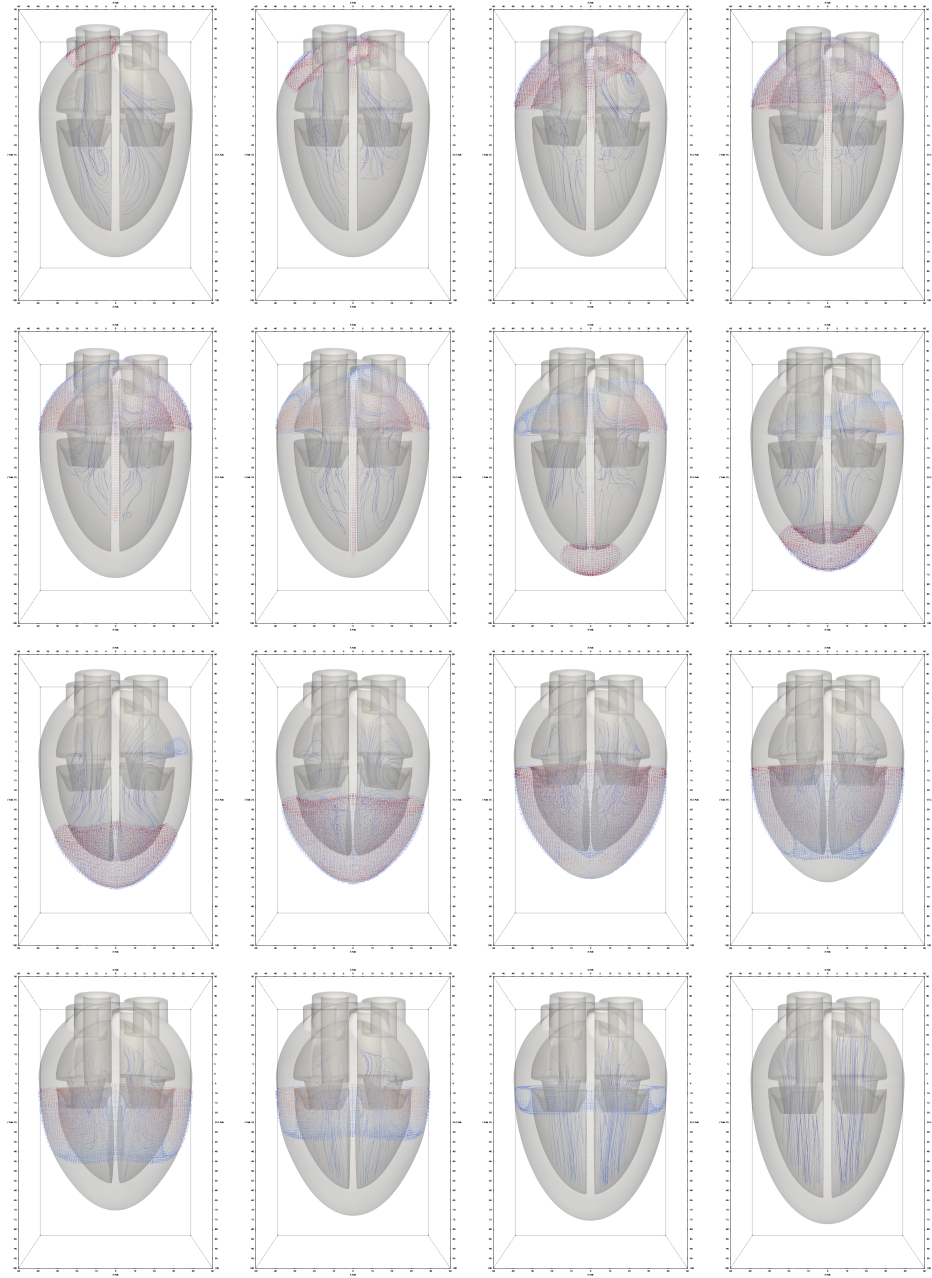
Figure 7.7: The evolution of the whole heart simulation at intervals of 50ms ($\delta x = 1.0$mm).

### 7.4.2.1   PV Loop

For the calculation of Pressure and Volume in the ventricle, a region of voxels were selected and tagged within the left ventricle. As the software generated

data the voxels were counted and the pressure and volumes extracted. The
volume was then divided by the number of voxels to give average values.
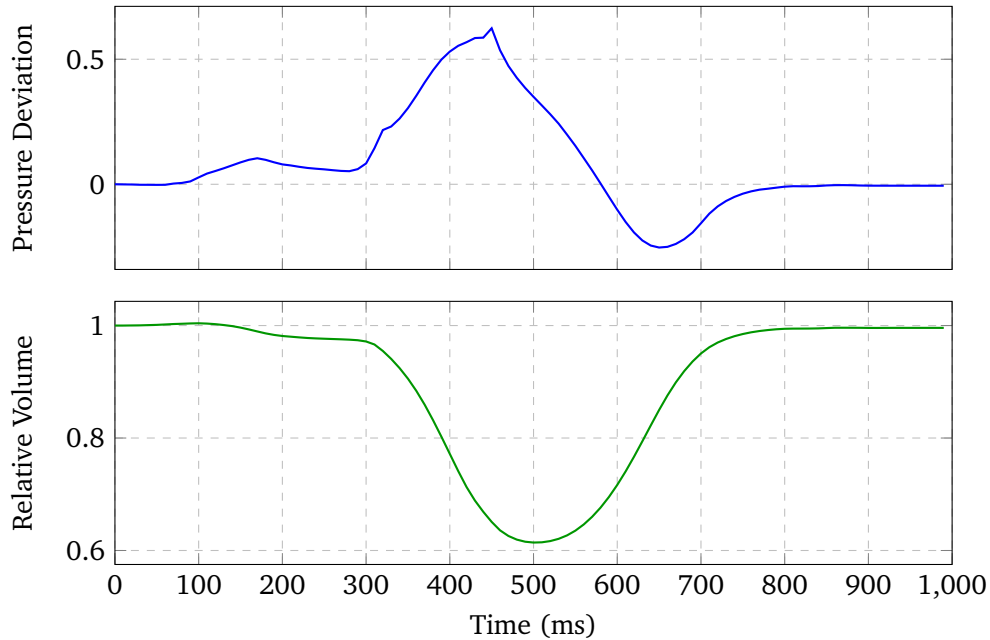


Figure 7.8: Deviation in Average Pressure and Relative Volume over a 1000ms
simulation of the heartbeat.

The results are shown plotted against time in Figure 7.8 over 100 frames, each
representing 10ms of simulation. They were then combined to plot a PV loop as
shown in Figure 7.9. From the plot it can be seen that the PV loop displays the
basic characteristics of the behaviour of the ventricle. An increase in pressure
followed by an increase in volume corresponds to the filling of the ventricle
from the atrium. Then the decrease in volume coupled to an increase in pres-
sure associated with ventricular contraction and expulsion. The results are by
no means physiologically accurate. This is partly due to the absence of valve
function from the simulation, and also due to the damped and compressible
fluid behaviour which results form the incomplete solution of the Poisson pres-
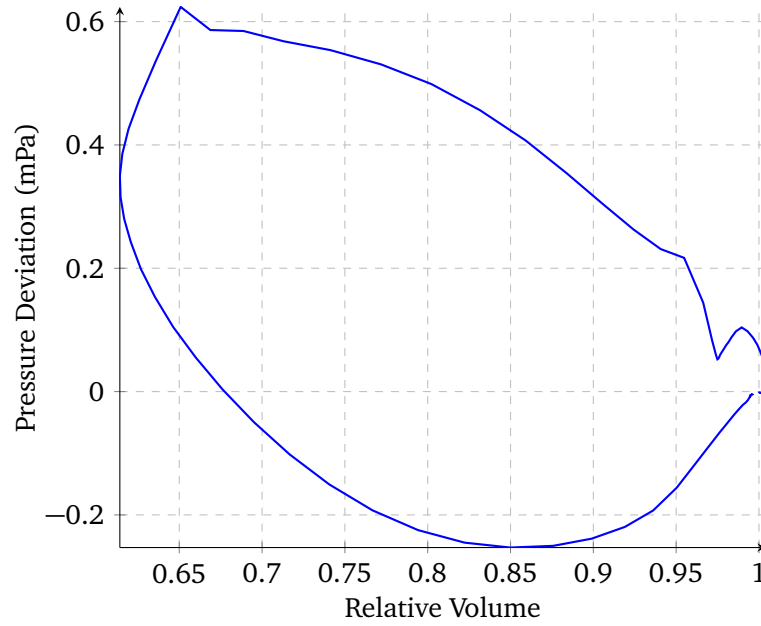sure equation.

Figure 7.9: Pressure-Volume Loop derived from the data in Figure 7.8.

Again, the simulation was computed on a single NVIDIA A100 PCIE 40GB GPU at resolution $\delta x = 1.0$mm and $\delta t = 0.001$ms. The simulation time was 6081 seconds or approximately 100 minutes.

### 7.4.2.2 Simulation Times

The following tables summarise whole heart simulation times for various resolutions and processors. In each case the total simulation time was 1000ms (1 second) and the software output a set of files containing a complete description of the heart at 10ms intervals (100 sets of files in total). The column labelled $k$ is the number of poisson iterations per time step $\delta t$.

We also performed a simulation on a single node Piz Daint at the Swiss national supercomputing centre. The machine has a faster file system but less powerful NVIDIA Tesla P100 16GB GPUs. The timings were slightly slower than the NVIDIA A100 of the USI ICS cluster and so the results are not shown.

| Dimensions | Elements | $\delta x$(mm) | $\delta t$(ms) | Poisson $k$ | Time(mm:ss) |
|---|---|---|---|---|---|
| 50×50×75 | 187,500 | 2.0 | 0.01 | 10 | 11:21 |
| 100×100×150 | 1,500,000 | 1.0 | 0.01 | 10 | 68:02 |

Table 7.5: AMD Radeon Pro 580 (2017 iMac).

| Dimensions | Elements | $\delta x$(mm) | $\delta t$(ms) | Poisson $k$ | Time(mm:ss) |
|---|---|---|---|---|---|
| 50×50×75 | 187,500 | 2.0 | 0.010 | 10 | 01:16 |
| 100×100×150 | 1,500,000 | 1.0 | 0.010 | 10 | 09:57 |

Table 7.6: NVIDIA A100 PCIE 40GB (USI ICS Cluster).

# Part IV

# Conclusion

# Chapter 8

# Review & Outlook

## 8.1 Review

We have given details of a mathematical model and computational algorithm that can provide a complete description of a single heartbeat, along with physiological metrics such as and ECG and PV loop. The computational results are available within seconds for a small model of electrophysiology up to around one hour for a complete model, using only a single GPU processor.

The design of the mathematical model and its discretisation is made with computational constraints in mind. It is based upon classically valid mathematical approaches and principles, present in textbooks rather than journal papers. There is a deliberate avoidance of complicated mathematics which gives a benefit in simplicity, both to the user and the processor. The algorithm is both mesh and matrix-free, relying on assembly per vertex or element as required. As a result the design is readily configurable and extensible to include more complex geometry or mathematical models in future.

The code is extremely short, and has no software or hardware dependence. There is no make file, and the compile line:

$$\texttt{gcc mesh.c ocl.c main.c -lOpenCL -lm} \qquad (8.1.1)$$

references only the C source code, OpenCL and the native C mathematics library. There are functions for data retrieval, calculus and linear algebra, as well as the signed distance functions of the geometry. For each of the operators there is a single computational kernel, such as for the update of membrane potential, diffusion of electrical charge, the calculation of divergence of stress etc.

The simplicity of the code contributes to speed of execution and low consumption of computational and human resources, and ultimately time and money. As

a result it can be used iteratively by a single competent user, for repeated experiments, parameter fitting or the training of Artificial Neural Networks (ANN). It thus has application to both academic research and clinical use. The configurable and extensible nature allows it to be adapted to different pathologies, interventions and ultimately individual subjects.

The result of this work is therefore a proof of concept of a novel computational approach to cardiac simulation which is practical, not theoretical.

## 8.2   Assesment

The work is at a very early stage. The aim was to arrive at a complete working model, accepting compromises along the way, without losing sight the ultimate objective. As a result there are many sources of error, but the process of completing the model allows us to consider these errors and to make judgements about the relative costs and benefits associated with their correction.

Again it is important to note that we are not presenting the cardiac simulation itself as state-of-the-art, but showing that it is possible to make a cardiac simulation with this novel computational and algorithmic approach.

The following sections give a brief discussions of the strengths and weaknesses of the work, grouping them together according to the structure of the algorithm.

### 8.2.1   Electrophysiology

The electrophysiology problem is a *reaction-diffusion* equation handled via operator splitting. The Mitchell-Schaeffer ionic membrane model is highly nonlinear and as a result is updated using an Explicit Euler time integration. The diffusive mono- and bi-domain models are integrated with Implicit Euler time integration.

The stiffness of the explicit problem demands a relatively short time step, and as a result the linear system for the implicit integration is dominated by the mass matrix. Thus it is extremely well-conditioned and thus suitable for an iterative solver such as a Jacobi or Richardson iteration. The properties of these solvers are well-known, removing high frequencies quickly and efficiently, but effectively failing for low frequencies.

At this point it is important to reconsider the reactive part of the equation. Charge diffuses to neighbouring tissue and will fire an action potential if it is over the stimulation threshold of the model. Thereafter the problem is completely changed and dominated by the newly depolarised region. As a result

the diffusion of charge at a distance, corresponding to the low-frequency solution is no longer relevant. The short time step and local nature of the problem lead to a well-conditioned system, requiring few iterations to reach a solution accurate in high frequencies.

This is an example of how consideration of the dynamic nature of the problem guides the design and efficiency of the algorithm.

### 8.2.2   Mechanics

The solid mechanics problem is again handled via operator-splitting. A Cauchy stress tensor field dependent upon both solid deformation via the nonlinear Finite Strain Theory, as well as fluid pressure is assembled. The divergence of the field is then calculated to give a resultant force which is used to update both solid and fluid velocity via the first step of a Leapfrog time integration scheme. This is a semi-implicit step, because it contains the implicit pressure correction provided by the fluid dynamics model. The Leapfrog is symplectic in the phase plane and as such conserves angular momentum, but the time step is limited by stability considerations.

The solid damping and fluid viscosity operators are then applied via an Implicit Euler scheme as for the diffusive part of the electrophysiology problem. Again, they benefit in exactly the same way. The sort time step and local behaviour of the dynamic problem allow for relatively few iterations of a Jacobi or Richardson solver.

### 8.2.3   Fluid-Stucture Interaction

There is a theme emerging in the design of the solver. If we accept that the nonlinear parts of the mathematical model are to be solved in an explicit manner, then we must in turn accept a short time step. The matrix- and mesh-free nature of the discretisation, combined with the suitability of the GPU for parallel iteration make this feasible. This in turn benefits the solution of the linear parts of the model.

The fluid-structure interaction problem takes this to its extreme. In order to know the movement of the surface it is necessary to solve the entire fluid dynamics problem to gain a reaction force. This in turn requires a knowledge of the solid velocity. As such there is a huge, highly nonlinear problem right at the centre of the entire cardiac simulation. Its solution dominates all other computational effort. The decision in design was to handle this problem in an explicit manner. This in turn limits the maximum time step for the entire model.

There is a benefit which comes from the the choice of Chorin projection in combination with Finite Volume Method. Since we update the solution with an explicit step then solve the Poisson Pressure equation, we are applying a *predictor-corrector* method. Firstly, the short time step provides and excellent initial guess for the pressure field from the previous iteration, and reduces the number of iterations to reach a reasonable solution. Secondly, if we do not make a complete solve of the pressure equation the error in the corrected solution is not lost, but remains due to the conservative nature of FVM into the next time step. Rather than a uniform pressure field we see a pressure wave travelling through the fluid as the tissue around it contracts. This is the speed of sound, a synthetic compressibility which enters the fluid model. Other methods, including the half stepping of the semi-implicit formulation of [Formaggia, 2009] have the same effect. Since the fluid does not correctly store the energy of compression, this is the largest source of error in our work. We hope to address this problem through the application of multi-resolution methods which are discussed in Section 8.3.

The advection kernel uses an interpolated gradient, and as such is subject to the well-known problem of 'false diffusion', where the solution of the advection problem also contains some mesh-dependent diffusion. In this case the diffusion is some artificial viscosity, which can be dealt with by underestimating the true viscosity. At higher mesh resolutions the problem is less significant.

### 8.2.4   GPU Processing

The use of the GPU itself brings with it some sources of compromise. In general, single precision arithmetic is the standard for graphics processing. This brings with it the obvious reduction in numerical precision, but also require care in the design calculation to avoid overflow. This can be seen in the convergence studies, where a finer mesh leads to division by a machine-zero volume. This problem can be overcome but is left in the work to demonstrate that whilst invisible it is always present.

Conversely there is a benefit in speed of processing and memory access associated with the smaller sized single precision variables. The design process aims to provide a tool of practical use, and thus the benefit in performance is seen to outweigh absolute numerical precision, especially since part of its objective is to determine the largely unknown parameters of the model.

The memory capacity of a single GPU processor limits the total size and resolution of the simulation. Since there is a minimum spatial mesh width $\delta x$ in space, there follows an absolute minimum time step $\delta t$.

The cost of communication across the PCI bus from GPU to CPU is too high to allow multi-processing per time step via MPI. There have been advances in

shared memory GPU multiprocessors, such as NVIDIA's NVLINK, but they are also limited in bandwidth. It is better to consider that it would be possible to run thousands of cardiac simulations in parallel on a supercomputer. As mentioned earlier, this makes the entire simulation an objective function for a set of optimisation problems, most notably parameter estimation.

## 8.3   Future Work

The title of the work states its aim, to provide a set of tools for cardiac simulation. The extensibility and simplicity of the code should allow for researchers in this field and others to adapt these techniques to their own work.

The development of the software has given insight into the behaviour of the tissue surrounding the heart and its potential contribution to AF and tachycardia. This will be investigated.

The entire design of the discretisation and solver make them ideal for multi-level solution. The current problem can be decomposed into many resolutions on both space and time. The representation of geometry via signed distance functions and the mesh-free assembly allow the entire problem to be reconstructed fully at any mesh size.

In order to generate a more physiologically accurate PV loop the design of valves must be improved. This can be done either physically or through the introduction of diodes into the fluid dynamics calculation.

The software allows for extension and refinement to include richer and more precise mathematical models. It should be possible in future to validate the simulation itself against physiological indicators. At this point the tools may be able to make a valid contribution to research and clinical practice.

# Bibliography

[Augustin et al., 2021] Augustin, C. M., Gsell, M. A. F., Karabelas, E., Willemen, E., Prinzen, F. W., Lumens, J., Vigmond, E. J., and Plank, G. (2021). A computationally efficient physiologically comprehensive 3d-0d closed-loop model of the heart and circulation. *Comput Methods Appl Mech Eng*, 386:114092.

[Balay et al., 1998] Balay, S., Gropp, W., McInnes, L. C., and Smith, B. F. (1998). Petsc, the portable, extensible toolkit for scientific computation. *Argonne National Laboratory*, 2(17).

[Briggs et al., 2000] Briggs, W. L., Henson, V. E., and McCormick, S. F. (2000). *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, USA.

[Chorin and Marsden, 2013] Chorin, A. J. and Marsden, J. E. (2013). *Mathematical Introduction to Fluid Mechanics*. Universitext. Springer London, Limited.

[Courtemanche et al., 1998] Courtemanche, M., Ramirez, R. J., and Nattel, S. (1998). Ionic mechanisms underlying human atrial action potential properties: insights from a mathematical model. *American Journal of Physiology-Heart and Circulatory Physiology*, 275(1):H301–H321.

[Del Corso et al., 2022] Del Corso, G., Verzicco, R., and Viola, F. (2022). A fast computational model for the electrophysiology of the whole human heart. *Journal of Computational Physics*, 457:111084.

[Fenton and Karma, 1998] Fenton, F. and Karma, A. (1998). Vortex dynamics in three-dimensional continuous myocardium with fiber rotation: Filament instability and fibrillation. *Chaos*, 8(1):20–47.

[FitzHugh, 1955] FitzHugh, R. (1955). Mathematical models of threshold phenomena in the nerve membrane. *The Bulletin of Mathematical Biophysics*, 17(4):257–278.

[Formaggia, 2009] Formaggia, L. (2009). *Cardiovascular Mathematics*. MS&A. Springer-Verlag Milan.

[Fung, 1981] Fung, Y. (1981). *Biomechanics: Mechanical Properties of Living Tissues*. Springer New York.

[Gerach et al., 2021] Gerach, T., Schuler, S., FrÃűhlich, J., Lindner, L., Kovacheva, E., Moss, R., WÃijlfers, E. M., Seemann, G., Wieners, C., and Loewe, A. (2021). Electro-mechanical whole-heart digital twins: A fully coupled multi-physics approach. *Mathematics*, 9(11):1247.

[Golub, 2013] Golub, G. H. (2013). *Matrix computations*. John Hopkins, third edition.

[Gonzalez and Stuart, 2008] Gonzalez, O. and Stuart, A. M. (2008). *A First Course in Continuum Mechanics*. Cambridge Texts in Applied Mathematics. Cambridge University Press.

[Griebel, 1997] Griebel, M. (1997). *Numerical simulation in fluid dynamics : a practical introduction*. SIAM monographs on mathematical modeling and computation. Society for Industrial and Applied Mathematics, Philadelphia.

[Guccione et al., 1995] Guccione, J. M., Costa, K. D., and McCulloch, A. D. (1995). Finite element stress analysis of left ventricular mechanics in the beating dog heart. *Journal of biomechanics*, 28 10:1167–77.

[Harris, 2005] Harris, M. (2005). Fast fluid dynamics simulation on the gpu. In Fernando, R., editor, *ACM SIGGRAPH 2005 Courses on*, pages 637–665. ACM Press.

[Hassaballah et al., 2013] Hassaballah, A. I., Hassan, M. A., Mardi, A. N., and Hamdi, M. (2013). An inverse finite element method for determining the tissue compressibility of human left ventricular wall during the cardiac cycle. *PLoS ONE*, 8(12):e82703.

[Holzapfel, 2000] Holzapfel, G. A. (2000). *Nonlinear Solid Mechanics*, volume 37. Wiley.

[Hughes et al., 1981] Hughes, T. J., Liu, W. K., and Zimmermann, T. K. (1981). Lagrangian-eulerian finite element formulation for incompressible viscous flows. *Computer Methods in Applied Mechanics and Engineering*, 29(3):329–349.

[Kaboudian et al., 2019] Kaboudian, A., Cherry, E. M., and Fenton, F. H. (2019). Real-time interactive simulations of large-scale systems on personal computers and cell phones: Toward patient-specific heart modeling and other applications. *Science Advances*, 5(3):eaav6019.

[Keener and Sneyd, 2010] Keener, J. and Sneyd, J. (2010). *Mathematical Physiology : I Cellular Physiology*. Interdisciplinary Applied Mathematics. Springer.

[Khalafvand et al., 2011] Khalafvand, S., Ng, E., and Zhong, L. (2011). CFD simulation of flow through heart: a perspective review. *Computer Methods in Biomechanics and Biomedical Engineering*, 14(1):113–132. PMID: 21271418.

[Khronos OpenCL Working Group, 2022] Khronos OpenCL Working Group (2022). *The OpenCL Specification Version 3.0.11*. Khronos.

[Kohl et al., 2001] Kohl, P., Noble, D., Hunter, P. J., KovÃącs, S. J., Mcqueen, D. M., and Peskin, C. S. (2001). Modelling cardiac fluid dynamics and diastolic function. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 359(1783):1299–1314.

[Land et al., 2015] Land, S., Gurev, V., Arens, S., Augustin, C. M., Baron, L., Blake, R., Bradley, C., Castro, S., Crozier, A., Favino, M., Fastl, T. E., Fritz, T., Gao, H., Gizzi, A., Griffith, B. E., Hurtado, D. E., Krause, R., Luo, X., Nash, M. P., Pezzuto, S., Plank, G., Rossi, S., Ruprecht, D., Seemann, G., Smith, N. P., Sundnes, J., Rice, J. J., Trayanova, N., Wang, D., Wang, Z. J., and Niederer, S. A. (2015). Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behaviour. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2184):20150641.

[Lee et al., 2009] Lee, J., Niederer, S., Nordsletten, D., Le Grice, I., Smaill, B., Kay, D., and Smith, N. (2009). Coupling contraction, excitation, ventricular and coronary blood flow across scale and physics in the heart. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1901):3331–3331.

[Luo and Rudy, 1994] Luo, C. H. and Rudy, Y. (1994). A dynamic model of the cardiac ventricular action potential. ii. afterdepolarizations, triggered activity, and potentiation. *Circ Res*, 74(6):1097–1113.

[Mitchell and Schaeffer, 2003] Mitchell, C. and Schaeffer, D. (2003). A two-current model for the dynamics of cardiac membrane. *Bulletin of Mathematical Biology*, 65(5):767–793.

[Mooney, 1940] Mooney, M. (1940). A theory of large elastic deformation. *Journal of Applied Physics*, 11(9):582–592.

[Morton, 1966] Morton, G. (1966). *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company.

[Nagumo et al., 1962] Nagumo, J., Arimoto, S., and Yoshizawa, S. (1962). An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070.

[Nash and Panfilov, 2004] Nash, M. P. and Panfilov, A. V. (2004). Electromechanical model of excitable tissue to study reentrant cardiac arrhythmias.

*Progress in Biophysics and Molecular Biology*, 85(2):501–522. Modelling Cellular and Tissue Function.

[Neic et al., 2017] Neic, A., Campos, F. O., Prassl, A. J., Niederer, S. A., Bishop, M. J., Vigmond, E. J., and Plank, G. (2017). Efficient computation of electrograms and ecgs in human whole heart simulations using a reaction-eikonal model. *Journal of Computational Physics*, 346:191–211.

[Neic et al., 2012] Neic, A., Liebmann, M., Hoetzl, E., Mitchell, L., Vigmond, E. J., Haase, G., and Plank, G. (2012). Accelerating cardiac bidomain simulations using graphics processing units. *IEEE Transactions on Biomedical Engineering*, 59(8):2281–2290.

[Newton, 1687] Newton, I. (1687). *Philosophiae naturalis principia mathematica*. William Dawson & Sons Ltd., London.

[Ngoma et al., 2017] Ngoma, D. V. P., Bourgault, Y., Pop, M., and Nkounkou, H. (2017). Adjustment of parameters in ionic models using optimal control problems. In *Functional Imaging and Modelling of the Heart*, pages 322–332. Springer, Springer International Publishing.

[Niederer et al., 2011a] Niederer, S., Kerfoot, E., Benson, A., Bernabeu, M., Bernus, O., Bradley, C., Cherry, E., Clayton, R., Fenton, F., Garny, A., Heidenreich, E., Land, S., Maleckar, M., Pathmanathan, P., Plank, G., Rodríguez, J., Roy, I., Sachse, F., Seemann, G., Skavhaug, O., and Smith, N. (2011a). Verification of cardiac tissue electrophysiology simulators using an n-version benchmark. *PHILOSOPHICAL TRANSACTIONS- ROYAL SOCIETY OF LONDON SERIES A MATHEMATICAL PHYSICAL AND ENGINEERING SCIENCES*, 369(1954):4331–51.

[Niederer et al., 2018] Niederer, S., Lumens, J., and Trayanova, N. (2018). Computational models in cardiology. *Nature Reviews Cardiology*, 16(2):100–111.

[Niederer et al., 2011b] Niederer, S., Mitchell, L., Smith, N., and Plank, G. (2011b). Simulating human cardiac electrophysiology on clinical timescales. *Frontiers in Physiology*, 2.

[Nordsletten et al., 2011a] Nordsletten, D., McCormick, M., Kilner, P. J., Hunter, P., Kay, D., and Smith, N. P. (2011a). Fluidâ€Şsolid coupling for the investigation of diastolic and systolic human left ventricular function. *International Journal for Numerical Methods in Biomedical Engineering*, 27(7):1017–1039.

[Nordsletten et al., 2011b] Nordsletten, D., Niederer, S., Nash, M., Hunter, P., and Smith, N. (2011b). Coupling multi-physics models to cardiac mechanics. *Progress in Biophysics and Molecular Biology*, 104(1-3):77–88. Cardiac Physiome project: Mathematical and Modelling Foundations.

[NVIDIA Corporation, 2011] NVIDIA Corporation (2011). *OpenCL Best Practices Guide*. NVIDIA Corporation.

[NVIDIA Corporation, 2012] NVIDIA Corporation (2012). *OpenCL Programming Guide for the CUDA Architecture Version 4.2*. NVIDIA Corporation.

[Ogden and Hill, 1972] Ogden, R. W. and Hill, R. (1972). Large deformation isotropic elasticity – on the correlation of theory and experiment for incompressible rubberlike solids. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 326(1567):565–584.

[O'Hara et al., 2011] O'Hara, T., VirÃąg, L., VarrÃş, A., and Rudy, Y. (2011). Simulation of the undiseased human cardiac ventricular action potential: Model formulation and experimental validation. *PLOS Computational Biology*, 7(5):1–29.

[Patankar, 1980] Patankar, S. V. (1980). Numerical heat transfer and fluid flow(book). *Washington, DC, Hemisphere Publishing Corp., 1980. 210 p*.

[Potse et al., 2006] Potse, M., Dube, B., Vinet, A., and Cardinal, R. (2006). A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE.

[Quarteroni et al., 2017] Quarteroni, A., Lassila, T., Rossi, S., and Ruiz Baier, R. (2017). Integrated heart - coupling multiscale and multiphysics models for the simulation of the cardiac function. *Computer Methods in Applied Mechanics and Engineering*, 314:345–407.

[Reddy, 2013] Reddy, J. N. (2013). *An Introduction to Continuum Mechanics, Second Edition*. Cambridge University Press, 2 edition.

[Rivlin and Rideal, 1948] Rivlin, R. S. and Rideal, E. K. (1948). Large elastic deformations of isotropic materials iv. further developments of the general theory. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 241(835):379–397.

[Rush and Larsen, 1978] Rush, S. and Larsen, H. (1978). A practical algorithm for solving dynamic membrane equations. *IEEE Transactions on Biomedical Engineering*, BME-25(4):389–392.

[Santiago et al., 2018] Santiago, A., Aguado-Sierra, J., Zavala-AkÃľ, M., Doste-Beltran, R., GÃşmez, S., ArÃṣs, R., Cajas, J. C., Casoni, E., and VÃązquez, M. (2018). Fully coupled fluid-electro-mechanical model of the human heart for supercomputers. *International Journal for Numerical Methods in Biomedical Engineering*, 34(12):e3140. e3140 cnm.3140.

[Scarpino, 2012] Scarpino, M. (2012). *OpenCL in action*. Manning.

[Sugiura et al., 2012] Sugiura, S., Washio, T., Hatano, A., Okada, J., Watanabe, H., and Hisada, T. (2012). Multi-scale simulations of cardiac electrophysiology and mechanics using the university of tokyo heart simulator. *Prog Biophys Mol Biol*, 110(2-3):380–389.

[Tang et al., 2007] Tang, D., Yang, C., Geva, T., and del Nido, P. J. (2007). Two-layer passive/active anisotropic fsi models with fiber orientation: Mri-based patient-specific modeling of right ventricular response to pulmonary valve insertion surgery. *Mol Cell Biomech*, 4(3):159–176.

[ten Tusscher et al., 2004] ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., and Panfilov, A. V. (2004). A model for human ventricular tissue. *American Journal of Physiology-Heart and Circulatory Physiology*, 286(4):H1573–H1589. PMID: 14656705.

[TenTusscher and Panfilov, 2006] TenTusscher, K. H. W. J. and Panfilov, A. V. (2006). Cell model for efficient simulation of wave propagation in human ventricular tissue under normal and pathological conditions. *Physics in Medicine and Biology*, 51(23):6141.

[Trayanova et al., 2004] Trayanova, N., Li, W., Eason, J., and Kohl, P. (2004). Effect of stretch-activated channels on defibrillation efficacy. *Heart Rhythm*, 1(1):67–77.

[Trayanova and Winslow, 2011] Trayanova, N. A. and Winslow, R. (2011). Whole-heart modeling: Applications to cardiac electrophysiology and electromechanics. *Circulation Research*, 108(1):113–128.

[Trottenberg et al., 2000] Trottenberg, U., Oosterlee, C. W., and Schuller, A. (2000). *Multigrid*. Academic Press, USA.

[Versteeg and Malalasekra, 1996] Versteeg, H. and Malalasekra, W. (1996). *An Introduction to Computational Fluid Dynamics*. Prentice Hall.

[Verzicco, 2022] Verzicco, R. (2022). Electro-fluid-mechanics of the heart. *Journal of Fluid Mechanics*, 941:P1.

[Viola et al., 2020] Viola, F., Meschini, V., and Verzicco, R. (2020). Fluid–structure-electrophysiology interaction (fsei) in the left-heart: A multi-way coupled computational model. *European Journal of Mechanics - B/Fluids*, 79:212–232.

[Viola et al., 2022] Viola, F., Spandan, V., Meschini, V., Romero, J., Fatica, M., de Tullio, M. D., and Verzicco, R. (2022). Fsei-gpu: Gpu accelerated simulations of the fluid–structure–electrophysiology interaction in the left heart. *Computer Physics Communications*, 273:108248.

[Watanabe et al., 2004] Watanabe, H., Sugiura, S., Kafuku, H., and Hisada, T. (2004). Multiphysics simulation of left ventricular filling dynamics us-

ing fluid-structure interaction finite element method. *Biophysical Journal*, 87(3):2074–2085.

[Westerhof et al., 1971] Westerhof, N., Elzinga, G., and Sipkema, P. (1971). An artificial arterial system for pumping hearts. *Journal of Applied Physiology*, 31(5):776–781. PMID: 5117196.

[Zhang et al., 2016] Zhang, Y., Barocas, V. H., Berceli, S. A., Clancy, C. E., Eckmann, D. M., Garbey, M., Kassab, G. S., Lochner, D. R., McCulloch, A. D., Tran-Son-Tay, R., and Trayanova, N. A. (2016). Multi-scale modeling of the cardiovascular system: Disease development, progression, and clinical intervention. *Annals of Biomedical Engineering*, 44(9):2642–2660.