Spatio-Temporal Deformation Analysis of Cardiac MR Images

HARI SUNDAR

A DISSERTATION

in

BIOENGINEERING

Presented to the Faculties of the University of Pennsylvania
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

2009

PROFESSOR CHRISTOS DAVATZIKOS
Supervisor of Dissertation

PROFESSOR SUSAN MARGULIES
Graduate Group Chairperson

Abstract

Cardiac diseases claim more lives than any other disease in the world. Early diagnosis and treatment can save many lives and reduce the associated socio-economic costs. Cardiac diseases are characterized by both changes in the myocardial structure as well as changes in cardiac function. *Diffuse cardiomyopathies* in particular (as opposed to *ischemic cardiomyopathies*) are difficult to diagnose by structural changes. Advances in MR Cine imaging methods have enabled us to acquire high-resolution 4D images of the heart that capture the structural and functional characteristics of individual hearts. However large inter and intra-observer variability in the interpretation of these images for the diagnosis of diffuse cardiomyopathies has been reported. This has created the need for sophisticated and highly automated image analysis methods, which can identify and precisely quantify subtle and spatially complex patterns of functional changes in the heart. The development of such methods is the primary goal of this project.

We propose a method for the analysis of Magnetic Resonance (MR) cardiac images with the goal of reconstructing the motion of the myocardial tissue. The main feature of our method is that the inversion parameter field is the active contraction of the myocardial fibers. This is accomplished with a biophysically-constrained, four-dimensional (space plus time) formulation that aims to complement information that can be gathered from the images by a priori knowledge of cardiac mechanics and electrophysiology. Incorporating biomechanical priors introduces major computational challenges, which constitute the main issue tackled by this thesis.

Our main hypothesis is that by incorporating biophysical information, we can generate more informative priors and thus, more accurate predictions of the ventricular wall motion. In this thesis, we outline the formulation, discuss the computational methodology for solving the inverse motion estimation, and present results using synthetic and tagged MR data. The major contribution in this regard is the development of a parallel octree-based adaptive meshing for finite element computations. This allows us to model complex geometries in an adaptive manner and solve them in parallel. For the purpose of cardiac motion estimation, we present methods for generating and solving using the spatially non-uniform octree meshing scheme with an adjoint-based inversion solver. The method uses myocardial fiber information, adapted to the subject, to reconstruct the motion.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis we present methods for biophysically constrained, image based estimation of myocardial motion. Myocardial motion is a primary measure of myocardial function, and differences in myocardial function can help diagnose and quantify cardiomyopathies better than myocardial structural differences. Our main hypothesis is that by incorporating biophysical information, we can generate more informative priors and thus, more accurate predictions of the ventricular wall motion. The main feature of our method is that we estimate the active contraction of the myocardial fibers instead of the displacements. This is accomplished with a biophysically-constrained, four-dimensional (space plus time) formulation that aims to complement information that can be gathered from the images by a priori knowledge of cardiac mechanics.

1.1 Motivation

Cardiac diseases claim more lives than any other disease in the world [4]. Early diagnosis and treatment can save many lives and reduce the associated socio-economic costs. Cardiac diseases are characterized by both changes in the myocardial structure as well as changes in cardiac function. Consequently, it is important for any cardiac diagnosis method to consider both these aspects.

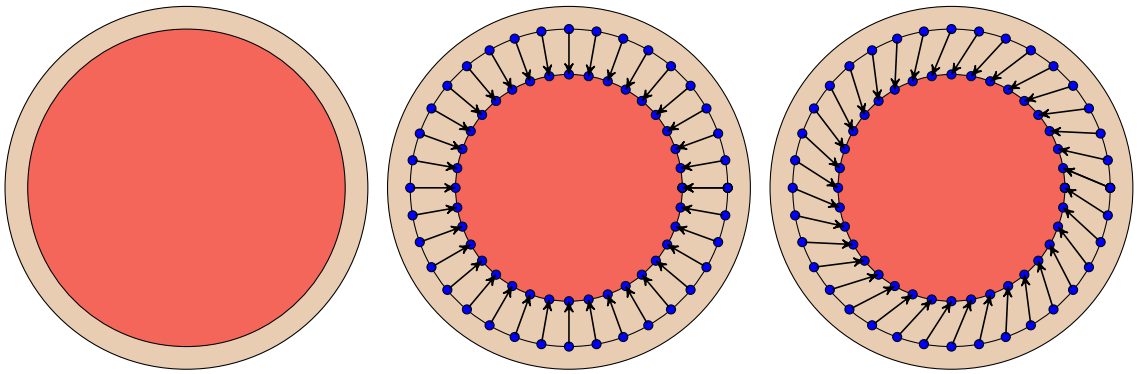
Imaging can help in the diagnosis of cardiac masses, cardiomyopathy, myocardial infarcion, and valvular disease. Advances in Cine Magnetic Resonance (MR) imaging methods

have enabled us to acquire high-resolution 4D images of the heart that capture the structural and functional characteristics of individual hearts. Studies suggest that standardizing acquisition protocols and objective analysis especially of regional myocardial function will help improve the accuracy and reduce inter-observer variability [11]. This has created the need for sophisticated and highly automated image analysis methods, which can identify and precisely quantify subtle and spatially complex patterns of functional changes in the heart. The development of such methods is the primary goal of this thesis.

1.2 Overview

Cardiomyopathies are characterized both by structural changes in the myocardial structure as well as changes in myocardial function. It is important to consider both these changes when designing an automatic diagnosis algorithm. The method by which the structural and functional information is obtained from the patient should be simple and account for minimal patient discomfort and trauma. Magnetic resonance imaging (MRI) has been very popular in the recent years for acquiring structural information in the human body. Apart from providing us with structural information of the myocardium, these images can provide us with myocardial wall motion if they can be extracted accurately.

The task of extracting myocardial wall motion from MR Cine sequences is challenging, and although a lot of work has been done in this field, most of the approaches rely purely on image similarity to drive the motion estimation. It is important to keep in mind that image matching does not necessarily imply good anatomical matching and consequently motion estimation. As is common in image registration, regularization (typically L_2) is usually used to constrain the motion estimation process. When used for motion estimation, only a smoothing regularization like L_2 is not sufficient and we are not guaranteed to get the true underlying motion. This is illustrated in Figure 1.1, where we see that two different motions can produce the same configuration. Because of this reason, it is important to provide intelligent priors to help the motion estimation algorithm obtain physically correct motions. In this thesis we present a framework for incorporating physics based constraints,



(a) (b) (c)

Figure 1.1: A schematic of the left ventricle is shown in (a). The same contracted configuration can be reached via two deformations; (b) via radial contractions, and (c) via a twist.

in the form of partial differential equations (pde), for the motion estimation problem.

1.3 Related work

Cardiomyopathies present themselves in different forms, both by structural changes like plaque formation, fat deposits, etc., and functional changes like variations in ventricular wall motion, ejection fraction, and perfusion. Both of these need to be extracted from the image before accurate diagnosis can be done. A lot of work has been done in feature extractors for structural characterizations of disease. This has focused primarily on extracting image features like intensities and gradients [73], moments [86, 100], Gabor features [54], and local frequency representations [48]. The problem with cardiomyopathies is that not all of them can be characterized by structural changes. Function at rest may be abnormal as a result of one of the spectrum of ischemic heart diseases (ischemia, infarction, hibernation) or of cardiomyopathy from other causes. During stress testing, new or worsening wall motion abnormalities are indicative of functionally significant coronary artery stenosis [90]. In addition, wall motion imaging to detect regional contractile reserve is an accurate measure of myocardial viability, and the results can help guide coronary revascularization therapy. Characterizing cardiomyopathies based on both structural and functional changes

will make the diagnosis algorithm more accurate and robust. Of course quantization of the myocardial wall motion represents a challenge in itself.

Most clinical modalities used to image myocardial function evaluate passive wall motion (ventriculography) or wall thickening (echocardiography, gated single-photon emission computed tomography, or cine MR imaging). MR imaging also allows quantitative measurement of regional intramyocardial motion and, subsequently, strain, which can be more sensitive to wall motion abnormalities than is wall thickening. MR imaging methods for the quantification of intramyocardial wall motion can be loosely classified into two approaches, those relying on specially developed MR imaging protocols to help in the estimation of myocardial motion and those relying on image analysis techniques to extract motion estimates from MR Cine sequences.

1.3.1 Specialized MR Protocols

MR Tagging

MR Tagging was developed to provide non-invasive virtual markers inside the myocardium, which deform with myocardial motion [120]. MR imaging and especially tagged MR are currently the reference modalities to estimate dense cardiac displacement fields with high spatial resolution. The deformation fields, as well as the derived motion parameters such as myocardial strain can be determined within an accuracy of 2mm x 2mm [89, 17]. The primary disadvantage of tagging is the reduced spatial resolution of strain relative to the image spatial resolution. In tagging, after the displaced tag lines are detected [51], the displacement field can be estimated and intramyocardial strain can be computed in a variety of ways [120]. With this approach, although strain may be interpolated to any desired spatial resolution, the fundamental spatial resolution of strain is nominally determined by the distance between the tag lines, which is typically several pixels. Tag detection has an additional disadvantage in that it typically requires substantial manual intervention and is therefore a time-consuming task. Harmonic phase analysis will likely obviate tag detection [62], but the spatial resolution of the resultant strain maps will not necessarily improve.

The spatial resolution of strain maps obtained from tagged images after harmonic phase analysis is determined by the k-space filter of the analysis; in practice with single breath-hold acquisitions, the resolution has been relatively poor [41]. Additionally since the right ventricle (RV) is much thinner than the left ventricle (LV), it is difficult to place more than a single tag within the RV, making the estimation of RV motion extremely difficult and inaccurate. Since we are most interested in the characterizing RV function, tagging is not appropriate for our purpose.

Phase Contrast Imaging

The second approach is that of MR phase contrast imaging [103], which is based on the concept that spins that are moving in the same direction as a magnetic field gradient develop a phase shift that is proportional to the velocity of the spins. This information can be used directly to determine the velocity of the spins, or in the cardiac case the velocity of any point within the myocardium. The main problem with this approach is that four acquisitions have to be made for each heart, one the regular MR cine sequence and one phase contrast acquisition each for the velocity components in the x, y, and the z directions. Consequently, MR phase contrast imaging is not used much in a clinical setting.

DENSE and HARP

Displacement encoded imaging with stimulated echoes (DENSE) [21] and harmonic phase imaging (HARP) [62] employ 1-1 spatial modulation of magnetization to cosine modulate the longitudinal magnetization as a function of position at end diastole. Later in the cardiac cycle the cosine-modulated signal is sampled and used to compute myocardial strain from the signal phase. The sampled signal generally includes three distinct echoes: a displacement encoded stimulated echo, the complex conjugate of the displacement-encoded echo, and an echo arising from T1 relaxation. If the T1 relaxation and complex conjugate echoes are suppressed, then a phase image representing just the displacement encoded echo can be reconstructed.

While the HARP method was introduced as a post-processing method for SPAMM-tagged images [62], DENSE arose from the framework of stimulated echo (STE) and displacement encoding using bipolar gradients [21]. Both techniques have evolved since their introduction, thereby becoming more similar over time and losing their distinct features. Newly introduced improvements have successfully been applied in both methods. DENSE is now used without its distinct STE timing, and for cine acquisition in conjunction with k-space filtering. HARP is used in conjunction with peak suppression methods that alleviate the need for band-pass filtering, and with a readout window targeted at one single tagging peak.

However, data acquisition in single breathhold DENSE MR imaging has been limited to only one cardiac phase. Multiple breathhold DENSE produces images at multiple phases of the cardiac cycle, but the resolution has been poor and is fundamentally a 2D approach and the estimation of through-plane displacement has been poor.

1.3.2 Extracting motion from MR Cine images

An alternate approach is to estimate myocardial motion from MR Cine sequences. MR Cine images in a clinical setting at sub millimeter resolutions (in-plane), with slice thickness in the range of $6 - 10$mm. The temporal resolution varies between $25 - 70$ms. A lot of work has been done in extracting cardiac motion fields from MR Cine sequences [45, 57, 63, 65, 89, 92, 113]. These can be classified into two main categories. The first approach uses segmentation of the myocardial wall, followed by geometric and mechanical modeling using active contours or surfaces to extract the displacement field and to perform the motion analysis [63, 89, 113]. For matching two contours or surfaces, curvatures are frequently used to establish initial sparse correspondences, followed by the dense correspondence interpolation in other myocardial positions by regularization or mechanical modeling [57, 89]. The lack of distinct landmarks on the myocardial wall makes it difficult to estimate the wall motion based on surface tracking. In addition this approach is very sensitive to the accuracy with which the myocardium can be segmented. Also it

performs poorly in regions within the myocardium, and manages to only align the myocardial boundaries. The other approach uses energy-based warping or optical flow techniques to compute the displacement of the myocardium [45, 65, 92]. Perperidis et al. [65] use a regular grid with a B-spline basis to model the deformation and use normalized mutual information as the similarity metric which is calculated over the whole image. One of the major shortcomings of these approaches is that the transformation estimated as a result of the registration is not unique and in fact does not necessarily conform to the underlying myocardial motion. The same algorithm can give different estimates of motion for different initial guesses and different parameters. The problem arises since these methods attempt to maximize the image similarity with only a smoothness constraint on the transformation. Since there can be many transformation that can map an image onto another (especially sparsely sampled ones as in the case of MR Cines) there is no guarantee that the estimated transformation is the correct one [14, 15]. These methods estimate motion by evolving the current estimate of motion under the action of external image forces (image similarity) and internal forces which constrain the regularity of the motion (smoothness). Such regularizers work well with respect to noise removal but they do not incorporate a priori knowledge of the underlying cardiac motion. Incorporating a biomechanically-inspired model for the myocardium has the potential for a more accurate motion estimation [56]. Functional models of the heart are direct computational models, designed to reproduce in a realistic manner the cardiac activity, often requiring high computational costs and the manual tuning of a very large set of parameters. Such methods can be computationally prohibitive for our purposes, and we instead select a level of modeling compatible with reasonable computing times and involving a limited number of parameters. Such simplifications add additional modeling errors, but our hypothesis is that in spite of these modeling errors the estimated motion fields shall be more accurate than those obtained from approaches not incorporating a priori knowledge. A detailed and thorough review of cardiac image registration methods can be found in [53] and a general review of image registration methods can be found in [121].

1.3.3 Motion estimation using biomechanical models

To address these problems in motion reconstruction, one of the main thrusts in recent research has been 4D motion estimation using biomechanical models. There is significant work on the integration of imaging with cardiovascular mechanics. In [38], a piecewise linear composite biomechanical model was used to determine active forces and the strains in the heart based on tagged MRI information. In [63] and [64], echocardiography and MR images were combined with biomechanical models for cardiac motion estimation. Interactive segmentation was combined with a Bayesian estimation framework that was regularized by an anisotropic, passive, linearly elastic, myocardium model. The authors recognized the importance of neglecting active contraction of the left ventricle. In [81, 82, 83], the need for realistic simulations and the need for inversion and data assimilation was outlined. In [59], Kalman filters were used to recover the initial state of the heart and spatial abnormalities. That method however, is difficult to generalize to nonlinear inversion with time-dependent inversion parameters.

1.3.4 Large-scale finite element (FEM) calculations

Although there are several arguments in favor of incorporating bio-mechanical models to constraint medical imaging algorithms, the main impediments to their widespread use have been the confidence in the fidelity of the models and the computational cost associated with FEM calculations. Two common approaches aimed at reducing the computation time for such calculations are parallelization and adaptive meshing. To this effect, we propose a new parallel, octree based system for performing FEM calculations and demonstrate its efficacy in solving large FEM calculations resulting from the discretization of partial differential equations.

There is a large literature on large-scale (FEM) calculations. Here we limit our review to recent work that has scaled to thousands of processors. One of the largest calculations was reported in [8]. The authors proposed a scheme for conforming discretizations and multigrid solvers on semi-structured meshes. Their approach is highly scalable for nearly

structured meshes and for constant coefficient PDEs. However, it does not support adaptive discretizations, and its computational efficiency diminishes in the case of variable coefficient operators. Examples of scalable approaches for unstructured meshes include [1] and [55]. In those works multigrid approaches for general elliptic operators were proposed. The associated constants for constructing the mesh and performing the calculations however, are quite large: setting up the mesh and building the associated linear operators can take thousands of seconds. A significant part of CPU time is related to the multigrid scheme (we do not consider multigrid in this thesis); even in the single grid cases however, the run times are quite large.

The high-costs related to partitioning, setup, and accessing generic unstructured grids, has motivated the design of octree-based data structures. Such constructions have been used in sequential and modestly parallel adaptive finite element implementations [7, 33, 43, 69], and many-body algorithms [32, 35, 118, 119]. State-of-the-art in-core and out-of-core implementations for finite elements can be found in [2, 106, 107, 108, 109]. In these papers the authors propose a top-down octree construction resulting in a Morton-ordered tree, followed by a 2:1 balance constraint using a novel algorithm (the parallel prioritized ripple propagation). Once the tree is constructed, a finite-element framework is built on top of the tree structure. Evaluating the discrete Laplacian operators (a matrix-free matrix-vector multiplication) requires three tree traversals, one traversal over the elements, and two traversals to apply projection over the so-called “hanging nodes”, to enforce a conforming discretization [43].

1.4 Contributions

An overall workflow corresponding to the proposed motion estimation process in shown in Figure 1.2. This allows us to appreciate the contributions in the context of the overall goal of cardiac motion estimation. The major contributions of this thesis are,

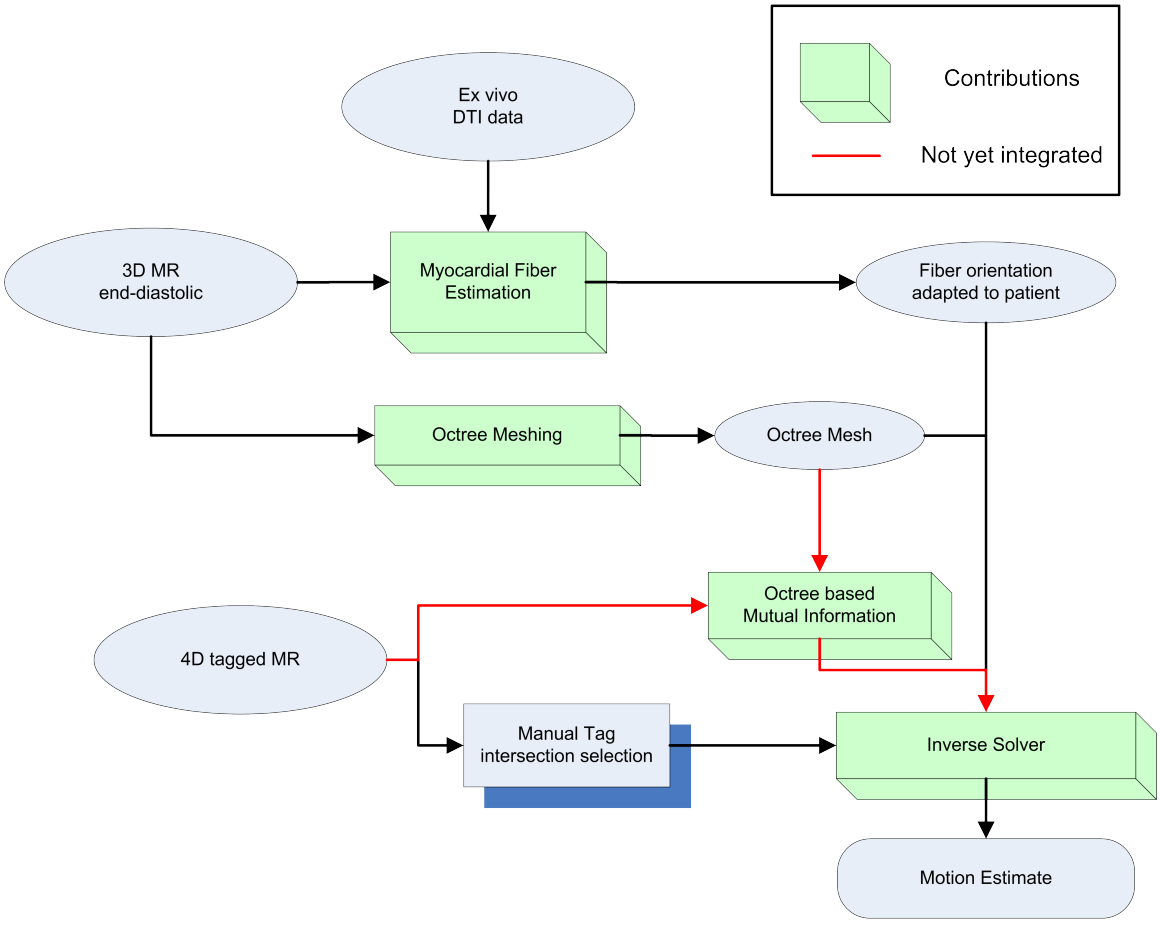


Figure 1.2: Overview of the motion estimation workflow. The major contributions of this thesis are shown as green boxes. The part of the workflow not yet integrated into the system is shown using the red arrows.

Myocardial fiber estimation Development of an automatic method for estimation of cardiac fiber-bundle orientations conforming to patient geometry using template warping of *in vitro* diffusion tensor images. This work was published in [98].

Octree Meshing Development of a parallel octree-based adaptive meshing for finite element computations. Specifically, a parallel bottom-up algorithm for constructing and for enforcing 2:1 balance refinement in linear octrees was developed. The algorithm constructs the minimum number of nodes to satisfy the 2:1 constraint. Its key feature is that it avoids parallel searches, which as we show in sections 3.2.5 and 3.2.5, are the main hurdles in achieving good isogranular scalability. In addition, an algorithm to build element-to-nodal connectivity information efficiently and to compress this information in a way that achieves a three-fold compression (a total of four words per octant). These methods are explained in sections 3.3 and 3.3.2. We developed a lookup table based conforming discretization scheme that requires only a single traversal for the evaluation of a partial differential operator. This work was published in [97, 96].

Octree based Mutual Information A novel and efficient way of computing mutual information. The proposed method defines a non-uniform, adaptive sampling scheme for estimating the entropies of the images, which is less vulnerable to local maxima as compared to uniform and random sampling. This work was published in [99].

Inverse Solver A patient-specific image-based inversion formulation for the active forces in the myocardium using an octree-based, adaptive finite-element forward solver that incorporates anatomically-accurate fiber information and an adjoint/Hessian-based inversion algorithm.

1.5 Limitations

The limitations of our current implementation (but not the overall methodological framework) are the assumptions of linear geometric and material response and the potential bias

due to template-based fibers that does not account for anatomical variability, that is still requires some preprocessing of the initial frame to assign material properties and fiber orientation, that assumes zero residual stresses and initial conditions, and that it does not include an electrophysiology model.

Our on-going work includes transition to an intensity-based image-registration inversion (in which case we need to solve a nonlinear inversion). We hypothesize that image-derived information will provide a dense field that will constrain the inverse estimation more than the sparse observations which are available from the tagged MR images, as used in this work. Among the many open problems are the level of required model complexity for clinically relevant motion reconstructions, the bias of the fibers, the sensitivity to the values of the material properties, and the sensitivity to the image similarity functional.

1.6 Organization of this thesis

The rest of the thesis is organized thus; In Chapter 2, we first discuss the anatomical issues that were considered while designing the mechanical model of the heart. This is followed by a method for estimating the cardiac fiber bundle orientations, which are an important part of the model. Finally, we present the linear elastic, fiber based model of the heart and present results from validation studies. In Chapter 3 we present algorithms for the construction, 2:1 balance refinement, meshing and a distributed array for solving linear systems. This is followed by our algorithm for estimating mutual information using octrees meshes. In Chapter 4 we present the pde constrained optimization framework for estimation of myocardial motion. We also present validation studies that assess the estimation accuracy of the proposed method on synthetic and tagged MR datasets.

Chapter 2

Biophysical Model of the Heart

In this chapter we describe the anatomical structure of the human heart and describe the aspects that are taken into consideration while constructing the biophysical model. The model does not attempt to be a generic all encompassing model of the heart, but attempts to factor key aspects of cardiac dynamic that are important for the successful estimation of myocardial motion. The use of this model to constrain the problem of cardiac motion estimation shall be explained in detail in Chapter 4. The heart is a complicated system, with an electro-mechanical system responsible for the activation and contraction of the heart muscles. Modeling of cardiac anatomy, electrophysiology and mechanics is an active research field and a comprehensive review of the field can be found in [76].

Muscle fiber orientations need to be considered while modeling cardiac electromechanics. The diffusion properties of the muscle fibers play an important role in the propagation of cardiac activation current. Similarly the force generated by the muscles is primarily along the fiber direction. Therefore knowledge of the diffusion properties within the myocytes or at least the myocyte orientations is very important for modeling purposes, especially if the patient specific models are desired. Diffusion Tensor MR Imaging (DTI) [80, 105] is a promising new imaging modality that allows the diffusion tensors within tissues to be quantified. Additionally, the principal direction of the diffusion tensor aligns with myocyte orientations [75, 37]. However, it is not possible to obtain cardiac DTI *in vivo* currently, and as a result most modeling approaches use synthetic fiber orientations [76]. Synthetic fiber orientations are in general very simple and capture only the basic trends

in the orientation. Instead of using synthetic models for myocyte orientations we estimate patient specific myocyte orientation using ex vivo DTI obtained from a human volunteer. The process of warping the *ex vivo* DT to patient specific DT shall be described in detail in Section 2.3.

Since our aim is to use the bio-mechanical model to constrain the problem of cardiac motion estimation, and specifically to address the shortcomings of image registration based motion estimation algorithms, our focus is in incorporating the fiber-bundle orientations. The fiber-orientation provide the additional information that is lacking in image registration based motion estimation algorithms, which is the direction of the internal forces. In order to keep the computational complexity low, all other aspects of the mechanical model have been keep intentionally simple. When comparing the proposed model with other models, like [64, 75, 76, 83], it is important to keep in mind that the proposed model is designed specifically for the task of motion estimation.

Cardiac myocytes on being excited contract along the myocyte orientation. This contraction causes the atria and ventricles to contract, in turn causing blood to ciculate within the body. We model the myocardium, blood and surrounding coelomic cavity as linear elastic materials and the active forces within the myocardium caused as a result of the myocyte contraction. The linear elastodynamic model is described in Section 2.5 and the formulation of the active forces is described in Section 2.6. Finally we present experimental results validating the various models described in this Chapter in Section 2.7. We start by giving a brief overview of cardiac anatomy.

2.1 Anatomical Structure of the Heart

Modeling of cardiac anatomy, electrophysiology, and mechanics is very important for understanding the complicated interactions that take place between different anatomical structures. This knowledge helps us to understand the mechanisms of heart failure, and can help devise ways to prevent and cure such pathologies. A large number of cardiac pathologies

occur because of problems with the electro-mechanical system within the heart. Consequently, a lot of active research is being carried out in this field. A comprehensive review of the field can be found in [76].

The walls of the heart are composed of cardiac muscle, called myocardium. It consists of four compartments: the right and left atria and ventricles. The heart is oriented so that the anterior aspect is the right ventricle while the posterior aspect shows the left atrium. The left ventricular free wall and the septum are much thicker than the right ventricular wall. This is logical since the left ventricle pumps blood to the systemic circulation, where the pressure is considerably higher than for the pulmonary circulation, which arises from right ventricular outflow. Since a muscle fiber can contract only in one direction, the heart structure is complex, to succeed at pumping the blood. Anatomically, to achieve this, the muscle walls of the ventricles and the atria are composed of a single helically folded muscular structure as can be seen in Figure 2.1. The cardiac muscle fibers are divided into four groups [30]: Two groups of fibers wind around the outside of both ventricles. Beneath these fibers a third group winds around both ventricles. Beneath these fibers a fourth group winds only around the left ventricle.

As a result of this complicated geometry, muscle fiber orientations need to be considered while modeling cardiac electro-mechanics. At present the prevailing hypothesis is that the diffusion properties of the muscle fibers play an important role in the propagation speed of cardiac activation current as the diffusion tensor is proportional to the local conductivity, and also in the estimation of the orientation of forces generated by the muscles which are along the fiber direction. Therefore, knowledge of the diffusion tensor or at least the fiber orientations is very important for modeling purposes, especially if patient specific models are desired. However, *in-vivo* diffusion tensor imaging of the heart is not a practically realizable application at present, and as a result most modeling approaches use synthetic data for the fiber orientations. A common approach is to vary the elevation angle between the fiber and the short axis plane between $+90^\circ$ and -90° from the endocardium to the epicardium [76, 84]. These models of fiber orientations capture only the overall trends in orientation, and thus are not sufficient for patient specific modeling.

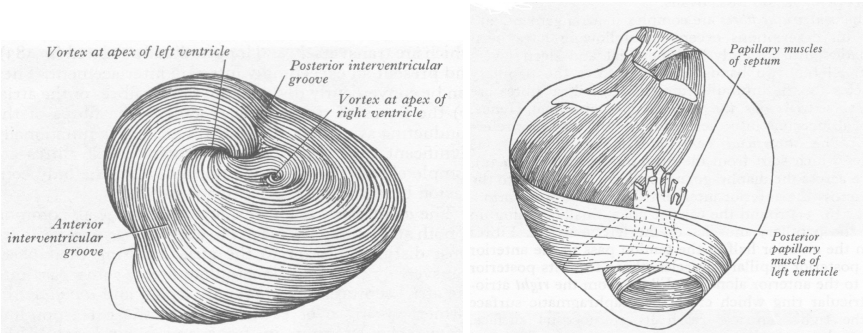


Figure 2.1: Fiber orientations in the Human Heart showing the helical structure of the muscles (from [30])

2.2 Diffusion Tensor Imaging

Diffusion tensor Imaging (DTI) is a technique to measure the anisotropic diffusion properties of biological tissues within the sample. This allows us to noninvasively infer the structure of the underlying tissue. Diffusion properties allow the classification of different types of tissues and can be used for tissue segmentation and detecting tissue orientations. The principal eigenvector of the diffusion tensor is known to align with fiber tracts in the brain [66] and in the heart [80]. Heart fibers reconstructed from *ex vivo* cardiac DTI are shown in Figure 2.2.

Diffusion tensors describe the diffusion properties of water molecules. In tissues diffusion properties are dictated by the cell structure of the tissue. Since cell membranes are selectively permeable, water molecules can move easily within a cell, but their diffusion across the membrane is limited. Thus diffusion properties of the tissue reflect the shape and orientation of the cells. For the specific case of elongated cells like the cardiac muscles, the diffusion will be maximum along the primary axis of the muscle, which also happens to be the direction along which maximum strain is developed.

Diffusion is measured through a diffusion coefficient, which is represented as a symmetric second order tensor:

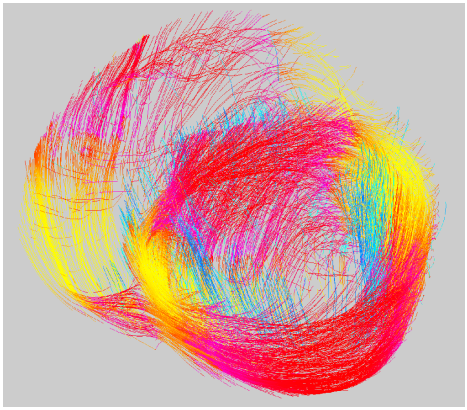


Figure 2.2: Heart fiber orientation in the human heart, obtained from diffusion tensor imaging

$$\mathbf{D} = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix} \tag{2.2.1}$$

The 6 independent values of the tensor elements vary continuously with the spatial location in the tissue.

Eigenvalues λ_i and eigenvectors $\mathbf{e}_i$ of the diffusion tensor (2.2.1) can be found as a solution to the eigenvalue problem:

$$\mathbf{D}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

Since the tensor is symmetric, its eigenvalues are always real numbers, and the eigenvectors are orthogonal and form a basis. Geometrically, a diffusion tensor can be thought of as an ellipsoid with its three axes oriented along these eigenvectors, with the three semiaxis lengths proportional to the square root of the eigenvalues of the tensor.

The heart fibers have strong linear diffusion and are oriented along the principal eigenvector, $\mathbf{e}_1$ [105, 80]. Therefore if we need fiber orientations for a specific subject, then computing the principal direction of the diffusion tensor is sufficient. However, *in vivo* acquisition of cardiac DTI is not possible using current scanners and acquisition protocols.

Consequently, we use an alternate strategy and map the diffusion tensors from a template image onto the subject. The procedure for warping DTIs from a template onto a subject is described in the following section.

2.3 Warping Diffusion Tensors from template to subjects

We have MR images for both the subject and the template. The diffusion tensor data is available only for the template. We use a very high dimensional elastic registration technique [86] to estimate the deformation that warps the template to the subject space. We use this deformation field to map the fibers from the template to the subject. It is more complicated to warp tensor fields than it is to warp scalar images. This is because the tensor must be reoriented on each image voxel, in addition to a voxel displacement that is implied by the deformation field. This is achieved by finding the rotational component of the deformation field. Clearly, the accuracy of this approach is bound to be limited and shall only provide a generic superposition of the template on the subject. However, until DTI technologies improve to a stage when *in vivo* cardiac acquisitions are possible, this is a viable alternative. We test the effectiveness of the tensor remapping algorithm by comparing the mapped tensors with the ground truth diffusion tensors for 19 canine datasets[1]. We demonstrate that the fiber orientation estimates obtained using the proposed method are more accurate compared with generic mathematical models of fiber orientation. The method of computing the transformation between the two geometries and the tensor reorientation algorithm are now described.

2.3.1 Deformable Image Registration

Image warping for deformable registration has received a great deal of attention during the past decade [121]. In the present work we used a very-high-dimensional elastic transformation procedure in 3D volume space, referred to as the hierarchical attribute matching

[1] CCBM, Johns Hopkins University

mechanism for elastic registration (HAMMER) method, which is determined from T1-weighted images and applied on the coregistered DT image of the template. This approach uses image attributes to determine point correspondences between an individual image and a template, which resides in the stereotaxic space and is the subject for which we have the diffusion tensors. A hierarchical sequence of piece-wise smooth transformations is then determined, so that the attributes of the warped images are as similar as possible to the attributes of the target. Relatively fewer, more stable attributes are used in the initial stages of this procedure, which helps avoid local minima, a known problem in high-dimensional transformations. The details of this algorithm can be found in [86].

2.3.2 Tensor Reorientation

It is a simple matter to warp a scalar image by a known spatial transformation. The image value from a particular voxel is transferred, via the displacement field of the spatial transformation, to a voxel in the target image. Typically, some sort of interpolation must also be applied. However, a more complex procedure is required to warp tensor fields, especially when the tensor estimates are noisy. We use an approach similar to that proposed by Xu et al [117].

If we know the direction, $\mathbf{v}$, of the fiber on voxel with coordinates $\mathbf{x}$, we can readily find the rotated version, $\mathbf{v}'$, of $\mathbf{v}$, according to the warping transformation. If $\mathbf{R}$ is the matrix that rotates $\mathbf{v}$ to $\mathbf{v}'$, then $\mathbf{R}$ should be applied to the respective tensor measurement. However, in practice we do not know $\mathbf{v}$. In fact, this is precisely what we would like to estimate. We only have a noisy orientation of v, which is the principal direction (PD) of the corresponding tensor measurement. One could use that PD in place of $\mathbf{v}$, as proposed in [3]. However, that makes the approach vulnerable to noise, since the PD is only a noisy observation, and could be quite different from the true underlying fiber orientation.

Assuming that we know the probability distribution function (PDF), $f(\mathbf{v})$, of the fiber direction $\mathbf{v}$, we can find the rotation matrix, $\tilde{\mathbf{R}}$ which minimizes the expected value of $\|\mathbf{v}' - \mathbf{R}\mathbf{v}\|^2$ over all orthonormal matrices $\mathbf{R}$:

$$\begin{aligned}\tilde{\mathbf{R}} &= \arg\min_{\mathbf{R}} E\{\|\mathbf{v}' - \mathbf{R}\mathbf{v}\|^2\} \\ &= \arg\min_{\mathbf{R}} \int_{\mathbf{v}} pdf(\mathbf{v})\|\mathbf{v}' - \mathbf{R}\mathbf{v}\|^2 d\mathbf{v}\end{aligned}$$

This problem can be solved by the Procrustean estimation [28], if a number of random samples, $\mathbf{v}$, are drawn from the PDF, and their respective rotated versions, $\mathbf{v}'$, are found by the rotation that the warping field applies to $\mathbf{v}$. If we arrange these vectors $\mathbf{v}'$ and $\mathbf{v}$ to form the columns of the matrices $\mathbf{A}$ and $\mathbf{B}$, respectively, then $\tilde{\mathbf{R}}$ is found by minimizing:

$$\|\mathbf{A} - \mathbf{R}\cdot\mathbf{B}\|_2^2 = \|\mathbf{A}\|_2^2 + \|\mathbf{B}\|_2^2 + 2\sum_i \sigma_i(\mathbf{A}\cdot\mathbf{B}^T)$$

where $\sigma_i(\mathbf{M})$ are the singular values of matrix $\mathbf{M}$. $\tilde{\mathbf{R}}$ can be determined via a singular value decomposition of $\mathbf{A}\cdot\mathbf{B}$.

$$\begin{aligned}\mathbf{A}\cdot\mathbf{B}^T &= \mathbf{V}\cdot\Sigma\cdot\mathbf{W}^T \\ \tilde{\mathbf{R}} &= \mathbf{V}\cdot\mathbf{W}^T\end{aligned}$$

More details on the algorithm and the estimation of the PDF can be found in [117].

2.4 Mechanical Modeling

We shall now describe the linear elastic formulation of the mechanical model of the heart. We model the heart as a linear elastic solid occupying a bounded region ω and assume that the normalized fiber orientations, $\mathbf{n}$, at the resting phase of the heart (end of diastole) are available.

We can write the forward problem as:

$$M\ddot{u}(t) + C\dot{u}(t) + Ku(t) + F(t) = 0 \quad t \in (0,1). \tag{2.4.1}$$

Using a Ritz-Galerkin formulation, with basis functions ϕ, the expressions for M and K are given by $M_{ij} = \int \mathrm{I}(\phi_i\phi_j)$, $K = \int(\lambda+\mu)\nabla\phi_i\otimes\nabla\phi_j + \mu\mathrm{I}(\nabla\phi_i\cdot\nabla\phi_j)$, and $C = \alpha M + \beta K$,

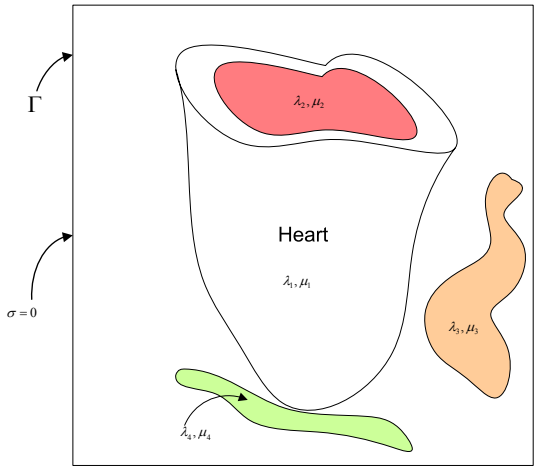


Figure 2.3: The material properties and boundary conditions for the cardiac model. We set the stress, $\sigma = 0$, at the boundary Γ. Different Lamé parameters are selected for the myocardium (λ_1, μ_1), blood (λ_2, μ_2), the lungs (λ_3, μ_3) and for bone (λ_4, μ_4)

with α and β viscous-damping parameters. Here, $\otimes$ is the outer vector product, λ and μ are the Lamé constants, and I is the 3D identity matrix. Equation (2.4.1) is derived by the Navier linear elastodynamics equation [34].

To solve (2.4.1) we embed ω in a regular domain Ω. We use trilinear finite elements to discretize (2.4.1), and piecewise constant functions for λ, ν. The Poisson ratio $\nu(\mathbf{x})$ varies between 0 for a fully compressible material to 0.5 for a fully incompressible material. When considering soft tissue deformations, the value of the Poisson ratio given for many tissue classes borders on the limit of incompressibility[2]. Gladilin [26] studies the sensitivity of the Poisson ratio on the displacement field obtained via an incompressible formulation and a compressible formulation. The results indicate that Poisson ratio does not affect the solution significantly.

We consider the input data as defined by the MR/DTI image to define the regular domain Ω. The boundary conditions are exactly prescribed on the six faces of the cuboid, Γ, are simply set to have zero stress, i.e., $\sigma = 0$. One important issue is the accurate integration of the elements that overlap in the inhomogeneity transition. For simplicity we have

[2] As ν approaches 0.5, commonly used displacement-based finite element implementations suffer from the so-called locking effect. We use underintegration for the $\nabla \cdot \mathbf{u}$ term in (2.4.1); see [39] for details.

Tissue Type	myocardium	blood	lungs	bone
Young's Modulus (kPa)	10	1	1	100
Poisson's ratio	0.45	0.35	0.35	0.45

Table 2.1: The material properties used for the different tissue types used in our cardiac model.

used voxelized material properties even in the case of exact geometry information. Within the regular domain Ω, different material properties are assigned based on the tissue type. We currently consider the myocardium (λ_1, μ_1), blood (λ_2, μ_2), the lungs (λ_3, μ_3) and bone (λ_4, μ_4). These are shown in Figure 2.3. The material properties for the different tissue types are listed in Table 2.1.

2.5 Linear Elastodynamics

We assume that the MR image occupies the region $\Omega \in \mathbb{R}^3$ in its reference state (end-diastole) at time $t = 0$. The boundary $\Gamma = \partial\Omega$ has the outward unit normal given by $\mathbf{n}$, the displacement vector is represented by $\mathbf{u}$, and the velocity by $\mathbf{v}$. The myocardium is assumed to be made of an elastic material and is subject to body force $\mathbf{f}$ per unit volume.

The strong form of the equations of linear elastodynamics are written as:

$$\begin{aligned} \rho\frac{\partial^2\mathbf{u}}{\partial t^2} &= \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \mathbf{f} && \text{in } \Omega\times]0,T[, \\ \boldsymbol{\sigma}\mathbf{n} &= 0, && \text{in } \Gamma\times]0,T[, \\ \mathbf{u}(t=0) &= \mathbf{u}_0, && \text{in } \Omega \\ \dot{\mathbf{u}}(t=0) &= \mathbf{v}_0, && \text{in } \Omega \end{aligned} \tag{2.5.1}$$

where $\mathbf{u}_0$ is the initial displacement, $\mathbf{v}_0$ is the initial velocity, $\boldsymbol{\sigma}$ is the stress tensor and $\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}$ denotes the divergence of $\boldsymbol{\sigma}$. Assuming that the material is isotropic and homogeneous, one may express the stress tensor as

$$\boldsymbol{\sigma} = \lambda \operatorname{tr} \boldsymbol{\epsilon}\mathbf{I} + 2\mu\boldsymbol{\epsilon}, \tag{2.5.2}$$

in terms of the Lamé constants λ and μ, the identity tensor $\mathbf{I}$, and the infinitesimal strain tensor $\boldsymbol{\epsilon}$. The strain is defined as

$$\boldsymbol{\epsilon} := \frac{1}{2}\left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^T\right] := \boldsymbol{\nabla}_s\mathbf{u}\,, \tag{2.5.3}$$

where $\boldsymbol{\nabla}\mathbf{u}$ is the gradient operator expessed in Cartesian component form as

$$[\boldsymbol{\nabla}\mathbf{u}] = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ u_{2,1} & u_{2,2} & u_{2,3} \\ u_{3,1} & u_{3,2} & u_{3,3} \end{bmatrix}.$$

Using equation (2.5.3), the components of the strain tensor are

$$[\boldsymbol{\epsilon}] = \begin{bmatrix} u_{1,1} & \frac{1}{2}(u_{1,2}+u_{2,1}) & \frac{1}{2}(u_{1,3}+u_{3,1}) \\ \frac{1}{2}(u_{1,2}+u_{2,1}) & u_{2,2} & \frac{1}{2}(u_{2,3}+u_{3,2}) \\ \frac{1}{2}(u_{1,3}+u_{3,1}) & \frac{1}{2}(u_{2,3}+u_{3,2}) & u_{3,3} \end{bmatrix}.$$

Let $\mathbf{w}$ denote the variations, and $\mathbf{w} \in \mathcal{V}$ the variation space. Then the weak form can be written as

$$\int_\Omega \mathbf{w}\cdot(\rho\frac{\partial^2\mathbf{u}}{\partial t^2} - \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} - \mathbf{f})\,d\Omega + \int_\Gamma \mathbf{w}\cdot\boldsymbol{\sigma}\mathbf{n}\,d\Gamma = 0. \tag{2.5.4}$$

Using the Einsteinian summation convention, we can write it as,

$$\begin{aligned}
0 &= \int_\Omega w_i(\rho u_{i,tt} - \sigma_{ij,j} - f_i)\,d\Omega + \int_\Gamma w_i\sigma_{ij}n_j\,d\Gamma \\
&= \int_\Omega w_i\rho_i u_{i,tt}\,d\Omega - \left(\int_\Omega w_i\sigma_{ij,j}\,d\Omega\right) - \int_\Omega w_i f_i\,d\Omega + \int_\Gamma w_i\sigma_{ij}n_j\,d\Gamma \\
&= \int_\Omega w_i\rho_i u_{i,tt}\,d\Omega - \left(\int_\Omega (w_i\sigma_{ij})_{,j}\,d\Omega - \int_\Omega w_{i,j}\sigma_{ij}\,d\Omega\right) - \\
&\quad \int_\Omega w_i f_i\,d\Omega + \int_\Gamma w_i\sigma_{ij}n_j\,d\Gamma \\
&= \int_\Omega w_i\rho_i u_{i,tt}\,d\Omega - \left(\int_\Gamma w_i\sigma_{ij}n_j\,d\Gamma - \int_\Omega w_{i,j}\sigma_{ij}\,d\Omega\right) - \\
&\quad \int_\Omega w_i f_i\,d\Omega + \int_\Gamma w_i\sigma_{ij}n_j\,d\Gamma \\
&= \int_\Omega w_i\rho_i u_{i,tt}\,d\Omega + \int_\Omega w_{(i,j)}\sigma_{ij}\,d\Omega - \int_\Omega w_i f_i\,d\Omega,
\end{aligned} \tag{2.5.5}$$

where use is made of integration by parts and the divergence theorem. It follows that,

$$\int_\Omega \mathbf{w} \cdot \rho \ddot{\mathbf{u}} \, d\Omega - \int_\Omega \boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma} \, d\Omega = \int_\Omega \mathbf{w} \cdot \mathbf{f} \, d\Omega \,, \tag{2.5.6}$$

where $\boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma}$ denotes the contraction of the tensors $\boldsymbol{\nabla}_s \mathbf{w}$ and $\boldsymbol{\sigma}$, expressed in component form as $\boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma} = w_{i,j}\sigma_{ij}$.

Equation (2.5.6) motivates us to define the following bilinear forms:

$$a(\mathbf{w}, \mathbf{u}) = -\int_\Omega \boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma} \, d\Omega, \tag{2.5.7}$$

$$(\mathbf{w}, \mathbf{f}) = \int_\Omega \mathbf{w} \cdot \mathbf{f} \, d\Omega, \text{ and} \tag{2.5.8}$$

$$(\mathbf{w}, \rho \ddot{\mathbf{u}}) = \int_\Omega \mathbf{w} \cdot \rho \ddot{\mathbf{u}} \, d\Omega. \tag{2.5.9}$$

The corresponding weak formulation can be written as:

Given $\mathbf{f}, \mathbf{u}_0$ and $\mathbf{v}_0$, find $\mathbf{u}(t) \in \mathcal{S}_t, t \in [0, T]$, such that for all $\mathbf{w} \in \mathcal{V}$, such that

$$\begin{aligned} (\mathbf{w}, \rho \ddot{\mathbf{u}}) + a(\mathbf{w}, \mathbf{u}) &= (\mathbf{w}, \mathbf{f}), \\ (\mathbf{w}, \rho \mathbf{u}(0)) &= (\mathbf{w}, \rho \mathbf{u}_0), \\ (\mathbf{w}, \rho \dot{\mathbf{u}}(0)) &= (\mathbf{w}, \rho \boldsymbol{v}_0). \end{aligned} \tag{2.5.10}$$

In order to simply the expressions, we express the components of tensorial quantities such as $\boldsymbol{\nabla}_s \mathbf{w}$ and $\boldsymbol{\sigma}$ in vector form. In particular we define the strain vector as,

$$\boldsymbol{\epsilon}(\mathbf{u}) = \begin{Bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{12} \\ 2\epsilon_{23} \\ 2\epsilon_{31} \end{Bmatrix} = \begin{Bmatrix} u_{1,1} \\ u_{2,2} \\ u_{3,3} \\ u_{2,3} + u_{3,2} \\ u_{1,3} + u_{3,1} \\ u_{1,2} + u_{2,1} \end{Bmatrix}$$

Likewise, the stress tensor can be written in vector form as,

$$\boldsymbol{\sigma} = \left\{ \begin{array}{c} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{31} \end{array} \right\}$$

The stress-strain law (2.5.2) can be written using the vector convention as

$$[\boldsymbol{\sigma}] = [\mathbf{D}][\boldsymbol{\epsilon}], \tag{2.5.11}$$

where $[\mathbf{D}]$ is a (6×6) elasticity matrix such that

$$\mathbf{D} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{2.5.12}$$

Since the matrix $[\mathbf{D}]$ is always symmetric, it follows that the integrand of the bilinear form in (2.5.7) can be written with the aid of (2.5.11) as

$$\boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma} = [\boldsymbol{\epsilon}(\mathbf{w})][\mathbf{D}][\boldsymbol{\epsilon}(\mathbf{u})] := \boldsymbol{\epsilon}(\mathbf{w}) \cdot \mathbf{D}\boldsymbol{\epsilon}(\mathbf{u}),$$

which shows that the bilinear form in (2.5.7) in indeed symmetric.

2.5.1 Semidiscrete Galerkin formulation of elastodynamics

Given $\mathbf{f}, \mathbf{u}_0$, and $\dot{\mathbf{u}}_0$, find $\mathbf{u}^h = \mathbf{v}^h + \boldsymbol{g}^h, \mathbf{u}^h(t) \in \mathcal{S}_t^h$, such that for all $\mathbf{w}^h \in \mathcal{V}^h$,

$$\begin{aligned} (\mathbf{w}^h, \rho \ddot{\boldsymbol{v}}^h) + a(\mathbf{w}^h, \boldsymbol{v}^h) &= (\mathbf{w}^h, f) - (\mathbf{w}^h, \rho \ddot{\boldsymbol{g}}^h) - a(\mathbf{w}^h, \boldsymbol{g}^h) \\ (\mathbf{w}^h, \rho \boldsymbol{v}^h(0)) &= (\mathbf{w}^h, \rho \mathbf{u}_0) - (\mathbf{w}^h, \rho \boldsymbol{g}^h(0)) \\ (\mathbf{w}^h, \rho \dot{\boldsymbol{v}}^h(0)) &= (\mathbf{w}^h, \rho \dot{\mathbf{u}}_0) - (\mathbf{w}^h, \rho \dot{\boldsymbol{g}}^h(0)) \end{aligned} \tag{2.5.13}$$

The representations of $\mathbf{w}^h, \boldsymbol{v}^h$ and $\boldsymbol{g}^h$ are given by

$$\begin{aligned}
\mathbf{w}^h(\boldsymbol{x},t) = w_i^h(\boldsymbol{x},t)\boldsymbol{e}_i &= \sum_{A\in\eta-\eta_{q_i}} N_A(\boldsymbol{x})c_{iA}(t)\boldsymbol{e}_i \\
\boldsymbol{v}^h(\boldsymbol{x},t) = v_i^h(\boldsymbol{x},t)\boldsymbol{e}_i &= \sum_{A\in\eta-\eta_{g_i}} N_A(\boldsymbol{x})d_{iA}(t)\boldsymbol{e}_i \\
\boldsymbol{g}^h(\boldsymbol{x},t) = g_i^h(\boldsymbol{x},t)\boldsymbol{e}_i &= \sum_{A\in\eta_{g_i}} N_A(\boldsymbol{x})g_{iA}(t)\boldsymbol{e}_i
\end{aligned} \tag{2.5.14}$$

Substituting (14) into (13) we get, (ignoring $\boldsymbol{e}_i$, and $\boldsymbol{e}_j$ for clarity,

$$\begin{aligned}
&\left(\sum_{A\in\eta_{\text{int}}} N_A(\boldsymbol{x})c_{iA}(t), \sum_{B\in\eta_{\text{int}}} N_B(\boldsymbol{x})\rho\ddot{d}_{jB}(t)\right) + \\
&a\left(\sum_{A\in\eta_{\text{int}}} N_A(\boldsymbol{x})c_{iA}(t), \sum_{B\in\eta_{\text{int}}} N_B(\boldsymbol{x})d_{iB}(t)\right) = \\
&\left(\sum_{A\in\eta_{\text{int}}} N_A(\boldsymbol{x})c_{iA}(t), \mathbf{f}\right) + \left(\sum_{A\in\eta_{\text{int}}} N_A(\boldsymbol{x})c_{iA}(t), \mathfrak{h}\right)_\Gamma - \\
&\left(\sum_{A\in\eta_{\text{int}}} N_A(\boldsymbol{x})c_{iA}(t), \sum_{B\in\eta_{q_i}} N_B\rho\ddot{g}_{iB}(t)\right) - \\
&a\left(\sum_{A\in\eta_{\text{int}}} N_A(\boldsymbol{x})c_{iA}(t), \sum_{B\in\eta_{q_i}} N_B g_{iB}(t)\right)
\end{aligned}$$

This gives us a set of $3\eta_{\text{int}} = 3(\eta - \eta_{q_i})$ equations, where η is the total number of nodes,

$$\begin{aligned}
&\sum_{j=1}^{n_{\text{dof}}}\sum_{B\in\eta_{\text{int}}} (N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)\rho\ddot{d}_{jB}(t) + \sum_{j=1}^{n_{\text{dof}}}\sum_{B\in\eta_{\text{int}}} a(N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)d_{jB}(t) \\
= \;& (N_A\boldsymbol{e}_i, \mathbf{f}) + (N_A\boldsymbol{e}_j, \mathfrak{h})_\Gamma - \sum_{j=1}^{n_{\text{dof}}}\sum_{B\in\eta_{q_j}} (N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)\rho\ddot{g}_{jB}(t) \\
- \;& \sum_{B\in\eta_{q_j}} a(N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)g_{jB}(t)
\end{aligned}$$

For the case of homogeneous dirichlet boundary conditions, $\boldsymbol{g} = 0$, and $\eta_{q_i} = 0$, therefore (10) reduces to a set of 3η equations in 3η unknowns,

$$\sum_{j=1}^{n_{\text{dof}}}\sum_{B\in\eta} (N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)\rho\ddot{d}_{jB}(t) + \sum_{j=1}^{n_{\text{dof}}}\sum_{B\in\eta} a(N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)d_{jB}(t) = (N_A\boldsymbol{e}_i, \mathbf{f}) + (N_A\boldsymbol{e}_i, \mathfrak{h})_\Gamma \tag{2.5.15}$$

In order to derive the mass matrix, the global stiffness matrix and the force vector, we need to specify the global ordering of equations. This shall be explained in detail later, for now we assume we have a function **id(i,A)** that takes the degree of freedom and the global node number as input and returns the global equation number. Using this we can write the **matrix problem** as:

where the mass matrix,

$$\boldsymbol{M} = \bigwedge_{e=1}^{n_{\text{el}}} (\boldsymbol{m}^e)$$

here, $\bigwedge$ is the matrix assembly operator, and the elemental mass matrix, $\boldsymbol{m}^e$ is given in terms of nodal submatrices as,

$$\begin{aligned} \boldsymbol{m}^e &= \left[m^e_{pq}\right] \\ m^e_{pq} &= \delta_{ij} \int_{\Omega_e} N_a \rho N_b d\Omega \end{aligned} \tag{2.5.16}$$

Notation 1 *In all these definitions, n_{en} is the number of element nodes, which for the trilinear hexahedral element is 8; n_{ed} is the number of element degrees of freedom per node, which is 3 in our case. Also n_{ee} stands for the number of element equations, which is $n_{\text{ed}} n_{\text{en}}$.*

Notation 2 *For the elemental mass matrix, $1 \leqslant p, q \leqslant n_{\text{ee}} = n_{\text{ed}} n_{\text{en}}$. Therefore in our case the elemental mass matrix shall be 24×24. For the nodal submatrices, indices $p = n_{\text{ed}}(a-1) + i$, and $q = n_{\text{ed}}(b-1) + j$.*

Similarly, the stiffness matrix can be written as,

$$\begin{aligned} \boldsymbol{K} &= \bigwedge_{e=1}^{n_{\text{el}}} (\boldsymbol{k}^e) \\ \boldsymbol{k}^e &= \left[k^e_{pq}\right] \\ k^e_{pq} &= \boldsymbol{e}_i^T \int_{\Omega_e} \boldsymbol{B}_a^T \boldsymbol{D} \boldsymbol{B}_a d\Omega \boldsymbol{e}_j \end{aligned} \tag{2.5.17}$$

where the matrices $\boldsymbol{D}$ was defined earlier (7) and $\boldsymbol{B}_a$ is given by,

$$\boldsymbol{B}_a = \begin{bmatrix} N_{a,x} & 0 & 0 \\ 0 & N_{a,y} & 0 \\ 0 & 0 & N_{a,z} \\ 0 & N_{a,z} & N_{a,y} \\ N_{a,z} & 0 & N_{a,x} \\ N_{a,y} & N_{a,x} & 0 \end{bmatrix}$$

We can now proceed to write the right hand side of our equation, i.e., the force vector. This can be written as,

$$\begin{aligned} \mathbf{f}(t) &= \mathbf{f}_{\text{nodal}}(t) + \bigwedge_{e=1}^{n_{\text{el}}} (\mathbf{f}^e(t)) \\ \mathbf{f}^e &= \{f_p^e\} \\ f_p^e &= \int_{\Omega_e} N_a f_i d\Omega + \int_{\Gamma_{\mathfrak{h}_i}} N_a \mathfrak{h}_i d\Gamma \end{aligned} \tag{2.5.18}$$

Also, we get the expressions for the initial conditions as,

$$\begin{aligned} \boldsymbol{d}_0 &= \boldsymbol{M}^{-1} \bigwedge_{e=1}^{n_{\text{el}}} (\hat{\boldsymbol{d}}^e) \\ \hat{\boldsymbol{d}}^e &= \{\hat{d}_p^e\} \\ \hat{d}_p^e &= \int_{\Omega_e} N_a \rho u_{0i} d\Omega \\ \dot{\boldsymbol{d}}_0 &= \boldsymbol{M}^{-1} \bigwedge_{e=1}^{n_{\text{el}}} (\widehat{\dot{\boldsymbol{d}}^e}) \\ \widehat{\dot{\boldsymbol{d}}^e} &= \{\widetilde{\dot{d}}_p^e\} \\ \widetilde{\dot{d}}_p^e &= \int_{\Omega_e} N_a \rho \dot{u}_{0i} d\Omega \end{aligned}$$

Of course under the assumption that the heart is stationary at the end of diastole, which is our reference frame, $u_{0i} = \dot{u_{0i}} = 0$ and therefore $\boldsymbol{d}_0 = \dot{\boldsymbol{d}}_0 = \mathbf{0}$.

2.5.2 Solving the forward problem: Newmark Scheme

In order to solve the forward problem, we use the Newmark method. We use the average acceleration or trapezoidal rule, which uses the values $\beta = 1/4$ and $\gamma = 1/2$. We first define

the predictors,

$$\begin{aligned} \tilde{\boldsymbol{d}}_{n+1} &= \boldsymbol{d}_n + \Delta t \boldsymbol{v}_n + \frac{\Delta t^2}{4}\boldsymbol{a}_n \\ \tilde{\boldsymbol{v}}_{n+1} &= \boldsymbol{v}_n + \frac{\Delta t}{2}\boldsymbol{a}_n \end{aligned} \tag{2.5.19}$$

We can determine the acceleration at the next time instant by solving,

$$\left(\boldsymbol{M} + \frac{\Delta t^2}{2}\boldsymbol{K}\right)\boldsymbol{a}_{n+1} = \mathbf{f}_{n+1} - \boldsymbol{K}\tilde{\boldsymbol{d}}_{n+1} \tag{2.5.20}$$

We solve this to obtain $\boldsymbol{a}_{n+1}$, and then obtain the values of $\boldsymbol{d}_{n+1}$ and $\boldsymbol{v}_{n+1}$ using the correctors

$$\begin{aligned} \boldsymbol{d}_{n+1} &= \tilde{\boldsymbol{d}}_{n+1} + \frac{\Delta t^2}{4}\boldsymbol{a}_{n+1} \\ \boldsymbol{v}_{n+1} &= \tilde{\boldsymbol{v}}_{n+1} + \frac{\Delta t}{2}\boldsymbol{a}_{n+1} \end{aligned} \tag{2.5.21}$$

2.6 Active Force Models

We consider two models for the active forces: the first being the generic case where arbitrary forces can be specified and the second model which accounts for the contractility of the myocytes. For the generic case the force term is given by,

$$F(t) = M\mathbf{f}(t), \tag{2.6.1}$$

where, $\boldsymbol{f}(t)$ are the nodal forces at time t.

To model the contractility of the myocytes given the fiber contractility $\boldsymbol{s}$ as a function of space and time and the myocyte orientation n, we define the active stretch tensor $U = 1 + \boldsymbol{s}\, n \otimes n$, whose divergence results in a distributed active force of the form $\mathrm{div}(\boldsymbol{s}\, n \otimes n)$. The force term in this case is given by,

$$F(t) = A\boldsymbol{s}(t), \quad A_{ij} = \int (n \otimes n)\nabla\phi_i\phi_j. \tag{2.6.2}$$

To reduce the computational cost for the calculations, we used a reduced-order model for $\boldsymbol{s}$ and $\boldsymbol{f}$ as a combination of B-splines bases in time and radial basis functions in space

(Gaussians). The forces can then be written in terms of the B-spline basis, B and the radial basis, G, and the control parameters $\boldsymbol{\mu}$,

$$\boldsymbol{f}(\boldsymbol{x},t) = \sum_{k=1}^{3} \boldsymbol{e}_k \sum_i G_i^k(\boldsymbol{x}) \sum_j \boldsymbol{\mu}_{ij} B_j(t), \tag{2.6.3}$$

$$\boldsymbol{s}(\boldsymbol{x},t) = \sum_i G_i(\boldsymbol{x}) \sum_j \boldsymbol{\mu}_{ij} B_j(t). \tag{2.6.4}$$

In the matrix form this defines the parametrization matrix C is given by,

$$C_{xt,ij} = G_i(x) B_j(t). \tag{2.6.5}$$

We can write the active forces in terms of the parametrization matrix C, as

$$F = M\boldsymbol{f} = MC\boldsymbol{\mu},$$

$$F = A\boldsymbol{s} = AC\boldsymbol{\mu}.$$

2.7 Validation

In this section we present results from the experiments that were conducted to validate the fiber estimation and the numerical correctness of the mechanical model.

2.7.1 Validation of the Fiber Estimation Algorithm

We used canine DTI datasets obtained from Center for Cardiovascular Bioinformatics and Modeling, Johns Hopkins University and acquired at the National Institue of Health, to validate the effectiveness of our diffusion tensor remapping algorithm. A total of 19 canine subjects were scanned, of which 12 subjects were normal, and 7 had cardiac failure. The scans were performed *in vitro* after the hearts were harvested. Each heart was placed in an acrylic container filled with Fomblin, a perfluoropolyether (Ausimon, Thorofare, NJ). Fomblin has a low dielectric effect and minimal MR signal, thereby increasing contrast and

eliminating unwanted susceptibility artifacts near the boundaries of the heart. The long axis of the hearts was aligned with the z axis of the scanner. Images were acquired with a 4-element knee phased array coil on a 1.5 T GE CV/I MRI Scanner (GE, Medical System, Wausheka, WI) using an enhanced gradient system with 40 mT/m maximum gradient amplitude and a 150 T/m/s slew rate.

Since only the long axis of the heart was aligned with the z axis of the scanner, we first need to correct for rotation about the z axis. We picked one of the normal canine hearts as the template, and performed affine registration to warp the remaining subjects onto the template space. The diffusion tensors for these subjects were also rotated by the rotational component of the affine transform. We then perform the elastic registration using HAMMER to estimate the transformation that maps the template to the subject space. This transformation is then used to warp and reorient the diffusion tensors of template onto the subject. The quality of the mapping is measured by computing the angle between the principal direction of the mapped tensor and the principal direction of the ground truth obtained from the subject's DTI. This is shown in Figure 2.4.

The error in the fiber orientations is further demonstrated on one slice in Figure 2.5, by comparing the original fibers (in red) and the mapped fibers (in blue). Our method was able to successfully map the fibers for healthy as well as for failing canine hearts. The percentage of voxels where the error in the principal directions is less than 10° is shown in Figure 2.6. We evaluate an error of less than 10° since it is close to the average error obtained from DTI imaging [80] and histological measurements[94]. We also compare this with the error by using a synthetic model to estimate fiber orientations. The synthetic fiber orientations were produced by varying the elevation angle between the short axis plane and the fiber between $+90^\circ$ and -90° from the endocardium to the epicardium. Similar synthetic models have been used in [76, 84].

We computed the average error in the estimation of the fiber orientation for both normal and subjects with induced cardiac failure. The average error and the standard deviation in for the normal subjects is shown in Figure 2.7 and Figure 2.8. Similarly, the average error and the standard deviation in for the subjects with induced cardiac failure is shown in Figure

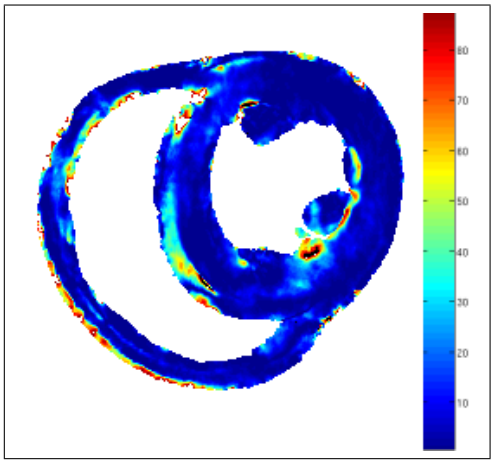


Figure 2.4: The angle between the mapped principal direction with the actual principal direction, in degrees. The image is overlaid on a segmentation of the heart based on the fractional anisotropy

2.9 and Figure 2.10.

The method was able to map the fibers accurately for both healthy as well as for failing hearts that had left bundle branch blocks. Although, the percentage of voxels having an error of less than 10° was lower in the failing hearts as compared to the healthy ones, we observe that most of these errors are in the vicinity of the block as expected, and the fiber orientations away from the bundle branch block were similar to that observed in healthy patients. Therefore, our method should perform better for other pathologies like ventricular hypertropy and infarction, since it has been shown that the muscle fiber orientations are not affected significantly by hypertrophy and myocardial infarction[111].

2.7.2 Synthetic Models for the validation of the forward solver

In order to validate the numerical accuracy and correctness of the forward, we used two synthetic models for validation. The first is a gaussian distribution of the forces with homogeneous material properties. The material properties are uniform over the entire domain and we choose a material withPoisson's ration $\nu = 0.45$ and Young's modulus 1 kPa. There are no fiber orientations in this model, and we work with full forces (2.6.1). We define

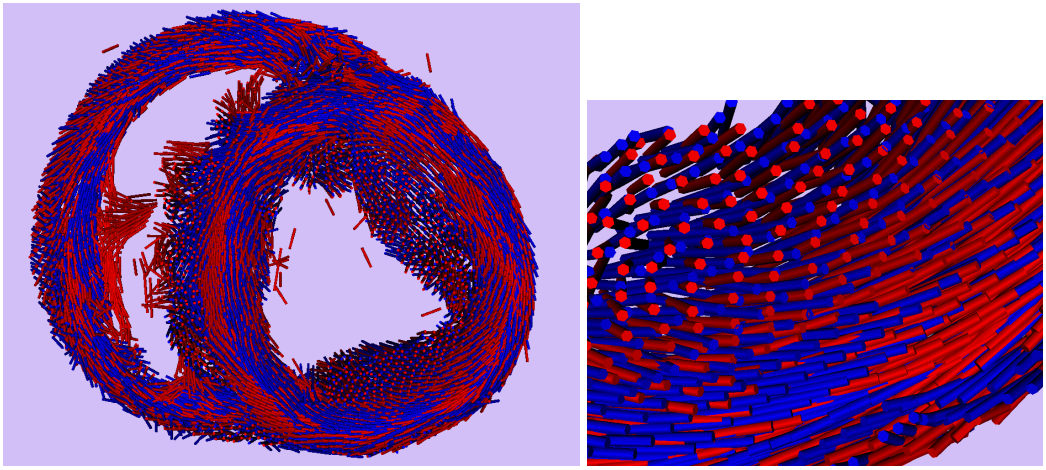


Figure 2.5: The principal directions of the original DT are shown in blue. The mapped principal directions are shown in red. The glyphs are overlaid on a segmentation of the heart based on the fractional anisotropy of the mapped DT

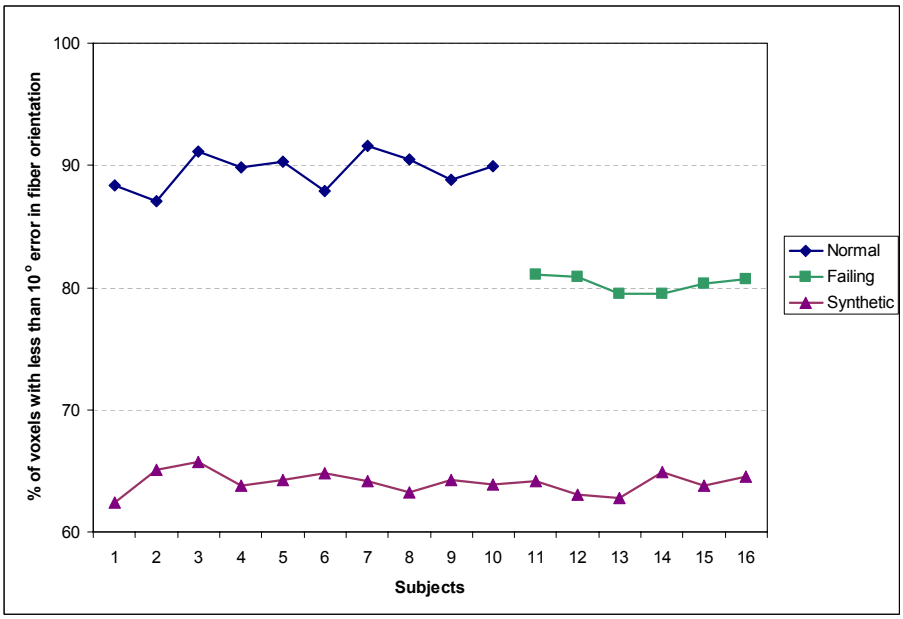


Figure 2.6: The percentage of voxels having less than 10° error in the principal directions after mapping

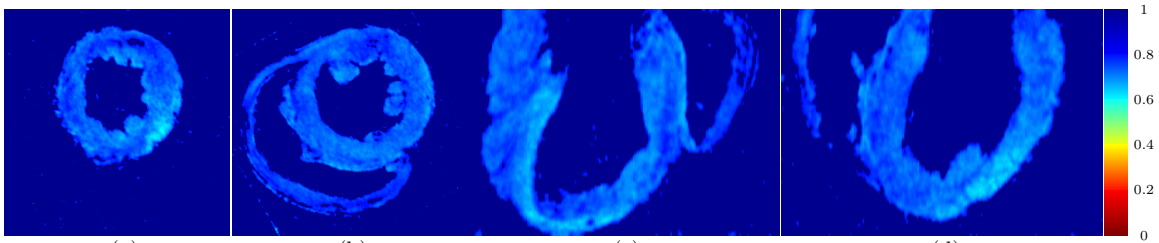


Figure 2.7: The projection of the estimated fiber orientation on the ground truth fiber estimation averaged over all normal subjects. Here a score of 1 implies a perfect alignment, whereas a score of 0 means that the estimated fiber orientation is perpendicular to the ground truth fiber orientation. Two short axis planes are shown in **(a)** and **(b)** and two long-axis views are shown in **(c)** and **(d)**.

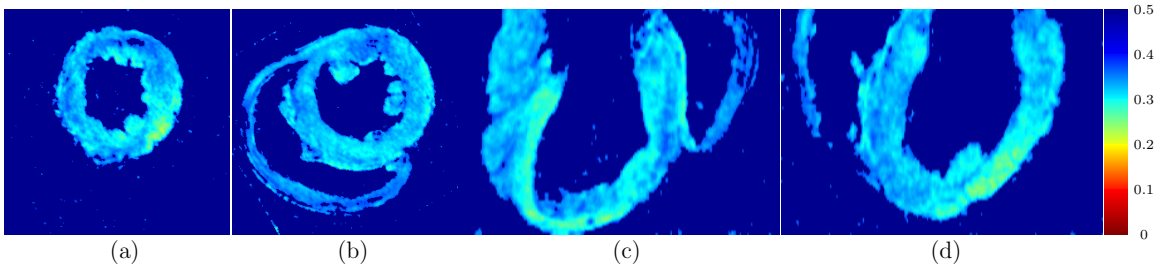


Figure 2.8: The standard deviation of the average projection of the estimated fiber orientation on the ground truth fiber estimation over all normal subjects. Two short axis planes are shown in **(a)** and **(b)** and two long-axis views are shown in **(c)** and **(d)**.

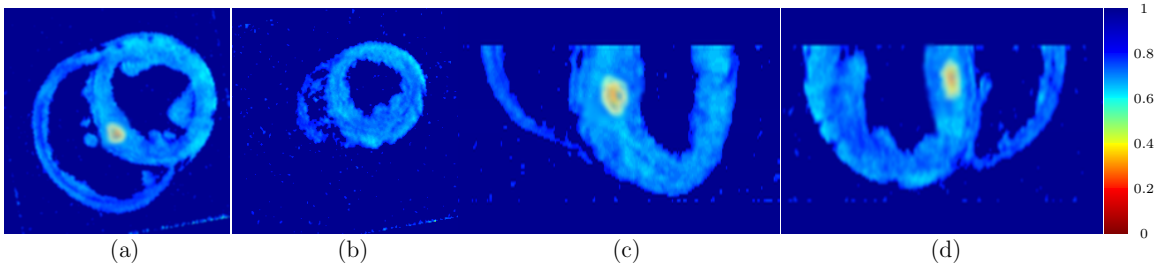


Figure 2.9: The projection of the estimated fiber orientation on the ground truth fiber estimation averaged over all subjects with induced cardiac failure. Here a score of 1 implies a perfect alignment, whereas a score of 0 means that the estimated fiber orientation is perpendicular to the ground truth fiber orientation. Two short axis planes are shown in **(a)** and **(b)** and two long-axis views are shown in **(c)** and **(d)**.

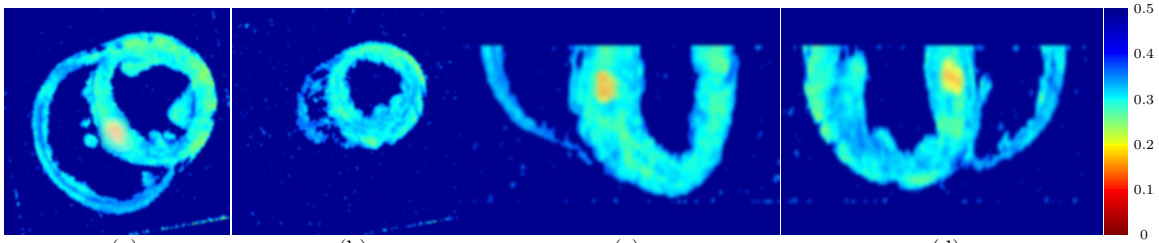


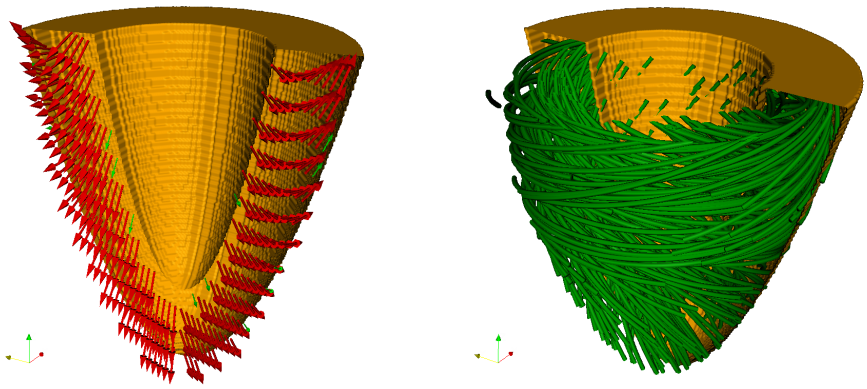
Figure 2.10: The standard deviation of the average projection of the estimated fiber orientation on the ground truth fiber estimation over all subjects with induced cardiac failure. Two short axis planes are shown in **(a)** and **(b)** and two long-axis views are shown in **(c)** and **(d)**.

forces over this domain as,

$$f_x = f_y = f_z = sin(kt)e^{-(x^2+y^2+z^2)}. \tag{2.7.1}$$

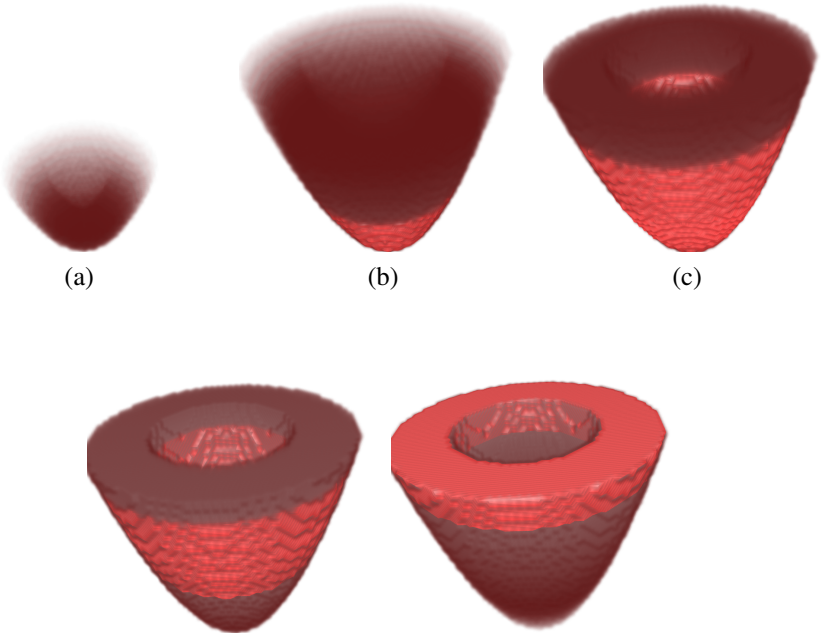
In order to assess the fiber-orientation parametrized model of the forces (2.6.2), we use an ellipsoidal model of the left ventricle. The fiber orientations are generated by varying the elevation angle between the fiber and the short axis plane between $+60°$ and $-60°$ from the endocardium to the epicardium [76, 84]. The model along with the fiber orientations is shown in Figure 2.11. For this model we selected a Poisson's ratio $\nu = 0.45$ and a Young's modulus of 10 kPa for the myocardial tissue and 1 kPa for the surrounding tissue and ventricular cavity. Raleigh damping ($C = \alpha M + \beta K$) was used with parameters $\alpha = 0$ and $beta = 7.5 \times 10^{-4}$. In order to drive the forward model, we generated forces by propagating a synthetic activation wave from the apex to the base of the ventricles. Snapshots of this activation wave at different phases of the cardiac cycle are shown in Figure 2.12.

The octree is generated using a gaussian distribution of points for the first case and using the geometry for the synthetic LV model. The balanced octrees for both models are shown in Figure 2.13. In both cases the smallest element size was set to be the same size as the regular grid counterpart. The number of octants for different grid sizes for both the models is listed in Table 2.2.



(a) (b)

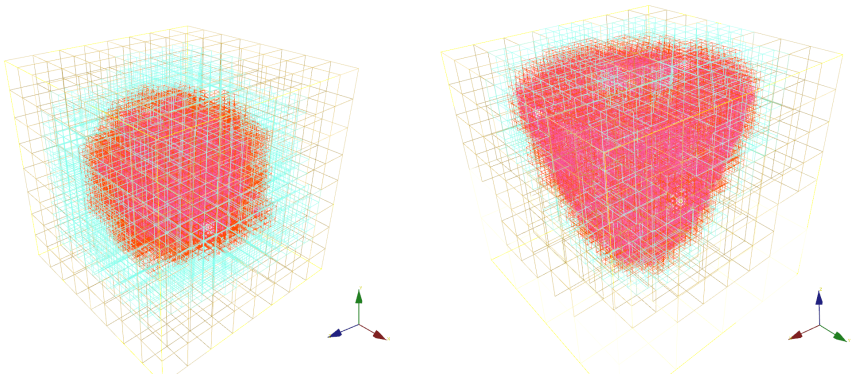
Figure 2.11: The ellipsoidal LV model shown with (**a**) the fiber orientations, and (**b**) the fibers obtained via fiber tracking.



(a) (b) (c)

(d) (e)

Figure 2.12: The activation function used in the synthetic model of the heart. The activation function starts at the apex and moves to the base of the ventricles.



(a) (b)

Figure 2.13: The octrees corresponding to (**a**) the gaussian force model, and (**b**) the ellipsoidal fiber force model.

2.7.3 Numerical Verification of the Forward Solver

We use the gaussian model to study the convergence of the forward solver by comparing the results obtained from the numerical solver with the analytical solution. The solution was obtained for discretizations corresponding to regular grids 32^3, 64^3, 128, and 256^3. The number of time steps were equal to the number of elements (along a axis) in all cases. The relative residual tolerance for the CG solver was set to 10^{-16} for the gaussian model. The time steps were again equal to the number of elements and the relative residual tolerance for the CG solver was set to 10^{-8} For both models the use of octrees for meshing reduced the number of elements significantly. We observed $O(h^2)$ convergence, where h is the grid size. These results, along with the number of processors used and the time for the computation are tabulated in Table 2.2.

The second model used was constructed from a MR image of a human heart. The fiber orientation was obtained from ex-vivo Diffusion Tensor (DTI) images of the heart. In order to drive the forward model, we generated forces by propagating a synthetic activation wave from the apex to the base of the ventricles. Snapshots of this activation wave at different phases of the cardiac cycle are shown in Figure 2.14. The fiber orientation, the myocardial forces and the resulting deformations within the myocardium are shown at different slices

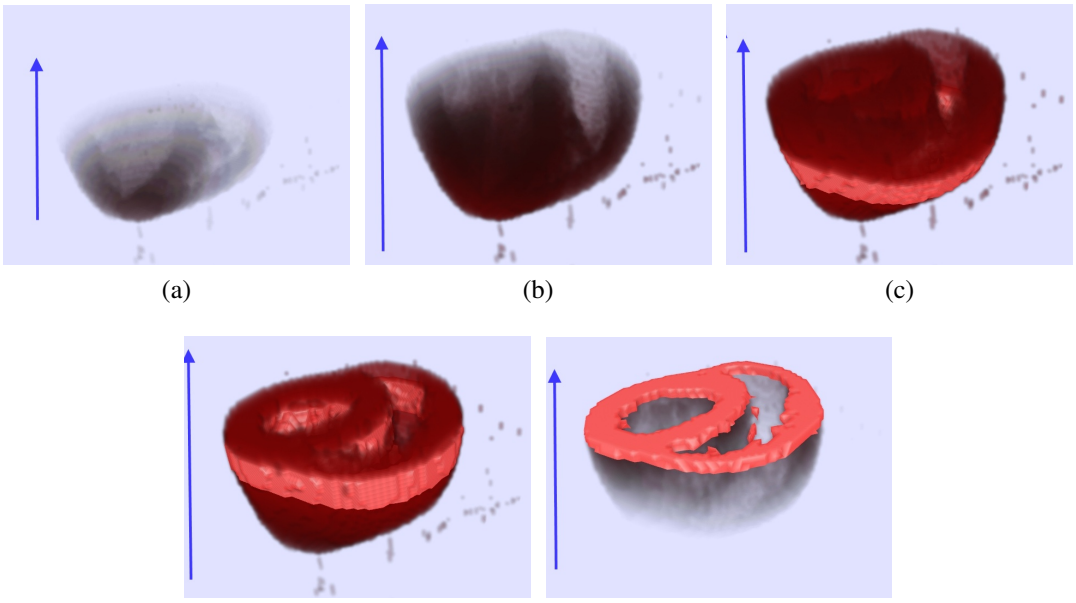
Problem Size	Number of Processors	Full Force				Fiber Force		
		#Octants	#Iters	Time	Error	#Octants	#Iters	Time
32^3	1	7386	5	87ms	7.63×10^{-2}	5958	19	291ms
64^3	8	52K	6	126ms	1.91×10^{-2}	24K	21	323ms
128^3	64	396K	8	186ms	4.77×10^{-3}	100K	26	383ms
256^3	512	2.8M	11	244ms	1.21×10^{-3}	456K	31	443ms

Table 2.2: Results from the validation of the forward solver for the two models. The regular grid size and the number of processors that the problem was run on are listed. We also list the number of octants in the resulting balanced Octree. The average number of KSP iterations and wall clock time per time step are listed. The relative error($\|\cdot\|_2$)are also provided. For the gaussian force, the relative error is computed with respect to the analytical solution.

and time points in Figures 2.15 and 2.16.

2.8 Conclusions

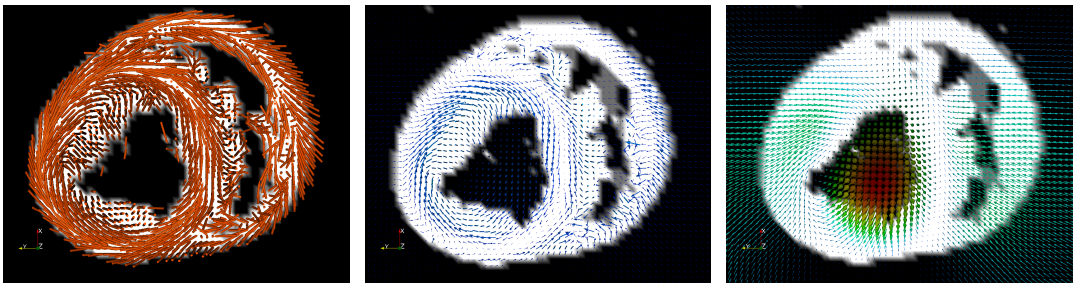
In this chapter we introduced algorithms for estimating myocardial fiber orientations and described the linear elastic bio-mechanical model of the heart that we shall use for constraining the motion estimation problem.



(a) (b) (c)

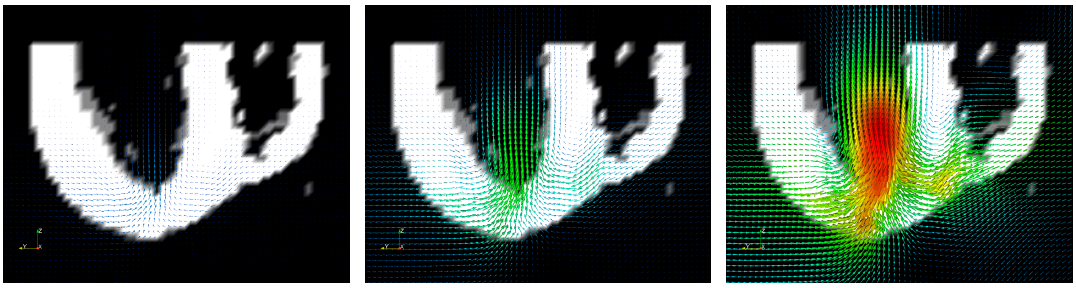
(d) (e)

Figure 2.14: The activation function used in the synthetic model of the heart. The activation function starts at the apex and moves to the base of the ventricles.



(a) (b) (c)

Figure 2.15: (a) The orientation of the cardiac fibers, (b) the forces developed within the myocardium as a result of the contraction of the fibers, and (c) the resulting deformations within the myocardium.



(a) (b) (c)

Figure 2.16: The deformations induced by the propagation of the activation from the apex to the base.

Chapter 3

Octree Meshing

Spatial decompositions of the d-dimensional cube have important applications in scientific computing: they can be used as algorithmic foundations for adaptive finite element methods [7, 43], adaptive mesh refinement methods [33, 69], and many-body algorithms [35, 101, 114, 118, 119]. *Quadtrees* [22] and *octrees* [58] are hierarchical data structures commonly used for partitioning 2D and 3D domains, respectively; they use axis aligned lines and planes, respectively. These tree data structures have been in use for over three decades now [22, 77]. However, design and use of large scale distributed tree data structures that scale to thousands of processors is still a major challenge and is an area of active research even today [9, 16, 23, 33, 35, 101, 108, 114, 115, 118, 119].

Octrees and quadtrees are usually employed while solving the following two types of problems.

- *Searching:* Searches within a domain using d-trees (d-dimensional trees with a maximum of 2^d children per node), benefit from the reduction of the complexity of the search from $O(n)$ to $O(\log n)$ [24, 61].

- *Spatial decomposition:* Unstructured meshes are often preferred over uniform discretizations because they can be used with complicated domains and permit rapid grading from coarse to fine elements. However, generating large unstructured meshes is a challenging task [88]. On the contrary, octree-based unstructured hexahedral meshes can be constructed efficiently [10, 31, 78, 79, 87, 107]. Although they are

not suitable for highly complicated geometries, they provide a good compromise between adaptivity and simplicity for numerous applications like solid modeling [58], object representation [5, 13], visualization [23], image segmentation [93], adaptive mesh refinement [33, 69], and N-body simulations [35, 101, 114, 118, 119].

Octree data structures used in discretizations of partial differential equations should satisfy certain spatial distribution of octant size [9, 108]. That is, there is a restriction on the relative sizes of adjacent octants.[1] Furthermore, conforming discretizations require the 'balance condition' to construct appropriate function spaces.In particular, when the 2:1 balance constraint is imposed on octree-based hexahedral meshes, it ensures that there is at most one *hanging* node on any edge or face. What makes the balance-refinement problem difficult and interesting is a property known as the *ripple effect*: An octant can trigger a sequence of splits whereby it can force an octant to split, even if it is not in its immediate neighborhood. Hence, balance-refinement is a non-local and inherently iterative process. Solving the balance-refinement problem in parallel, introduces further challenges in terms of synchronization and communication since the ripple can propagate across multiple processors.

Related Work. Limited work has been done on large scale parallel construction [35, 115, 119] and balance refinement [43, 108] of octrees, and the best known algorithms exhibit suboptimal isogranular scalability. The key component in constructing octrees is the partitioning of the input in order to achieve good load balancing. The use of space-filling curves for partitioning data has been quite popular [35, 108, 115, 119]. The proximity preserving property of space-filling curves makes them attractive for data partitioning. All of the existing algorithms for constructing octrees use a top-down approach after the initial partition. The major hurdle in using a parallel top-down approach is avoiding overlaps. This typically requires some synchronization after constructing a portion of the tree [108, 115, 119]. Section 3.2.3 describes the issues that arise in using a parallel top-down approach.

[1] This is referred to as the balance constraint. A formal definition of this constraint is given in section 3.1.4.

Bern et al. [9] proposed an algorithm for constructing and balancing quadtrees for EREW PRAM architectures. However, it cannot be easily adapted for distributed architectures. In addition, the balanced quadtree produced is suboptimal and can have up to 4 times as many cells as the optimal balanced quadtree. Tu et al. [108] propose a more promising approach, which was evaluated on large octrees. They construct and balance 1.22B octants for the Greater Los Angeles basin dataset [52] on 2000 processors in about 300 seconds. This experiment was performed on the TCS-1 terascale computing HP AlphaServer Cluster at the Pittsburgh Supercomputing Center. In contrast, we construct and balance[2] 1B octants (approximately) for three different point distributions (Gaussian, Log-Normal and Uniform) on 1024 processors on the same cluster in about 60 seconds.

There is a large literature on large-scale finite element (FEM) calculations. Here we limit our review to recent work that has scaled to thousands of processors. One of the largest calculations was reported in [8]. The authors proposed a scheme for conforming discretizations and multigrid solvers on semi-structured meshes. Their approach is highly scalable for nearly structured meshes and for constant coefficient PDEs. However, it does not support adaptive discretizations, and its computational efficiency diminishes in the case of variable coefficient operators. Examples of scalable approaches for unstructured meshes include [1] and [55]. In those works multigrid approaches for general elliptic operators were proposed. The associated constants for constructing the mesh and performing the calculations however, are quite large: setting up the mesh and building the associated linear operators can take thousands of seconds. A significant part of CPU time is related to the multigrid scheme (we do not consider multigrid in this thesis); even in the single grid cases however, the run times are quite large.

The high-costs related to partitioning, setup, and accessing generic unstructured grids, has motivated the design of octree-based data structures. Such constructions have been used in sequential and modestly parallel adaptive finite element implementations [7, 33, 43, 69], and many-body algorithms [32, 35, 118, 119]. State-of-the-art in-core and out-of-core

[2] While we enforce the 0-*balance* constraint, [108] only enforce the 1-*balance* constraint. Note that it is harder to 0-*balance* a given octree. See section 3.1.4 for more details on the different balance constraints.

implementations for finite elements can be found in [2, 106, 107, 108, 109]. In these papers the authors propose a top-down octree construction resulting in a Morton-ordered tree, followed by a 2:1 balance constraint using a novel algorithm (the parallel prioritized ripple propagation). Once the tree is constructed, a finite-element framework is built on top of the tree structure. Evaluating the discrete Laplacian operators (a matrix-free matrix-vector multiplication) requires three tree traversals, one traversal over the elements, and two traversals to apply projection over the so-called "hanging nodes", to enforce a conforming discretization [43].

Contributions. Our goal is to significantly improve the construction, balancing, and discretization performance of octree-based algorithms. We present novel parallel algorithms to construct complete linear octrees from a list of points; and one to enforce an optimal 2:1 balance constraint[3] on complete linear octrees. We use a linear octree Morton-encoding-based representation. Given a set of points, partitioned across processors, we create a set of octants that we sort and repartition using the Morton ordering. A complete linear octree is constructed using the seed octants. Then, we build an additional auxiliary list of a small number of coarse octants or *blocks*. This auxiliary octant set encapsulates and compresses the local spatial distribution of octants; it is used to accelerate the 2:1-balance refinement, which we implement using a hybrid strategy: intra-block balancing is performed by a classical level-by-level balancing/duplicate-removal scheme; and inter-block balancing is performed by a variant of the ripple-propagation algorithm proposed in [108]. The main parallel tools used are sample sorts (accelerated by biotonic sorts), and standard point-to-point/collective communication calls.[4] We build upon these algorithms and develop efficient data structures that support trilinear conforming finite element calculations on linear octrees. We focus on reducing (1) the time to build these data structures; (2) the memory overhead associated in storing them; and (3) the time to perform finite element calculations using these data structures.

[3] There exists a unique least common balance refinement for a given octree [60].

[4] When we discuss communication costs, we assume a Hypercube network topology with $\Theta(n_p)$ Bisection Width.

We avoid using multiple passes (projections) to enforce conformity; instead, we perform a single traversal by mapping each octant/element to one of eight pre-computed element types, depending on the configuration of hanging nodes for that element. Our data structure does not allow efficient random queries in the octree, but such access patterns are not necessary for FEM calculations.

The memory overhead associated with unstructured meshes arises from the need to store the element connectivity information. In regular grids such connectivity is not necessary as the indexing is explicit. For general unstructured meshes of trilinear elements one has to directly store eight indices defining the element nodes. In octrees we still need to store this information, but it turns out that instead of storing eight integers (32 `bytes`), we need to store only 12 `bytes`. We use the Golomb-Rice encoding scheme to compress the element connectivity information and to represent it as a Uniquely Decodable Code (UDC) [46]. In addition, the linear octree is stored in a compressed form that requires only one byte per octant (the level of the octant).

Finally, we employ overlapping of communication and computation to efficiently handle octants shared by several processors or *"ghost"* octants.[5] In addition, the Morton-ordering offers reasonably good memory locality. The cost of applying the discretized Laplacian operator using our approach is comparable to that of a discretization on a regular grid (without stored indexing). Thus, our approach offers significant savings (over structured grids and general unstructured grids) in the case of non-uniform discretizations.

The main parallel cost of the algorithm is that related to the parallel sorts that run in $O(N \log N)$ work and $O(\frac{N}{n_p} \log(\frac{N}{n_p}) + n_p \log(n_p))$ time, assuming uniformly distributed points [29]. In the following sections we present several algorithms for which we give precise `work` and `storage` complexity. For some of the parallel algorithms we also give `time` complexity estimates; this corresponds to wall-clock time and includes work/per processor and communication costs. The precise number depends on the initial distribution and the

[5] Every octant is owned by a single processor. However, the values of unknowns associated with octants on interpocessor boundaries need to be shared among several processors. We keep multiple copies of the information related to these octants and we term them "ghost" octants.

effectiveness of the partitioning. Thus the numbers for `time` are only an estimate under uniform distribution assumptions. If the `time` complexity is not specifically mentioned then it is comparable to that of a sample-sort.

Our algorithm has scaled to four billion octants on 4096 processors on a Cray XT3 ("Big Ben") at the Pittsburgh Supercomputing Center. The overall time for octree construction, balancing, and meshing is slightly over a minute; a second-order accurate evaluation of the discrete Laplacian takes only a few seconds. Our experiments demonstrate that the algorithms proposed as part of this thesis achieve a significantly lower running time compared to previous implementations.

Organization of this chapter. In Section 3.1 we introduce some terminology that will be used in the rest of the chapter. In Section 3.2, we describe the various components of our construction and balance refinement algorithms. Tables 3.1 and 3.2 summarize the notation that is used in the subsequent sections. In Section 3.3 we describe octree meshing, and the octree and mesh compression. In Section 3.4 we describe how we perform the finite element computation. In Section 3.7 we present performance results, including fixed size and isogranular scalability tests on different data distributions, that demonstrate the efficiency of our implementation. Finally, in Section 3.8, shortcomings of the proposed approach are discussed and some suggestions for future work are also offered.

3.1 Background

An octree is a tree data structure in which every node has a maximum of eight children. Octrees are analogous to binary trees (maximum of two children per node) in 1-D and quadtrees (maximum of four children per node) in 2-D. A node with no children is called a *leaf* and a node with one or more children is called an *interior node*. The only node with no parent is the *root* and all other nodes have exactly one parent. Nodes that have the same parent are called *siblings*. A node's children, grandchildren and so on and so forth are collectively referred to as the node's *descendants* and this node will be an *ancestor* of

Table 3.1: Symbols for terms

$\mathcal{L}(N)$	Level of octant N.
$\mathcal{L}^*$	Maximum level attained by any octant.
D_{max}	Maximum permissible depth of the tree. (Upper bound for $\mathcal{L}^*$).
$\mathcal{P}(N)$	Parent of octant N.
$\mathcal{B}(N)$	The block that is equal to or is an ancestor of octant N.
$\mathcal{S}(N)$	Siblings (sorted) of octant N.
$\mathcal{C}(N)$	Children (sorted) of octant N.
$\mathcal{D}(N)$	Descendant of octant N.
$\mathcal{FC}(N)$	First child of octant N.
$\mathcal{LC}(N)$	Last child of octant N.
$\mathcal{FD}(N, l)$	First descendant of octant N at level l.
$\mathcal{LD}(N, l)$	Last descendant of octant N at level l.
$\mathcal{DFD}(N)$	Deepest first descendant of octant N.
$\mathcal{DLD}(N)$	Deepest last descendant of octant N.
$\mathcal{A}(N)$	Ancestor of octant N.
$\mathcal{A}_{finest}(N, K)$	Nearest Common Ancestor of octants N and K.
$\mathcal{N}(N, l)$	List of all potential neighbors of octant N at level l.
$\mathcal{N}^s(N, l)$	A subset of $\mathcal{N}(N, l)$, with the property that all of these share the same common corner with N. This is also the corner that N shares with its parent.
$\mathcal{N}(N)$	Neighbor of N at any level.
$\mathcal{I}(N)$	Insulation layer around octant N.
N^p_{max}	Maximum number of points per octant.
n_p	Total number of processors.
A_{global}	Union of the list A from all the processors.
$\{\cdots\}$	A set of elements.
$\emptyset$	The empty set.

Table 3.2: Symbols for operations

$A \leftarrow B$	Assignment operation.
$A \oplus B$	Bitwise A `XOR` B.
$\{A\} \cup \{B\}$	Union of the sets A and B. The order is preserved, if possible.
$\{A\} \cap \{B\}$	Intersection of the sets A and B.
$A + B$	The list formed by concatenating the lists A and B.
$A - B$	Remove the contents of B from A.
$A[i]$	i^{th} element in list A.
`len`(A)	Number of elements in list A.
`Sort`(A)	Sort A in the ascending Morton order.
A`.push_front(`B`)`	Insert B to the beginning of A.
A`.push_back(`B`)`	Append B to the end of A.
`Send(`A`,`r`)`	Send A to processor with rank = r.
`Receive()`	Receive from any processor.

its descendants. A node along with all its descendants can be viewed as a separate tree in itself with this node as its root. Hence, this set is also referred to as a *subtree* of the original tree. The depth of a node from the root is referred to as its *level*. As shown in Fig. 3.1a, the root of the tree is at level 0 and every interior node is one level lower than its children.

Octrees and quadtrees[6] can be used to partition cuboidal and rectangular regions, respectively (Fig. 3.1b). These regions are referred to as the domain of the tree. A set of octants is said to be complete if the union of the regions spanned by them covers the entire domain. Alternatively, one can also define complete octrees as octrees in which every interior node has exactly eight child nodes. We will frequently use the equivalence of these two definitions.

There are many different ways to represent trees [20]. In this work, we will use a linearized representation of octrees known as *linear octrees*. In this representation, we discard the interior nodes and only store the complete list of leaves. This representation is advantageous for the following reasons.

- It has lower storage costs than other representations.

[6] All the algorithms described in this chapter are applicable to both octrees and quadtrees. For simplicity, we will use quadtrees to illustrate the concepts and use the terms 'octrees' and 'octants', consistently, in the rest of the thesis.

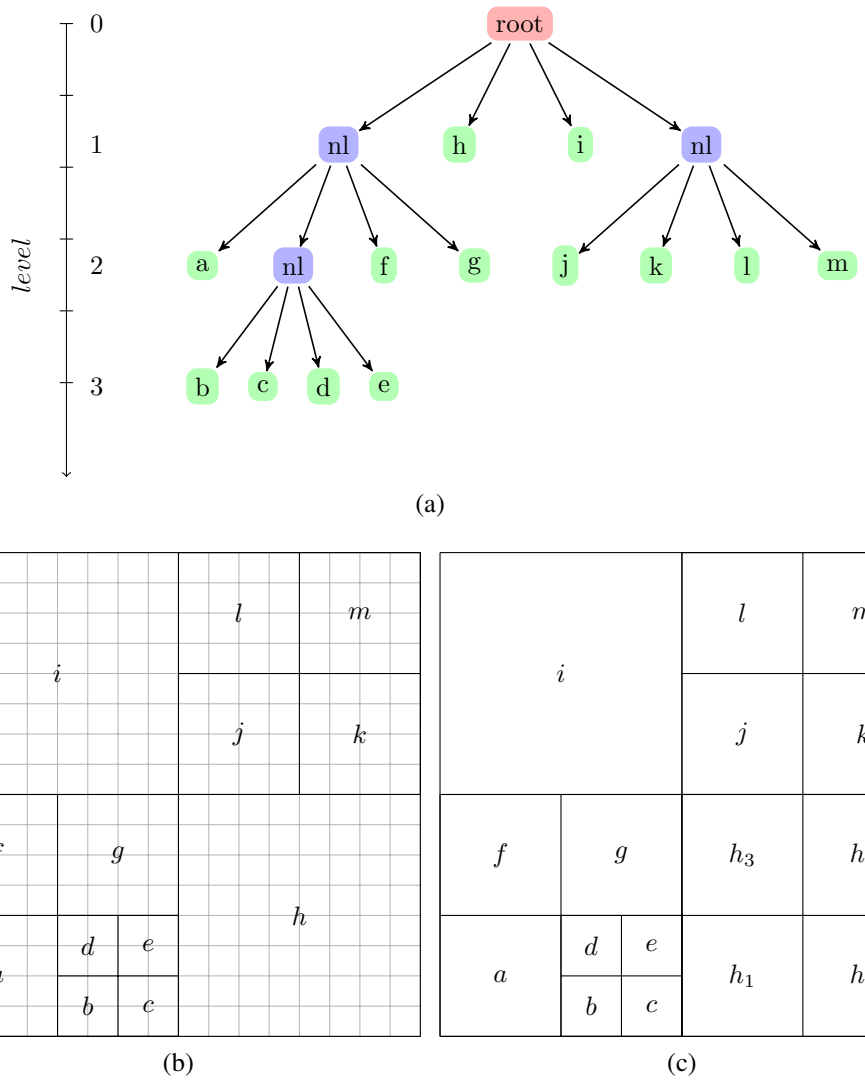


Figure 3.1: (a) Tree representation of a quadtree and (b) decomposition of a square domain using the quadtree, superimposed over a uniform grid, and (c) a balanced linear quadtree: result of balancing the quadtree.

- The other representations use pointers, which add synchronization and communication overhead for parallel implementations.

To use a linear representation, a *locational code* is needed to identify the octants. A locational code is a code that contains information about the position and level of the octant in the tree. The following section describes one such locational code known as the *Morton encoding*.[7]

3.1.1 Morton encoding

In order to construct a Morton encoding, the maximum permissible depth, D_{max}, of the tree is specified *a priori*. Note that D_{max} is different from $\mathcal{L}^*$, the maximum level attained by any node. In general, $\mathcal{L}^*$ can not be specified a priori. D_{max} is only a weak upper bound for $\mathcal{L}^*$.

The domain is represented by an uniform grid of $2^{D_{max}}$ indivisible cells in each dimension (Fig. 3.1b). Each cell is identified by an integer triplet representing its x, y and z coordinates, respectively. Any octant in the domain can be uniquely identified by specifying one of its vertices, also known as its *anchor*, and its level in the tree (Fig. 3.3). By convention, the anchor of a quadrant is it's lower left corner and the anchor of an octant is it's lower left corner facing the reader (corner v_0 in Figure 3.2).

The Morton encoding for any octant is derived by interleaving[8] the binary representations (D_{max} bits each) of the three coordinates of the octant's anchor, and then appending the binary representation (($\lfloor(\log_2 D_{max})\rfloor + 1$) bits) of the octant's level to this sequence of bits [9, 16, 104, 108]. Interesting properties of the Morton encoding scheme are listed in Appendix 3.1.2. In the rest of the chapter the terms *lesser* and *greater* and the symbols $<$ and $>$ are used to compare octants based on their Morton ids, and *coarser* and *finer* to compare them based on their relative sizes, i.e., their levels in the octree.

[7] Morton encoding is one of many space-filling curves [16]. Our algorithms are generic enough to work with other space-filling curves as well. However, Morton encoding is relatively simpler to implement since, unlike other space-filling curves, no rotations or reflections are performed.

[8] Instead of bit-interleaving as described here, we use a multicomponent version (Appendix 3.1.3) of the Morton encoding scheme.

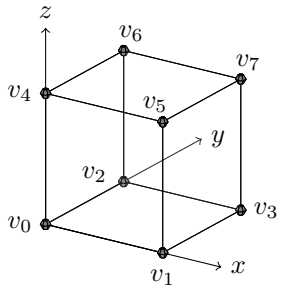


Figure 3.2: Orientation for an octant. By convention, v_0 is chosen as the anchor of the octant. The vertices are numbered in the Morton ordering.

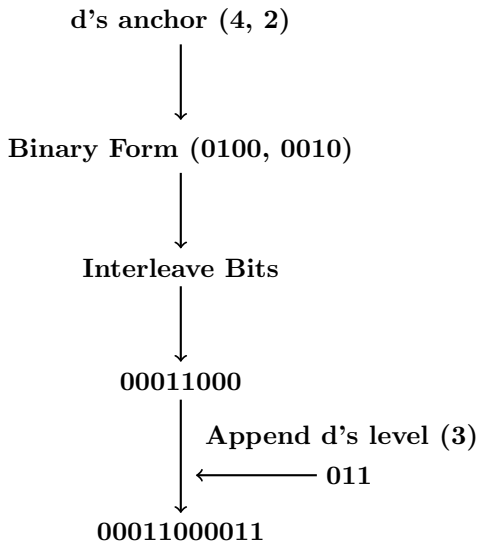


Figure 3.3: Computing the Morton id of quadrant 'd' in the quadtree shown in Fig. 3.1b. The anchor for any quadrant is it's lower left corner.

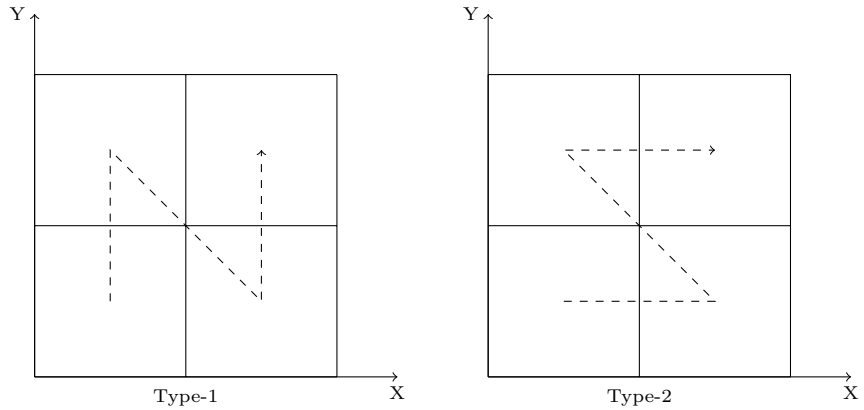


Figure 3.4: Two types of z-ordering in quadtrees.

3.1.2 Properties of Morton encoding

Property 1 *Sorting all the leaves in the ascending order of their Morton ids is identical to a preorder traversal of the leaves of the octree. If one connects the centers of the leaves in this order, one can observe a Z-pattern in the Cartesian space. The space-filling Z-order curve has the property that spatially nearby octants tend to be clustered together. The octants in Figures 3.1b and 3.1c are all labeled according to this order. Depending on the order of interleaving the coordinates, different Z-order curves are obtained. The two possible Z-curves in 2-D are shown in the Figure 3.4. Similarly, in 3-D six different types of Morton ordering are possible.*

Property 2 *Given three octants, $a < b < c$ and $c \notin \{\mathcal{D}(b)\}$:*

$$a < d < c, \quad \forall d \in \{\mathcal{D}(b)\}.$$

Property 3 *The Morton id of any node is less than those of its descendants.*

Property 4 *Two distinct octants overlap if and only if one is an ancestor of the other.*

Property 5 *The Morton id of any node and of its first child*[9] *are consecutive. It follows from Property 3 that the first child is also the child with the least Morton id.*

Property 6 *The first descendant at level l, denoted by* $\mathcal{FD}(N,l)$*, of any node N is the descendant at level l with the least Morton id. This can be arrived at by following the first child at every level starting from N.* $\mathcal{FD}(N,D_{max})$ *is also the anchor of N and is also referred to as the* deepest first descendant*, denoted by* $\mathcal{DFD}(N)$*, of node N.*

Property 7 *The range* $(N,\mathcal{DFD}(N)]$ *only contains the first descendants of N at different levels and hence there can be no more than one leaf in this range in the entire linear octree.*

Property 8 *The* last descendant *at level l, denoted by* $\mathcal{LD}(N,l)$*, of any node N is the descendant at level l with the greatest Morton id. This can be arrived at by following the* last child[10] *at every level starting from N.* $\mathcal{LD}(N,D_{max})$ *is also referred to as the* deepest last descendant*, denoted by* $\mathcal{DLD}(N)$*, of node N.*

Property 9 *Every octant in the range* $(N,\mathcal{DLD}(N)]$ *is a descendant of N.*

3.1.3 Multicomponent Morton Representation

Every Morton id is a set of 4 entities: The three co-ordinates of the anchor of the octant and the level of the octant. We have implemented the node as a C++ class, which contains these 4 entities as its member data. To use this set as a locational code for octants, we define two primary binary logical operations on it: a) Comparing if 2 ids are equal and b) Comparing if one id is lesser than the other.

Two ids are equal if and only if all the 4 entities are respectively equal. If two ids have the same anchor then the one at a coarser level has a lesser Morton id. If the anchors are different, then we can use Algorithm 1 to determine the lesser id. The Z-ordering produced by this operator is identical to that produced by the scalar Morton ids described in section 3.1.1. The other logical operations can be readily derived from these two operations.

[9] *the child that has the same anchor as the parent*
[10] *child with the greatest Morton id*

Algorithm 1 FINDING THE LESSER OF TWO MORTON IDS (SEQUENTIAL)

Input: Two Morton ids, A and B with different anchors.
Output: R, the lesser of the two Morton ids.

1. $X_i \leftarrow (A_i \oplus B_i)$, $i \in \{x, y, z\}$
2. $e \leftarrow \arg\max_i (\lfloor \log_2(X_i) \rfloor)$

3. **if** $A_e < B_e$
   $R \leftarrow A$
4. **else**
   $R \leftarrow B$
5. **end if**

3.1.4 Balance Constraint

In many applications involving octrees, it is desirable to impose a restriction on the relative sizes of adjacent octants [36, 43, 108]. Generalizing Moore's [60] categorization of the general balance conditions, we have the following definition for the 2:1 balance constraint:

Definition 1 *A linear d-tree is k-balanced if and only if, for any $l \in [1, \mathcal{L}^*)$, no leaf at level l shares an m-dimensional face*[11] *($m \in [k, d)$) with another leaf, at level greater than $l + 1$.*

For the specific case of octrees we use 2-*balanced* to refer to octrees that are balanced across faces, 1-*balanced* to refer to octrees that are balanced across edges and faces, and 0-*balanced* to refer to octrees that are balanced across corners, edges and faces. The result of imposing the 2:1 balance constraint is that no octant can be more than twice as coarse as its adjacent octants. Similarly, 4:1 and higher constraints can be imposed. In this work, we will restrict the discussion to 2:1 balancing alone. Although, the algorithms presented in this work can be extended easily to satisfy higher balance constraints as well. An example of a 0-*balanced* quadtree is shown in Figure 3.1c. The balance algorithm proposed in this

[11] *A corner is a 0-dimensional face, an edge is a 1-dimensional face and a face is a 2-dimensional face.*

work is capable of *k-balancing* a given complete linear octree, and since it is hardest to 0-*balance* a given octree we report all results for the 0-*balance* case.

3.2 Algorithms for Constructing and Balancing Octrees

We will first describe a key algorithmic component (Section 3.2.1) that forms the backbone for both our parallel octree construction and balancing algorithms. This is a partition heuristic known as *Block Partition* and is specifically designed for octrees. It has two main sub-components, which are described in Sections 3.2.1 and 3.2.1.

We then present the parallel octree construction algorithm in Section 3.2.3 and the parallel balancing algorithm in Section 3.2.5. The overall parallel balancing algorithm (Algorithm 12) is made up of numerous components, which are described in Sections 3.2.5 through 3.2.5.

3.2.1 Block Partition

A simple way to partition the domain into an union of blocks would be to take a top-down approach and create a coarse regular grid, which can be divided[12] amongst the processors. However, this approach does not take load balancing into account since it does not use the underlying data distribution. Alternatively, one could use a space-filling curve to sort the octants and then partition them so that every processor gets an almost equal sized chunk of octants, contiguous in this order. This can be done by assigning the same weight to all the octants and then using Algorithm 3. However, this approach does not avoid overlaps.

Two desirable qualities of any partitioning strategy are load balancing, and minimization of overlap between the processor domains. We use a novel parallel bottom-up coarsening strategy to achieve these. The main intuition behind this partition algorithm (Algorithm 2) is that a coarse grid partition is more likely to have a smaller overlap between the processor domains as compared to a partition computed on the underlying fine grid. This

[12] If we create a regular grid at level l then the number of cells will be $n = 2^{dl}$, where d is the dimension. l is chosen in such a way that $n > p$.

algorithm comprises of 3 main stages:

1. Constructing a distributed coarse complete linear octree that is representative of the underlying data distribution.

2. Assigning weights to the octants in the coarse octree and partitioning them to achieve almost uniform load across the processors.

3. Projecting the partitioning computed in the previous step onto the original (fine) linear octree.

We sort the leaves according to their Morton ordering and then distribute them uniformly across the processors. We select the least and the greatest octant at each processor (e.g., octants *a* and *h* from Figure 3.5a) and complete the region between them, as described in Section 3.2.1, to obtain a list of coarse octants. We then select the coarsest cell(s) out of this list of coarse octants (octant *e* in Figure 3.5a). We use the selected octants at each processor and construct a complete linear octree as described in Section 3.2.1. The leaves of this complete linear octree are referred to as *Blocks*. This gives us a distributed coarse complete linear octree that is based on the underlying data distribution.[13]

We compute the load of each of the blocks created above by computing the number of original octants that lie within it. The blocks are then distributed across the processors using Algorithm 3 so that the total weight on each processor is roughly the same.[14]

The original octants are then partitioned to align with the coarse block boundaries. Note that the domain occupied by the blocks and the original octants on any given processor is not the same, but it does overlap to a large extent. The overlap is guaranteed by the fact that both are sorted according to the Morton ordering and that the partitioning was based on the same weighting function (i.e., the number of original octants).

Algorithm 2 lists all the steps described above and Figures 3.5b and 3.5c illustrate a sample input to Algorithm 2 and the corresponding output, respectively.

[13]Refer to the Appendix 3.2.2 for an estimate of the number of blocks produced.

[14]Some of the coarse blocks could be split if it facilitates achieving better load balance across the processors.

Algorithm 2 - BlockPartition
PARTITIONING OCTANTS INTO LARGE CONTIGUOUS BLOCKS (PARALLEL)

Input: A distributed sorted list of octants, F.

Output: A list of the blocks, G. F is re-distributed, but the relative order of the octants is preserved.

Work: $O(n)$, where $n = \text{len}(F)$.
Storage: $O(n)$, where $n = \text{len}(F)$.
Time: Refer to the Appendix 3.2.2.

1. $T \leftarrow$ CompleteRegion($F[1]$, $F[\text{len}(F)]$) (Algorithm 4)
2. $C \leftarrow \{x \in T \mid \forall y \in T,\ \mathcal{L}(x) \leq \mathcal{L}(y)\}$
3. $G \leftarrow$ CompleteOctree(C) (Algorithm 5)
4. **for each** $g \in G$
5. weight(g) $\leftarrow \text{len}\Big(F_{global} \cap \{g, \{\mathcal{D}(g)\}\}\Big)$
6. **end for**
7. Partition(G) (Algorithm 3)
8. $F \leftarrow F_{global} \cap \{\{g, \{\mathcal{D}(g)\}\},\ \forall\ g \in G\}$

Algorithm 3 - `Partition`
PARTITIONING A DISTRIBUTED LIST OF OCTANTS (PARALLEL)

Input: A distributed list of octants, W.

Output: The octants re-distributed across processors so that the total weight on each processor is roughly the same. The relative order of the octants is preserved.

Work: $O(n)$, where $n = \text{len}(W)$.
Storage: $O(n)$, where $n = \text{len}(W)$.

1. $S \leftarrow$ **Scan**(weight(W))
2. **if** rank $= (n_p - 1)$
3. TotalWeight $\leftarrow \max(S)$
4. **Broadcast**(TotalWeight)
5. **end if**
6. $\bar{w} \leftarrow \frac{\texttt{TotalWeight}}{n_p}$
7. $k \leftarrow (\texttt{TotalWeight}) \mod n_p$
8. $Q_{tot} \leftarrow \emptyset$
9. **for** $p \leftarrow 1$ **to** n_p
10. **if** $p \leq k$
11. $Q \leftarrow \{x \in W \mid (p-1).(\bar{w}+1) \leq S(x) < p.(\bar{w}+1)\}$
12. **else**
13. $Q \leftarrow \{x \in W \mid (p-1).\bar{w} + k \leq S(x) < p.\bar{w} + k\}$
14. **end if**
15. $Q_{tot} \leftarrow Q_{tot} + Q$
16. **Send**(Q, $(p-1)$)
17. **end for**
18. $R \leftarrow$ **Receive**()
19. $W \leftarrow W - Q_{tot} + R$

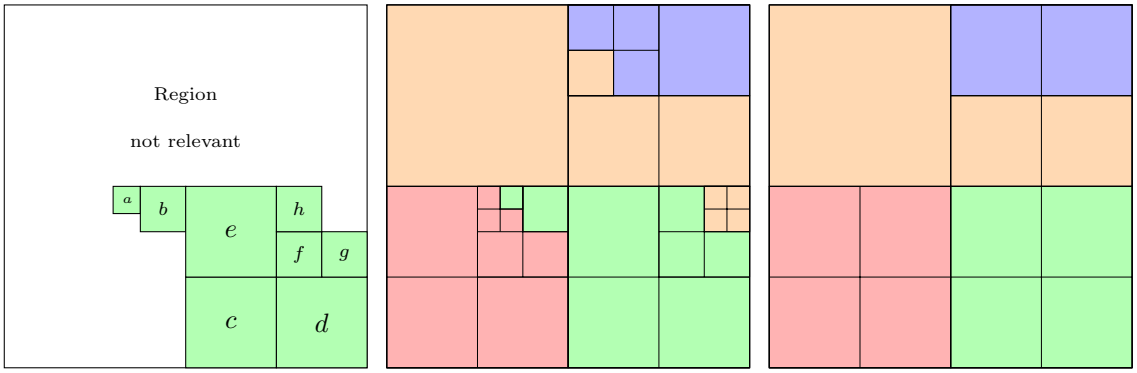


Figure 3.5: (a) A minimal list of quadrants covering the local domain on a processor, and (b) A Morton ordering based partition of a quadtree across 4 processors, and (c) the coarse quadrants and the final partition produced by using the quadtree shown in (b) as input to Algorithm 2.

Constructing a minimal linear octree between two octants

Given two octants, a and $b > a$, we wish to generate the minimal number of octants that span the region between a and b according to the Morton ordering. The algorithm (Algorithm 4) first calculates the nearest common ancestor of the octants a and b. This octant is split into its eight children. Out of these, only the octants that are either greater than a and lesser than b or ancestors of a are retained and the rest are discarded. The ancestors of either a or b are split again and we iterate until no further splits are necessary. This produces the minimal coarse complete linear octree (Figure 3.6b) between the two octants a and b (Figure 3.6a). This algorithm is based on the Properties 2 and 3 of the Morton ordering, which are listed in Appendix 3.1.2.

Constructing complete linear octrees from a partial set of octants

In order to construct a complete linear octree from a partial set of octants (e.g. Figure 3.6c), the octants are initially sorted based on the Morton ordering. Algorithm 9 is subsequently used to remove overlaps, if any. Two additional octants are added to complete the domain

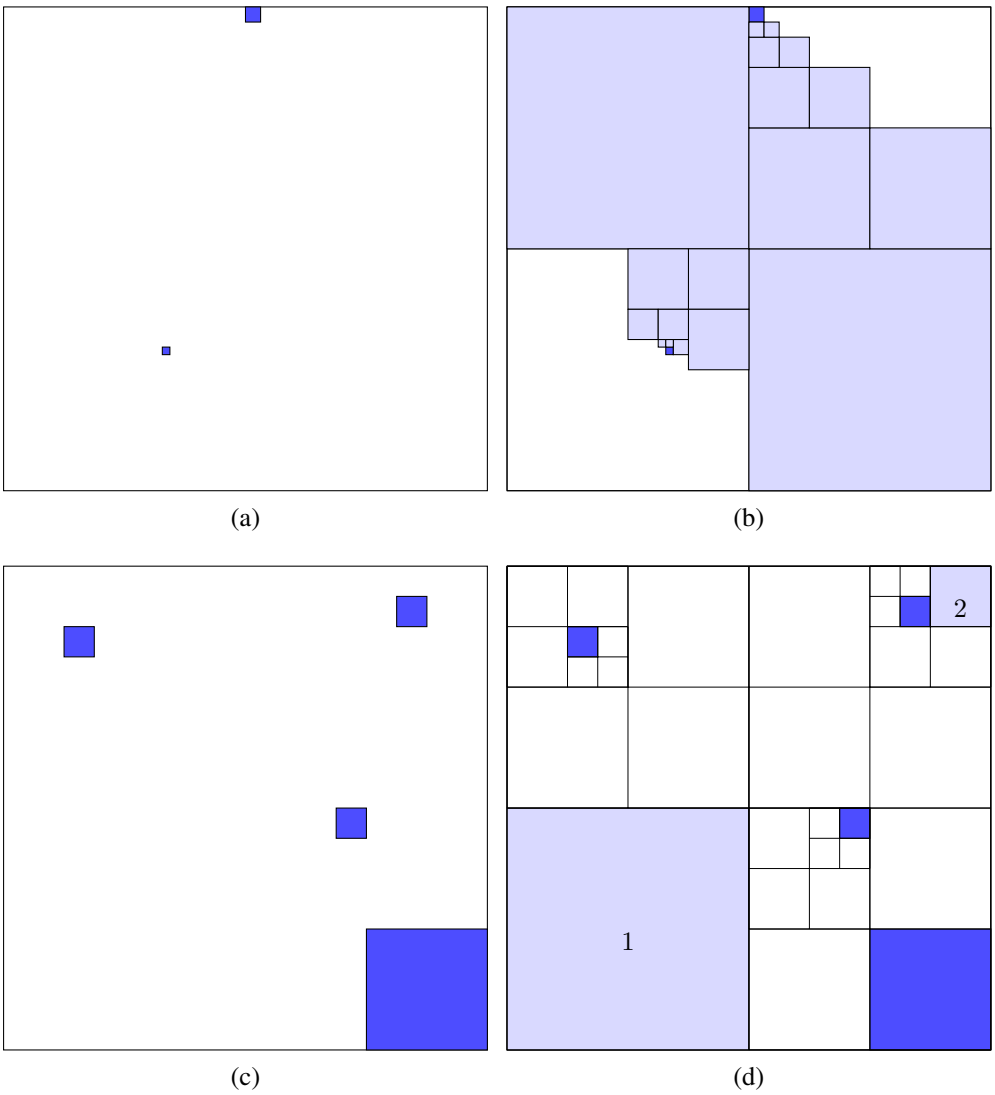


Figure 3.6: (b) The minimal number of octants between the cells given in (a). This is produced by using (a) as an input to `Algorithm 4`. (d) The coarsest possible complete linear quadtree containing all the cells in (c). This is produced by using (c) as an input to `Algorithm 5`. The figure also shows the two additional octants added to complete the domain. The first one is the coarsest ancestor of the least possible octant (the deepest first descendant of the root octant), which does not overlap the least octant in the input. This is also the first child of the nearest common ancestor of the least octant in the input and the deepest first decendant of root. The second is the coarsest ancestor of the greatest possible octant (the deepest last descendant of the root octant), which does not overlap the greatest octant in the input. This is also the last child of the nearest common ancestor of the greatest octant in the input and the deepest last decendant of root.

Algorithm 4 - `CompleteRegion`
CONSTRUCTING A MINIMAL LINEAR OCTREE BETWEEN TWO OCTANTS (SEQUENTIAL)

Input: Two octants, a and $b > a$.
Output: R, the minimal linear octree between a and b.
Work: $O(n \log n)$, where $n =$ len(R).
Storage: $O(n)$, where $n =$ len(R).

```
1.  W ← C(A_finest(a, b))
2.  for each w ∈ W
3.      if (a < w < b) AND (w ∉ {A(b)})
4.          R ← R + w
5.      else if (w ∈ {{A(a)}, {A(b)}})
6.              W ← W − w + C(w)
7.      end if
8.  end for
9.  Sort(R)
```

(Figure 3.6d). The first is the coarsest ancestor of the least possible octant (the deepest first descendant of the root octant, Property 6), which does not overlap the least octant in the input. This is also the first child of the nearest common ancestor of the least octant in the input and the deepest first decendant of root. The second is the coarsest ancestor of the greatest possible octant (the deepest last descendant of the root octant, Property 8), which does not overlap the greatest octant in the input. This is also the last child of the nearest common ancestor of the greatest octant in the input and the deepest last decendant of root. The octants are distributed across the processors to get a weight-based uniform load distribution. The local complete linear octree is subsequently generated by completing the region between every consecutive pair of octants as described in Section 3.2.1. Each processor is also responsible for completing the region between the first octant owned by that processor and the last octant owned by the previous processor, thus ensuring that a global complete linear octree is produced.

3.2.2 Analysis of the Block Partitioning Algorithm

Assume that the input to the partitioning algorithm is a sorted distributed list of N octants. Then, we can guarantee coarsening of the input if there are more than eight octants[15] per processor. The minimum number of octants on any processor, n_{min}, can be expressed in terms of N and the imbalance factor[16], c, as follows:

$$n_{min} = \frac{N}{1 + c(n_p - 1)}.$$

This implies that the coarsening algorithm will coarsen the octree if,

$$\begin{aligned} n_{min} = \frac{N}{1 + c(n_p - 1)} &> 2^d, \\ \implies N &> 2^d(1 + c(n_p - 1)). \end{aligned}$$

The total number of blocks created by our coarsening algorithm is $O(p)$. Specifically,

[15] 2^d cells for a d-tree.

[16] The imbalance factor is the ratio between the maximum and minimum number of octants on any processor.

Algorithm 5 - `CompleteOctree`
CONSTRUCTING A COMPLETE LINEAR OCTREE FROM A PARTIAL (INCOMPLETE) SET OF OCTANTS (PARALLEL)

Input: A distributed sorted list of octants, L.
Output: R, the complete linear octree.
Work: $O(n \log n)$, where $n =$ len(R).
Storage: $O(n)$, where $n =$ len(R).

```
1.  RemoveDuplicates(L)
2.  L ← Linearise(L) ( Algorithm 9 )
3.  Partition(L) ( Algorithm 3 )
4.  if rank = 0
5.    L.push_front(FC(A_finest(DFD( root ), L[1])))
6.  end if
7.  if rank = (n_p − 1)
8.    L.push_back(LC(A_finest(DLD( root), L[len (L)])))
9.  end if
10. if rank > 0
11.   Send(L[1] ,(rank−1) )
12. end if
13. if rank < (n_p − 1)
14.   L.push_back(Recieve() )
15. end if
16. for i ← 1 to (len(L) − 1)
17.     A ← CompleteRegion (L[i], L[i + 1]) ( Algorithm 4 )
18.     R ← R + L[i] + A
19. end for
20. if rank = (n_p − 1)
21.   R ← R + L[len(L)]
22. end if
```

the total number of blocks produced by the coarsening algorithm, N_{blocks}, satisfies:

$$p \leq N_{blocks} < 2^d p.$$

If the input is sorted and if $c \approx 1$, then the communication cost for this partition is $O(\frac{N}{n_p})$.

3.2.3 Constructing large linear octrees in parallel

Octrees are usually constructed by using a top-down approach: starting with the root octant, cells are split iteratively based on some criteria, until no further splits are required. This is a simple and efficient sequential algorithm. However, it's parallel analogue is not so. We use the case of point datasets to discuss some shortcomings of a parallel top-down tree construction. Formally, the problem might be stated as: Construct a complete linear octree in parallel from a distributed set of points in a domain with the constraint that no octant should contain more than (N^p_{max}) number of points. Each processor can independently construct a tree using a top-down approach on its local set of points. Constructing a global linear octree requires a parallel merge. Merging however, is not straightforward.

1. Consider the case where the local number of points in some region on every processor was less than (N^p_{max}), and hence all the processors end up having the same level of coarseness in the region. However, the total number of points in that region could be more than (N^p_{max}) and hence the corresponding octant should be refined further.

2. In most applications, we would also like to associate a unique processor to each octant. Thus, duplicates across processors must be removed.

3. For linear octrees overlaps across processors must be resolved.

4. Since there might be overlaps and duplicates, not all the work done by the processors can be accounted as useful work. This is a subtle yet important point to consider while analyzing the algorithm for load-balancing.

Previous work [35, 108, 115, 119] on this problem has addressed these issues; However, all the existing algorithms involve many synchronization steps and thus suffer from a sizable overhead, resulting in suboptimal isogranular scalability. Instead, we propose a bottom-up approach for constructing octrees from points. The crux of the algorithm is to distribute the data across the processors in such a way that there is uniform load distribution across processors and the subsequent operations to build the octree can be performed by the processors independently, i.e., requiring no additional communication.

First, all points are converted into octants at the maximum depth and then partitioned across the processors using the algorithm described in Section 3.2.1. This produces a contiguous set of coarse blocks (with their corresponding points) on each processor. The complete linear octree is generated by iterating through the blocks and by splitting them based on number of points per block.[17] This process is continued until no further splits are required. This procedure is summarized in Algorithm 6.

3.2.4 Special case during construction

We can not always guarantee the coarsest possible octree for an arbitrary distribution of N points and arbitrary values of N^p_{max}, especially when $N^p_{max} \approx \frac{N}{n_p}$. However, if every processor has at least two *well-separated* [18] points and if $N^p_{max} = 1$, then the algorithm will produce the coarsest possible octree under these constraints. However, this is not too restrictive because the input points can always be sampled in such a way that the algorithm produces the desired octree. Besides, the maximum depth of the octree can also be used to control the coarseness of the resulting octree. In all our experiments, we used $N^p_{max} = 1$ and we always got the same octree for different number of processor counts (Table 3.5).

[17] Refer to the Appendix 3.2.4 on how to sample the points in order to construct the coarsest possible octree.

[18] Convert the points into octants at D_{max} level. If there exists at least one coarse octant between these two octants, then the points are considered to be well-separated.

Algorithm 6 - `Points2Octree`
CONSTRUCTING A COMPLETE LINEAR OCTREE FROM A DISTRIBUTED LIST OF POINTS (PARALLEL)

Input: A distributed list of points, L and a parameter, (N^p_{max}), which specifies the maximum number of points per octant.
Output: Complete linear Octree, B.
Work: $O(n \log n)$, where $n =$ len(L).
Storage: $O(n)$, where $n =$ len(L).

```
1.  F ← [Octant(p, D_max), ∀p ∈ L]
2.  Sort(F)
3.  B ← BlockPartition(F) ( Algorithm 2 )

4.  for each b ∈ B
5.      if NumberOfPoints(b) > N^p_max
6.          B ← B − b + C(b)
7.      end if
8.  end for
```

3.2.5 Balancing large linear octrees in parallel

Balance refinement is the process of refining (subdividing) nodes in a complete linear octree, which fail to satisfy the balance constraint described in Section 3.1.4. The nodes are refined until all their descendants, which are created in the process of subdivision, satisfy the balance constraint. These subdivisions could in turn introduce new imbalances and so the process has to be repeated iteratively. The fact that an octant can affect octants not immediately adjacent to is known as the *ripple effect*.

We use a two-stage balancing scheme: first we perform local balancing on each processor, and follow this up by balancing across the inter-processor boundaries. One of the goals is to get a union of blocks (select non-leaf nodes of the octree) to reside on each processor so that the surface area and thereby the corresponding inter-processor boundaries are minimized. Determining whether a given partition provides the minimal surface area[19] is NP complete and determining the optimal partition is NP hard, since the problem is equivalent to the set-covering problem [20].

We use the parallel bottom-up coarsening and partitioning algorithm (described in section 3.2.1) to construct coarse blocks on each processor and to distribute the underlying octants. By construction, the domains covered by these blocks are disjoint and the union of these blocks covers the entire domain. We use the blocks as a means to minimize the number of octants that need to be split due to inter-processor violations of the 2:1 balancing rule.

Local balancing

There are two approaches for balancing a complete octree. In the first approach, every node constructs the coarsest possible neighbors satisfying the balance constraint, and subsequently duplicates and overlaps are removed [9]. We describe this approach in Algorithm 7. In an alternative approach, the nodes search for neighbors and resolve any violations of

[19]The number of cells at the boundary depends on the underlying distribution and cannot be known *a priori*. This further complicates the balancing algorithm.

the balance constraint [106, 108]. The main advantage of the former approach is that constructing nodes is inexpensive, since it does not involve any searches. However, this could produce a lot of duplicates and overlaps making the linearizing operations expensive. Another disadvantage of this approach is that it cannot handle incomplete domains, and can only operate on subtrees. The advantage of the second approach is that the list of nodes is complete and linear at any stage in the algorithm. The drawback, however, is that searching for neighbors is an expensive operation. Our algorithm uses a hybrid approach: it keeps the number of duplicates and overlaps to a minimum and also reduces the search space thereby reducing the cost of the searching operation. The complete linear octree is first partitioned into coarse blocks using the algorithm described in Section 3.2.1. The descendants of any block, which are present in the fine octree, form a linear subtree with this block as its root. This block-subtree is first balanced using the approach described in Section 3.2.5; the size of this tree will be relatively small, and hence the number of duplicates and overlaps will be small too. After balancing all the blocks, the inter-block boundaries in each processor are balanced using a variant of the *ripple propagation* algorithm [108] described in Section 3.2.5. The performance improvements from using the combined approach are presented in Section 3.7.1.

Balancing a local block

In principle, Algorithm 7 can be used to construct a complete balanced subtree of this block for each octant in the initial unbalanced linear subtree. Note that these balanced subtrees may have substantial overlap. Hence, Algorithm 9 is used to remove these overlaps. Lemma 1 shows that this process of merging these different balanced subtrees results in a complete linear balanced subtree. However, this implementation would be inefficient due to the number of overlaps, which would in turn increase the storage costs and also make the subsequent operations of sorting and removing duplicates and overlaps more expensive. Instead, we interleave the two operations: constructing the different complete balanced subtrees and merging them. The overall scheme is described in Algorithm 8.

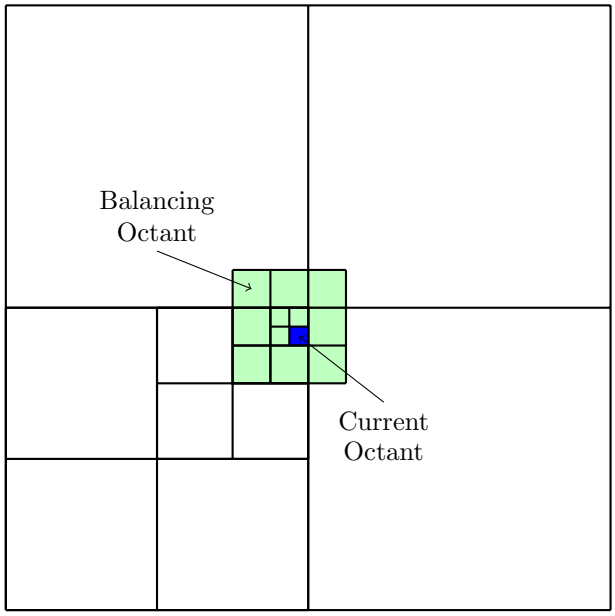


Figure 3.7: The minimal list of balancing quadrants for the current quadrant is shown. This list of quadrants is generated in one iteration of `Algorithm 7`.

We note that a list of octants forms a balanced complete octree, if and only if for every octant all its neighbors are at the same level as this octant or one level finer or one level coarser. Hence, the coarsest possible octants in a complete octree that will be balanced against this octant are the siblings and the neighbors at the level of this octant's parent. Starting with the finest level and iterating over the levels up to but not including the level of the block, the coarsest possible (without violating the balance constraint) neighbors (Figure 3.7) of every octant at this level in the current tree (union of the initial unbalanced linear subtree and newly generated octants) are generated. After processing all the octants at any given level, the list of newly introduced coarse octants is merged with the previous list of octants at this level and duplicate octants are removed. The newly created octants are included while working on subsequent levels. Algorithm 9 still needs to be used in the end to remove overlaps, but the working size is much smaller now compared to the earlier case (Algorithm 7). To avoid redundant work and to reduce the number of duplicates to be removed in the end, we ensure that no two elements in the working list at any given level are siblings of one another. This can be done in a linear pass on the working list for that

level as shown in Algorithm 8.

Lemma 1 *Let T_1 and T_2 be two complete balanced linear octrees with n_1 and n_2 number of potential ancestors respectively, then*

$$T_3 = (T_1 \cup T_2) - \left(\sum_{i=1}^{n_1} \{\mathcal{A}(T_1[i])\}\right) - \left(\sum_{j=1}^{n_2} \{\mathcal{A}(T_2[j])\}\right)$$

is a complete linear balanced octree.

Proof 1 *$T_4 = (T_1 \cup T_2)$ is a complete octree. Now,*

$$\left(\left(\sum_{i=1}^{n_1} \{\mathcal{A}(T_1[i])\}\right) + \left(\sum_{j=1}^{n_2} \{\mathcal{A}(T_2[j])\}\right)\right) = \left(\sum_{k=1}^{n_3} \{\mathcal{A}(T_4[k])\}\right)$$

So, $T_3 = \left(T_4 - \left(\sum_{k=1}^{n_3} \{\mathcal{A}(T_4[k])\}\right)\right)$ is a complete linear octree.
Now, suppose that a node $N \in T_3$ has a neighbor $K \in T_3$ such that $\mathcal{L}(K) \geq (\mathcal{L}(N) + 2)$. It is obvious that exactly one of N and K must be present in T_1 and the other must be present in T_2. Without loss of generality, assume that $N \in T_1$ and $K \in T_2$. Since T_2 is complete, there exists at least one neighbor of K,$L \in T_2$, which overlaps N. Also, since T_2 is balanced $\mathcal{L}(L) = \mathcal{L}(K)$ or $\mathcal{L}(L) = (\mathcal{L}(K) - 1)$ or $\mathcal{L}(L) = (\mathcal{L}(K) + 1)$. So, $\mathcal{L}(L) \geq (\mathcal{L}(N) + 1)$. Since L overlaps N and since $\mathcal{L}(L) \geq (\mathcal{L}(N) + 1)$, $L \in \{\mathcal{D}(N)\}$. Hence, $N \notin T_3$. This contradicts the initial assumption. Therefore, T_3 is also balanced.

Searching for neighbors

A leaf needs to be refined if and only if the level of one of its neighbors is at least 2 levels finer than its own. In terms of a search this presents us two options: search for coarser neighbors or search for finer neighbors. It is much easier to search for coarser neighbors than it is to search for finer neighbors. If we consider the 2D case, only 3 neighbors coarser than the current cell need to be searched for. However, the number of potential neighbors finer than the cell is extremely large, (in 2D it is $2 \cdot 2^{D_{max}-l} + 3$, where l is the level of the current quadrant), and therefore not practical to search. In addition the search strategy

Algorithm 7 -
CONSTRUCTING A COMPLETE BALANCED SUBTREE OF AN OCTANT, GIVEN ONE OF ITS DESCENDANTS (SEQUENTIAL)

Input: An octant, N, and one of its descendants, L.
Output: Complete balanced subtree, R.
Work: $O(n \log n)$, where $n = \text{len}(R)$.
Storage: $O(n)$, where $n = \text{len}(R)$.

1. $W \leftarrow L,\ T \leftarrow \emptyset,\ R \leftarrow \emptyset$

2. **for** $l \leftarrow \mathcal{L}(L)$ **to** $(\mathcal{L}(N) + 1)$
3. **for each** $w \in W$
4. $R \leftarrow R + w + \mathcal{S}(w)$
5. $T \leftarrow T + \{\mathcal{N}(\mathcal{P}(w), l - 1) \cap \{\mathcal{D}(N)\}\}$
6. **end for**
7. $W \leftarrow T,\ T \leftarrow \emptyset$
8. **end for**

9. Sort(R)
10. RemoveDuplicates(R)
11. $R \leftarrow$ Linearise(R) (Algorithm 9)

Algorithm 8 - `BalanceSubtree`
BALANCING A LOCAL BLOCK (SEQUENTIAL)

Input: An octant, N, and a partial list of its descendants, L.
Output: Complete balanced subtree, R.
Work: $\mathcal{O}(n \log n)$, where $n =$ len(R).
Storage: $\mathcal{O}(n)$, where $n =$ len(R).

1. $W \leftarrow L,\ P \leftarrow \emptyset,\ R \leftarrow \emptyset$

2. **for** $l \leftarrow D_{max}$ **to** $(\mathcal{L}(N) + 1)$
3. $\quad Q \leftarrow \{x \in W \mid \mathcal{L}(x) = l\}$
4. $\quad$ Sort(Q)
5. $\quad T \leftarrow \{x \in Q \mid \mathcal{S}(x) \notin T\}$
6. $\quad$ **for each** $t \in T$
7. $\quad\quad R \leftarrow R + t + \mathcal{S}(t)$
8. $\quad\quad P \leftarrow P + \{\mathcal{N}(\mathcal{P}(t), l-1) \cap \{\mathcal{D}(N)\}\}$
9. $\quad$ **end for**
10. $\quad P \leftarrow P + \{x \in W \mid \mathcal{L}(x) = l - 1\}$
11. $\quad W \leftarrow \{x \in W | \mathcal{L}(x) \neq l - 1\}$
12. $\quad$ RemoveDuplicates(P)
13. $\quad W \leftarrow W + P,\ P \leftarrow \emptyset$
14. **end for**

15. Sort(R)
16. $R \leftarrow$ Linearise(R) (Algorithm 9)

Algorithm 9 - `Linearise`
REMOVING OVERLAPS FROM A SORTED LIST OF OCTANTS (SEQUENTIAL)

Input: A sorted list of octants, W.
Output: R, an octree with no overlaps.
Work: $O(n)$, where $n = \texttt{len}(W)$.
Storage: $O(n)$, where $n = \texttt{len}(W)$.

```
1.  for i ← 1 to (len(W) − 1)
2.      if (W[i] ∉ {A(W[i + 1])})
3.          R ← R + W[i]
4.      end if
5.  end for
6.  R ← R + W[len(W)]
```

depends on the way the octree is stored; the pointer based approach is more popular [9, 106], but has the overhead that it has to be rebuilt every time octants are communicated across processors. In the proposed approach the octree is stored as a linear octree in which the octants are sorted globally in the ascending Morton order, allowing us to search in $O(\log n)$.

In order to find neighbors coarser than the current cell, we use the approach illustrated in Figure 3.8. First, the finest cell at the far corner (marked as 'Search Corner' in Figure 3.8) is determined. This is the corner that this octant shares with its parent. This is also the corner diagonally opposite to the corner common to all the siblings of the current cell.[20] The neighbors (at the finest level) of this cell (N) are then selected and used as the search keys. These are denoted by $\mathcal{N}^s(N, D_{max})$. The maximum lower bound[21] for the given search key is determined by searching within the complete linear octree. In a complete linear octree, the maximum lower bound of a search key returns its finest ancestor. If the search result is at a level finer than or equal to the current cell then it is guaranteed that

[20] We do not need to search in the direction of the siblings.
[21] The greatest cell lesser than or equal to the search key is referred to as its maximum lower bound.

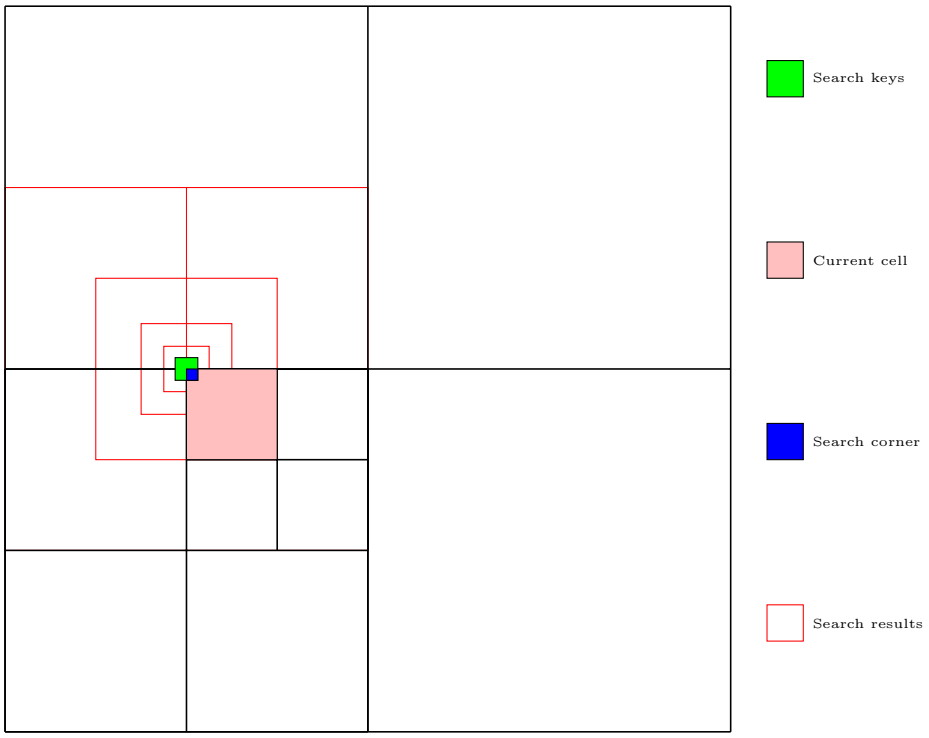


Figure 3.8: To find neighbors coarser than the current cell, we first select the finest cell at the far corner. The far corner is the one that is not shared with any of the current cell's siblings. The neighbors of this corner cell are determined and used as the search keys. The search returns the greatest cell lesser than or equal to the search key. The possible candidates in a complete linear quadtree, as shown, are ancestors of the search key.

no coarser neighbor can exist in that direction. This idea can be extended to incomplete linear octrees (including multiply connected domains). In this case, the result of a search is ignored if it is not an ancestor of the search key.

Ripple propagation

A variant (Algorithm 10) of the *prioritized ripple propagation* algorithm first proposed by Tu et al. [106], modified to work with linear octrees, is used to balance the boundary leaves. The algorithm selects all leaves at a given level (successively decreasing levels starting with the finest), and searches for neighbors coarser than itself. A list of balancing descendants[22] for neighbors that violate the balance condition are stored. At the end of each level, any octant that violated the balance condition is replaced by a complete linear subtree. This subtree can be obtained either by using the sequential version of Algorithm 5 or by using Algorithm 11, which is a variant of Algorithm 8. Both the algorithms perform equally well.[23]

One difference with earlier versions of the ripple propagation algorithm is that our version works with incomplete domains. In addition, earlier approaches [9, 106, 108] have used pointer-based representations of the local octree, which incurs the additional cost of constructing the pointer-based tree from the linear representation and also increases the memory footprint of the octree as 9 additional pointers[24] are required per octant. The work and storage costs incurred for balancing using the proposed algorithm to construct n balanced octants are $O(n \log n)$ and $O(n)$, respectively. This is true irrespective of the domain, including domains that are not simply connected.

[22] Balancing descendants are the minimum number of descendants that will balance against the octant that performed the search.

[23] We indicate which algorithms are parallel and which are sequential. In our notation the sequential algorithms are sometimes invoked with a distributed object: it is implied that the input is the local instance of the distributed object.

[24] One pointer to the parent and eight pointers to its children.

Algorithm 10 - `Ripple`
RIPPLE PROPAGATION ON INCOMPLETE DOMAINS (SEQUENTIAL)

Input: L, `a sorted incomplete linear octree.`
Output: W, `a balanced incomplete linear octree.`
Work: $O(n \log n)$, `where` $n =$ `len`(L).
Storage: $O(n)$, `where` $n =$ `len`(L).

```
1.   W ← L

2.   for l ← D_max to 3
3.       T, R ← ∅
4.       for each w ∈ W
5.           if L(w) = l
6.               K ← search_keys(w) ( Section 3.2.5 )
7.               (B, J) ← maximum_lower_bound (K, W)
                          (J is the index of B in W)
8.               for each (b, j) ∈ (B, J) | (∃ k ∈ K | b ∈ {A(k)})
9.                   T[j] ← T[j] + ({N^s(w, (l − 1))} ∩ {D(b)})
10.              end for
11.          end if
12.      end for
13.      for i ← 1 to len(W)
14.          if T[i] ≠ ∅
15.              R ← R+ CompleteSubtree(W[i], T[i]) ( Algorithm 11 )
16.          else
17.              R ← R + W[i]
18.          end if
19.      end for
20.      W ← R
21.  end for
```

Algorithm 11 - `CompleteSubtree`
COMPLETING A LOCAL BLOCK (SEQUENTIAL)

Input: An octant, N, and a partial list of its descendants, L.
Output: Complete subtree, R.
Work: $O(n \log n)$, where $n = \text{len}(R)$.
Storage: $O(n)$, where $n = \text{len}(R)$.

```
1.   W ← L

2.   for l ← D_max to L(N) + 1
3.       Q ← {x ∈ W | L(x) = l}
4.       Sort(Q)
5.       T ← {x ∈ Q | S(x) ∉ T}
6.       for each t ∈ T
7.           R ← R + t + S(t)
8.           P ← P + S(P(t))
9.       end for
10.      P ← P + {x ∈ W | L(x) = l − 1}
11.      W ← {x ∈ W | L(x) ≠ l − 1}
12.      RemoveDuplicates(P)
13.      W ← W + P, P ← ∅
14.  end for

15.  Sort(R)
16.  R ← Linearise(R) ( Algorithm 9 )
```

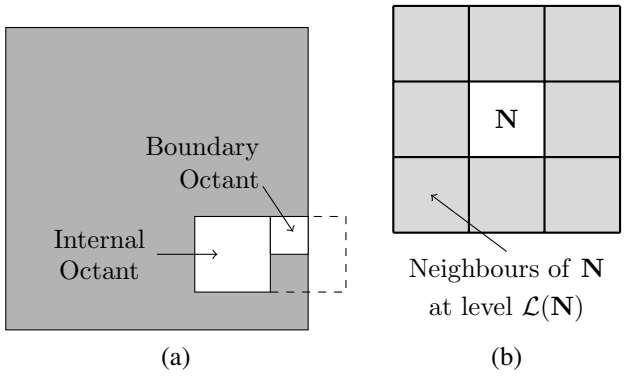


Figure 3.9: (a) A boundary octant cannot be finer than its internal neighbors, and (b) an illustration of an insulation layer around octant **N**. No octant outside this layer of insulation can force a split on **N**.

Insulation against the ripple-effect

An interesting property of complete linear octrees is that a boundary octant cannot be finer than its internal neighbors[25] (Figure 3.9a) [106]. So, if a node (at any level) is internally balanced then to balance it with all its neighboring domains, it is sufficient to appropriately refine the internal boundary leaves.[26] The interior leaves need not be refined any further. Since the interior leaves are also balanced against all their neighbors, they will not force any other octant to split. Hence, interior octants do not participate in the remaining stages of balancing.

Observe that the phenomenon with interior octants described above is only an example of a more general property:

Definition 2 *For any octant, N, in the octree, we refer to the union of the domains occupied by its potential neighbor's at the same level as N* ($\mathcal{N}(N, \mathcal{L}(N))$) *as the insulation layer around octant N. This will be denoted by* $\mathcal{I}(N)$.

Property 10 *No octant outside the insulation layer around octant N can force N to split*

[25] A neighbor of a boundary octant that does not touch the boundary is referred to as an internal neighbor of the boundary octant.

[26] We refer to the descendants of a node that touch its boundary from the inside as its internal boundary leaves.

(Figure 3.9b).

This property allows us to decouple the problem of balancing and allows us to work on only a subset of nodes in the octree and yet ensure that the entire octree is balanced.

Balancing inter-processor boundaries

After the intra processor, and inter block boundaries are balanced, the inter processor boundaries need to be balanced. Unlike the internal leaves (Section 3.2.5), the octants on the boundary do not have any insulation against the ripple-effect. Moreover, a ripple can propagate across multiple processors. Most approaches to perform this balance have been based on extensions of the sequential ripple algorithm to a parallel case by performing parallel searches. In an earlier attempt we developed efficient parallel search strategies allowing us to extend our sequential balancing algorithms to the parallel case. Although this approach works well for small problems on a small number of processors, it shows suboptimal isogranular scalability as has been seen with other similar approaches to the problem [108]. The main reason is iterative communication. Although there are many examples of scalable parallel algorithms that involve iterative communication, they overlap communication with computation to reduce the overhead associated with communication [29, 91]. Currently, there is no method that overlap communication with computation for the balancing problem. Thus, any algorithm that uses iterative parallel searches for balancing octrees will have high communication costs.

In order to avoid parallel searches, the problem of balancing is decoupled. In other words, each processor works independently without iterative communication. To achieve this, two properties are used: `(1)` the only octants that need to be refined after the local balancing stage are the ones whose insulation layer is not contained entirely within the same processor. We will refer to them as the unstable octants. and `(2)` an artificial insulation layer (Property 10) for these octants can be constructed with little communication overhead (Section 3.2.5).

Note that although it is sufficient to build an insulation layer for octants that truly touch

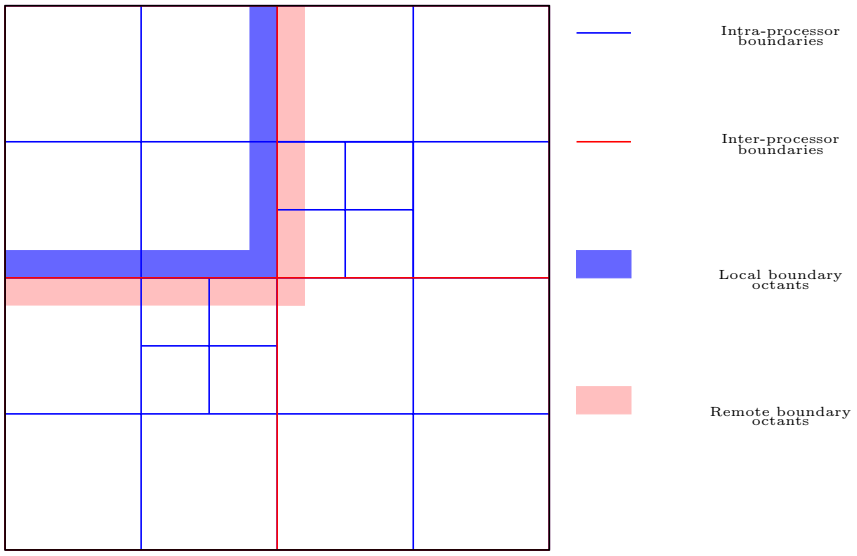


Figure 3.10: A coarse quadtree illustrating inter and intra processor boundaries. First, every processor balances each of its local blocks. Then, each processor balances the cells on its intra-processor boundaries. The octants that lie on inter-processor boundaries are then communicated to the respective processors and each processor balances the combined list of local and remote octants.

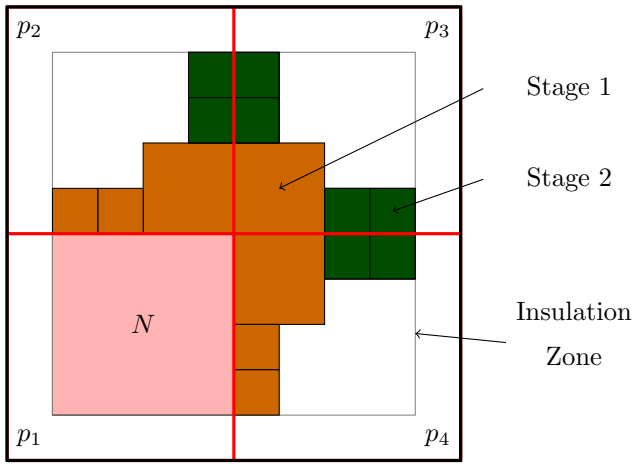


Figure 3.11: Communication for inter-processor balancing is done in two stages: First, every octant on the inter-processor boundary (Stage 1) is communicated to processors that overlap with its insulation layer. Next, all the local inter-processor boundary octants that lie in the insulation layer of a remote octant (N) received from another processor are communicated to that processor (Stage 2).

the inter-processor boundary, it is non-trivial to identify such octants. Moreover, even if it was easy to identify the true inter-processor boundary octants all unstable octants must participate in subsequent balancing as well. Hence, the insulation layer is built for all unstable octants as they can be identified easily. Since most of the unstable octants do touch the inter-processor boundaries, we will simply refer to them as inter-processor boundary octants in the following sections.

The construction of the insulation layer for the inter-processor boundary octants is done in two stages (Figure 3.11): First, every local octant on the inter-processor boundary (Figure 3.10) is communicated to processors that overlap with its insulation layer. These processors can be determined by comparing the local boundary octants against the global coarse blocks. In the second stage of communication, all the local inter-processor boundary octants that overlap with the insulation layer of a remote octant received from another processor are communicated to that processor. Octants that were communicated in the first stage are not communicated to the same processor again. For simplicity, Algorithm 12 only describes a naïve implementation for determining the octants that need to be communicated in this stage. However, this can be performed much more efficiently using the results of Lemma 2 and Lemma 3. After this two-stage communication, each processor balances the union of the local and remote boundary octants using the ripple propagation based method (Section 3.2.5). At the end only the octants spanning the original domain spanned by the processors are retained. Although there is some redundancy in the work, it is compensated by the fact that we avoid iterative communications. Section 3.2.5 gives a detailed analysis of the communication cost involved.

Lemma 2 *If octants a and $b > a$ do not overlap, then there can be no octant $c > b$ that overlaps a.*

Proof 2 *If a and c overlap, then either $a \in \{\mathcal{A}(c)\}$ or $a \in \{\mathcal{D}(c)\}$. Since $c > a$, the latter is a direct violation of Property 3 and hence is impossible. Hence, assume that $c \in \{\mathcal{D}(a)\}$. By Property 8, $c \leq \mathcal{DLD}(a)$. Property 9 would then imply that $b \in \{\mathcal{D}(a)\}$. Property 4 would*

then imply that a and b must overlap. Since, this is not true our initial assumption must be wrong. Hence, a and c can not overlap.

Lemma 3 *Let N be an inter-processor boundary octant belonging to processor q. If the $\mathcal{I}(N)$ is contained entirely within processors q and p, then the inter-processor boundary octants on processor p that overlap with $\mathcal{I}(N)$ and that were not communicated to q in the first stage, will not force a split on N.*

Proof 3 *Note that at this stage both p and q are internally balanced. Thus, N will be forced to split if and only if there is a true inter-processor boundary octant, a, on p touching an octant, b, on q such that $\mathcal{L}(a) > (\mathcal{L}(b)+1)$ and when b is split it starts a cascade of splits on octants in q that in turn force N to split. Since every true inter-processor boundary octant is sent to all its adjacent processors, a must have been sent to q during the first stage of communication.*

Algorithm 12 gives the pseudo-code for the overall parallel balancing.

Communication costs for parallel balancing

Although not all unstable octants are true inter-processor boundaries, it is easier to visualize and understand the arguments presented in this section if this subtle point is ignored. Moreover, since we only compare the communication costs associated with the two approaches (upfront communication versus iterative communication) and since the majority of unstable octants are true inter-processor boundary octants it is not too restrictive to assume that all unstable octants are true inter-processor boundary octants.

Let us assume that prior to parallel balancing there are a total of N octants in the global octree. The octants that lie on the inter-processor boundary can be classified based on the *degree of the face*[27] that they share with the inter-processor boundary. We use N_k to represent the number of octants that touch any m-dimensional face ($m \in [0, k]$) of the inter-processor boundary.

[27] *A corner is a 0-degree face, an edge is a 1-degree face and a face is a 2-degree face.*

Algorithm 12 BALANCING COMPLETE LINEAR OCTREES (PARALLEL)

Input: A distributed sorted complete linear octree, L.
Output: A distributed complete balanced linear octree, R.

1. $B \leftarrow$ BlockPartition(L) (Algorithm 2)
2. $C \leftarrow \emptyset$
3. **for each** $b \in B$
4. $C \leftarrow C +$ BalanceSubtree(b, $\{\{\mathcal{D}(b)\} \cap L\}$) (Algorithm 8)
5. **end for**
6. $D \leftarrow \{x \in C \mid \exists\ z \in \{\mathcal{I}(x)\} \mid \mathcal{B}(z) \neq \mathcal{B}(x)\}$
   (intra-processor boundary octants)
7. $S \leftarrow$ Ripple(D) (Algorithm 10)
8. $F \leftarrow (C - D) \cup S$
9. $G \leftarrow \{x \in S \mid \exists\ z \in \{\mathcal{I}(x)\} \mid$ rank(z) $\neq$ rank(x) $\}$
10. **for each** $g \in G$
11. **for each** $b \in B_{global} - B$
12. **if** $\{b \cap \mathcal{I}(g)\} \neq \emptyset$
13. **Send**(g, rank(b))
14. **end if**
15. **end for**
16. **end for**
17. $T \leftarrow$ Receive()
18. **for each** $g \in G$
19. **for each** $t \in T$
20. **if** $\{g \cap \mathcal{I}(t)\} \neq \emptyset$
21. **if** g was not sent to rank(t) in Step 10
22. **Send**(g, rank(t))
23. **end if**
24. **end if**
25. **end for**
26. **end for**
27. $K \leftarrow$ Receive()
28. $H \leftarrow$ Ripple($G \cup T \cup K$)
29. $R \leftarrow \{x \in \{H \cup F\} \mid \{B \cap \{x, \{\mathcal{A}(x)\}\}\} \neq \emptyset\}$
30. $R \leftarrow$ Linearise(R) (Algorithm 9)

Note that all vertex boundary octants are also edge and face boundaries and that all edge boundary octants are also face boundary octants. Therefore we have, $N \geq N_2 \geq N_1 \geq N_0$, and for $N \gg n_p$, we have $N \gg N_2 \gg N_1 \gg N_0$.

Although it is theoretically possible that an octant is larger than the entire domain controlled by some processors, it is unlikely for dense octrees. Thus, ignoring such cases we can show that the total number of octants of a d-tree that need to be communicated in the first stage of the proposed approach is given by

$$N_u = \sum_{k=1}^{d} 2^{d-k} N_{k-1}. \tag{3.2.1}$$

Consider the example shown in Figure 3.12. The domain on the left is partitioned into two regions, and in this case all boundary octants need to be transmitted to exactly one other processor. The addition of the additional boundary, in the figure on the right, does not affect most boundary nodes, except for the boundary octants that share a corner, i.e., a 0-dimensional face with the inter processor boundaries. These octants need to be sent to an additional 2 processors, and that is the reason we have a factor of 2^{d-k} in Equation 3.2.1. For the case of octrees, additional communication is incurred because of edge boundaries as well as vertex boundaries. Edge boundary octants need to be communicated to 2 additional processors whereas the vertex boundary octants need to be communicated to 4 additional processors (7 processors in all).

Now, we analyze the cost associated with the second communication step in our algorithm. Consider the example shown in Figure 3.11. Note that all the immediate neighbors of the octant under consideration (octant on processor 1 in the figure), were communicated during the first stage. The octants that lie in the insulation zone of this octant and that were not communicated in the first stage are those that lie in a direction normal to the inter-processor boundary. However, most octants that lie in a direction normal to the inter-processor boundary are internal octants on other processors. As shown in Figure 3.11, the only octants that lie in a direction normal to one inter-processor boundary and are also tangential to another inter-processor boundary are the ones that lie in the shadow of some

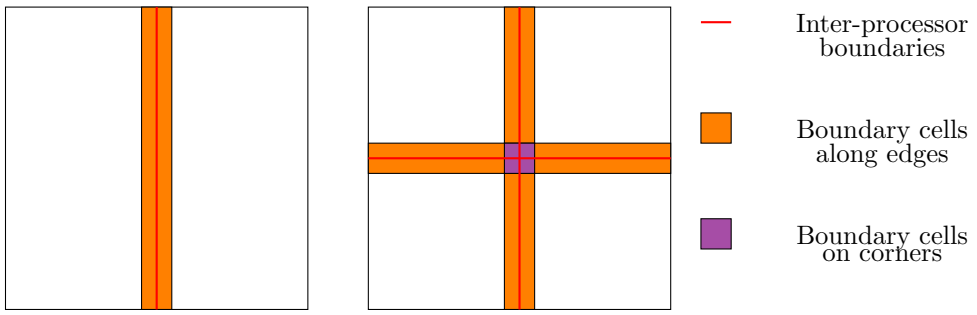


Figure 3.12: Cells that lie on the inter-processor boundaries. The figure on the left shows an inter-processor boundary involving 2 processors and the figure on the right shows an inter-processor boundary involving 4 processors.

edge or corner boundary octant. Therefore, we only communicate $O(N_1 + N_0)$ octants during this stage. Since $N \gg n_p$ and $N_2 \gg N_1 \gg N_0$ for most practical applications, the cost for this communication step can be ignored.

The minimum number of search keys that need to be communicated in a search based approach is given by

$$N_s = \sum_{k=1}^{d} 2^{k-1} N_{k-1}. \tag{3.2.2}$$

Again considering the example shown in Figure 3.12, each boundary octant in the figure shown on the left, generates 3 search keys, out of which one lies on the same processor. The other two need to be communicated to the other processor. The addition of the extra boundary, in the figure on the right, does not affect most boundary nodes, except for the boundary octants that share a corner, i.e., a 0-dimensional face with the inter processor boundaries. These octants need to be sent to an additional processor, and that is the reason we have a factor of 2^{k-1} in Equation 3.2.2. It is important to observe the difference between the communication estimates for upfront communication, 3.2.1, with that of the search based approach, 3.2.2. For large octrees,

$$N_u \approx N_2,$$

while,

$$N_s \approx 4N_2.$$

Note, that in arriving at the communication estimate for the search based approaches, we have not accounted for the additional octants created during the inter-processor balancing. In addition, iterative search based approaches are further affected by communication lag and synchronization. Our approach in contrast requires no subsequent communication.

In conclusion, the communication cost involved in the proposed approach is lower than that of search based approaches.[28]

3.3 Octree Meshing

By octree meshing we refer to the construction of a data structure on top of the linear octree that allows FEM type calculations. In this section, we describe how we construct the support data structures in order to perform the matrix-vector multiplications (`MatVec`) efficiently. The data structure is designed to be cache efficient by using a Morton ordering based element traversal, and by reducing the memory footprint using compressed representations for both the octree and the element-node connectivity tables. The algorithm for generating the mesh given a distributed, sorted, complete, balanced linear octree is outlined in Algorithm 13.

3.3.1 Computing the Element to Node Mapping

The nodes (vertices) of a given element are numbered according to the Morton ordering. An example is shown in Figure 3.13a. The 0-node of an element, the node with the smallest index in the Morton ordering, is also referred to as the *"anchor"* of the element. An octant's configuration with respect to its parent is specified by specifying the node that it shares with its parent. Therefore, a 3-child octant is the child that shares its third node with its parent. Nodes that exist at the center of a face of another octant are called face-hanging nodes. Nodes that are located at the center of an edge of another octant are called edge-hanging nodes.

[28] We are assuming that both the approaches use the same partitioning of octants.

Algorithm 13 OCTREE MESHING AND COMPRESSION

Input: *A distributed sorted complete balanced linear octree, L*
Output: *Compressed Octree Mesh and Compressed Octree.*

1. *Embed L into a larger octree, O, and add boundary octants.*
2. *Identify 'hanging' nodes.*
3. *Exchange 'Ghost' octants.*
4. *Build lookup tables for first layer of octants. (Section 3.3.1)*
5. *Perform 4-way searches for remaining octants. (Section 3.3.1)*
6. *Store the levels of the octants and discard the anchors.*
7. *Compress the mesh (Section 3.3.2).*

Since all nodes, except for boundary nodes, can be uniquely associated with an element (the element with its anchor at the same coordinate as the node) we use an interleaved representation where a common index is used for both the elements and the nodes. Because of this mapping, the input balanced octree does not have any elements corresponding to the positive boundary nodes. To account for this we embed the input octree in a larger octree with maximum depth $D_{max} + 1$, where D_{max} is the maximum depth of the input octree. All elements on the positive boundaries in the input octree add a layer of octants, with a single linear pass ($O(n/p)$), and a parallel sort ($O(n/p \log n/p)$). The second step in the the computation of the element-to-node mapping is the identification of hanging nodes. All nodes which are not hanging are flagged as being *nodes*. Octants that are the 0 or 7 children of their parent (a_0, a_7) can never be hanging; by default we mark them as *nodes* (see Figure 3.13a). Octants that are $3, 5, 6$ children of their parent (a_3, a_5, a_6) can only be *face hanging*, and their status is determined by a single negative search.[29] The remaining octants ($1, 2, 4$ children) are *edge hanging* and identifying their status requires three searches.

After identifying hanging nodes, we repartition the octree using the algorithm described

[29] By *"negative"* searches we refer to searches in the $-x$ or $-y$ or $-z$ directions. We use *"positive searches"* to refer to searches along the positive directions.

in Section 3.2.1 and all octants touching the inter-processor boundaries are communicated to the neighbouring processors. On the processors that receive such octants, these are the ghost elements, and their corresponding nodes are called ghost nodes. In our implementation of the `MatVec`, we do not loop over ghost elements recieved from a processor with greater rank and we do not write to ghost values. This framework gives rise to a subtle special case for *singular blocks*. A singular block is a block (output of the partition algorithm), which is also a leaf in the underlying fine octree (input to the partition algorithm). If the singular block's anchor is hanging, it might point to a ghost node and if so this ghost node will be owned by a processor with lower rank. This ghost node will be the anchor of the singular block's parent. We tackle this case while partitioning by ensuring that any singular block with a hanging anchor is sent to the processor to which the first child of the singular block's parent is sent. We also send any octant that lies between (in the Morton ordering) the singular block and its parent to the same processor in order to ensure that the relative ordering of the octants is preserved.

After exchanging ghosts, we perform independent sequential searches on each processor to build the element-to-node mappings. We present two methods for the same, an exhaustive approach (Section 3.3.1) that searches for all the 8 nodes for each element, and a more efficient approach that utilizes the mapping of its negative face neighbours (Section 3.3.1) to construct its own mapping.

Exhaustive searches to compute mapping

The simplest approach to compute the element-to-node mapping would be to search for the nodes explicitly using a parallel search algorithm, followed by the computation of global-to-local mappings that are necessary to manage the distributed data. However, this would incur expensive communication and synchronization costs. To reduce these costs, we chose to use *a priori* communication of "ghost" octants[30] followed by independent local searches on each processor that require no communication. A few subtle cases that can not be

[30]The use of blocks makes it easy to identify "ghost" octants.

identified easily during a priori communication are identified during the local searches and corrected for later.

For any element, all its nodes are in the positive direction (except the anchor). The exhaustive search strategy is as follows: Generate search keys at the location of the eight nodes of the element at the maximum depth and search for them in the linear octree. Since the linear octree is sorted, the search can be performed in $O(\log n)$. If the search result is a *node*, then the lookup table is updated. If we discover a hanging node instead, then a secondary search is performed to recover the correct (non-hanging) node index. As can be seen from Figure 3.13a, hanging nodes are always mapped to the corresponding nodes of their parent[31]. Unfortunately, secondary searches can not be avoided despite the identification of hanging nodes prior to searching. This is because only the element whose anchor is hanging knows this information and the other elements that share this vertex must first search and find this element in order to learn this information.

Using exhaustive searches we can get the mapping for most elements, but certain special cases arise for ghost elements. This case is illustrated in Figure 3.13b, where the ghost elements are drawn in red and the local elements are drawn in blue. Consider searching for the $+z$ neighbor of element a. Since a is a ghost, and we only communicate a single layer of ghost octants across processor boundaries, the node b will not be found. In such cases, we set the mapping to point to one of the hanging siblings of the missing node (c or d in this case). The most likely condition under which b in not found is when the $+z$ neighbor(s) is smaller. In this case, we know that at least one of the siblings will be hanging. Although the lookup table for this element is incorrect, we make use of the fact that the lookup table points to a non-existent node, and the owner of the true node simply copies its data value to the hanging node locations, thereby ensuring that the correct value is read. This case can only happen for ghost elements and need not be done for local elements. In addition, we occasionally observe cases where neither the searched node nor any of its siblings is found. Such cases are flagged and at the end of the lookup table construction, a parallel search is done to obtain the missing nodes directly from the processors that own them.

[31] The 2:1 balance constraint ensures that the nodes of the parent can never be hanging.

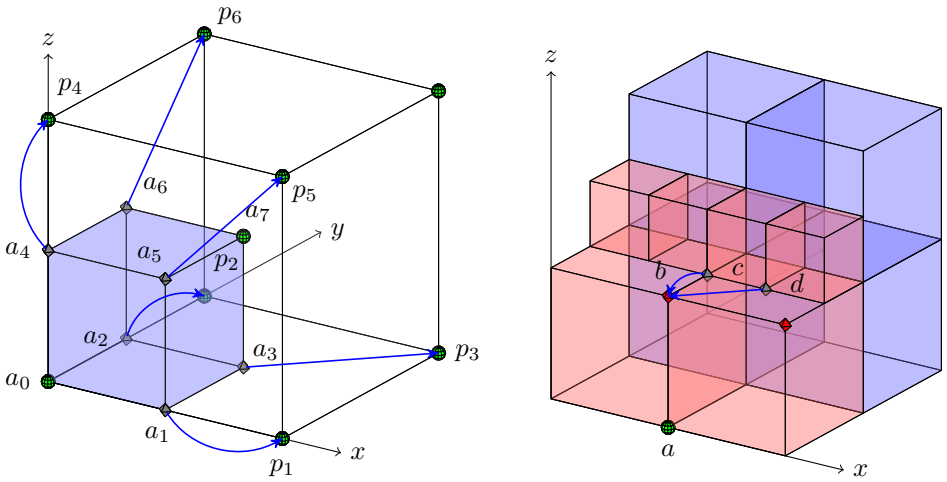
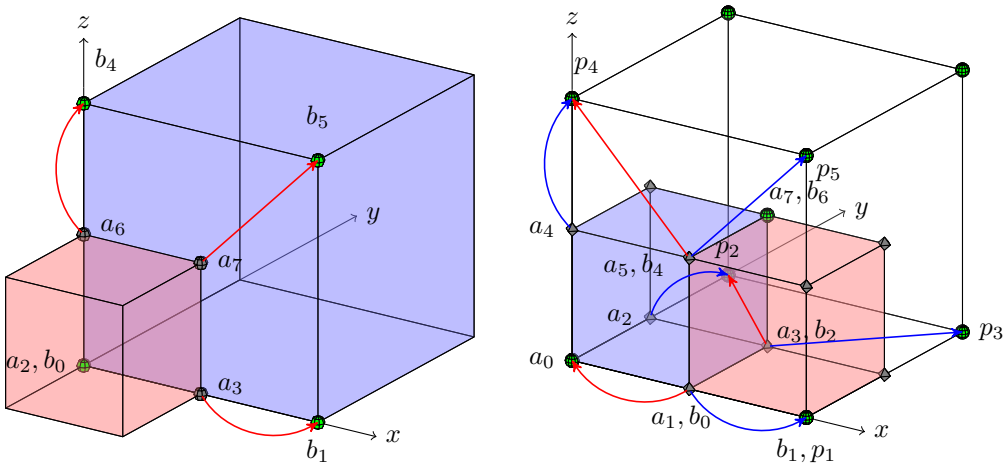


Figure 3.13: (a) Illustration of nodal connectivities required to perform conforming finite element calculations using a single tree traversal. Every octant has at least 2 non-hanging nodes, one of which is shared with the parent and the other is shared amongst all the siblings. The octant shown in blue (a) is a child 0, since it shares its zero node (a_0) with its parent. It shares node a_7 with its siblings. All other nodes, if hanging, point to the corresponding node of the parent octant instead. Nodes, a_3, a_5, a_6 are face hanging nodes and point to p_3, p_5, p_6, respectively. Similarly a_1, a_2, a_4 are edge hanging nodes and point to p_1, p_2, p_4. (b) The figure explains the special case that occurs during exhaustive searches of ghost elements. Element anchored at a, when searching for node b, will not find any node. Instead, one of the hanging siblings of b, (c, d) which are hanging will be pointed to. Since hanging nodes do not carry any nodal information, the nodal information for b will be replicated to all its hanging siblings during update for the ghosts.

Four-way searches to compute mapping

The exhaustive search explicitly searches for all nodes and in many cases is the only way to get the correct element to node mapping. However, it requires a minimum of 7 and a maximum of 13 searches per element. All searches are performed on a sorted list, and can be done in $O(\log n)$. In order to reduce the constants associated with the exhaustive search, we use the exhaustive search only for the first layer of octants (octants that do not have neighbours in the negative x, y and z directions). For all other octants, the lookup information can be copied from the elements in the negative directions. Each element in the negative x, y and z directions that shares a face with the current element, also shares 4 nodes. Therefore, by performing negative searches along these directions, we can obtain the lookup information for 7 out of the 8 nodes of an element. Only the last node, labeled a_7 in Figure 3.13a, cannot be obtained using a negative search and a positive search is required.

In order to get the mapping information using negative searches, we perform the search in the negative direction and check if the current element is a sibling of the element obtained via the negative search. If the element found by the search is not a sibling of the current element, then the lookup information can be copied via a mapping. For the example shown in Figure 3.14a, given the element b and searching in the $-y$ direction, we find a, then the node mapping is $(b_0, b_1, b_4, b_5) = (a_2, a_3, a_6, a_7)$. Corresponding mappings are $(b_0, b_2, b_4, b_6) = (a_1, a_3, a_5, a_7)$, and $(b_0, b_1, b_2, b_3) = (a_4, a_5, a_6, a_7)$, for negative searches along the x and z axes, respectively. Unfortunately, the mapping is a bit more complex if the negative search returns a sibling of the current element. If the node in question is not hanging, then we can copy its value according to the above mentioned mapping. However, if the node in question is hanging, then instead of the mapping, the corresponding node indices from element a are copied. This case is explained in Figure 3.14b, where we observe that if node a_1, b_0 is hanging, we need to use $b_0 = a_0$ and use $b_0 = a_1$ if it is a node.



(a) (b)

Figure 3.14: Computing element to node mapping using negative searches. (a) If the found octant (a) is not a sibling of the current octant (b), then the element to node mapping can be copied via the mapping $b_0 \leftarrow a_2$, $b_1 \leftarrow a_3$, $b_4 \leftarrow a_6$, and $b_5 \leftarrow a_7$. (b) In case the found octant (a) is a sibling of the current octant (b), then the mapping depends on whether or not the node in question is hanging. If the node is not hanging, then the same mapping as used in (a) can be applied. If the node is hanging, then the corresponding node indices for the found element are directly copied. For the case shown, $(b_0, b_2, b_4, b_6) \leftarrow (a_0, a_2, a_4, a_7) = (p_0, p_2, p_4, a_7)$.

3.3.2 Mesh Compression

One of the major problems with unstructured meshes is the storage overhead. In the case of the octree, this amounts to having to store both the octree and the lookup table. In order to reduce the storage costs associated with the octree, we compress both the octree and the lookup table. The sorted, unique, linear octree can be easily compressed by retaining only the offset of the first element and the level of subsequent octants. Storing the offset for each octant requires a storage of three integers (12 bytes) and a byte for storing the level. Storing only the level represents a 12x compression as opposed to storing the offset for every octant.

It is much harder to compress the element to node mapping, which requires eight integers for each element. In order to devise the best compression scheme, we first estimate the distribution of the node indices. The following lemma helps us analyze the distribution of the indices of the nodes of a given element.

Lemma 4 *The Morton ids of the nodes of a given element are greater than or equal to the Morton id of the element.*

Proof 4 *Let the anchor of the given element be* (x, y, z) *and let its size be h. In that case the anchors of the 8 nodes of the element are given by* $(x, y, z), (x + h, y, z), (x, y + h, z), (x + h, y + h, z) \cdots$ *. By the definition of the Morton ordering all of these except* (x, y, z) *are greater than the Morton id of the element. The node at* (x, y, z) *is equal to the Morton id of the element.*

Corollary 1 *Given a sorted list of Morton ids corresponding to the combined list of elements and nodes of a balanced linear octree, the indices of the 8 nodes of a given element in this list are strictly greater than the index of the element. Moreover, if the nodes are listed in the Morton order, the list of indices is monotonically increasing. If we store offsets in the sorted list, then these offsets are strictly positive.*

Based on these observations we can estimate the expected range of offsets. Let us assume a certain balanced octree, O, with n octants (elements and hanging-nodes) and

with maximum possible depth $D_{\max}$. Consider an element in the octree, o_i, whose index is i, $0 \leqslant i < n$. The offset of the anchor of this element is either i (if the anchor is not hanging) or $n_0 < i$. The indices for the remaining 7 nodes do not depend on octants with index less than i. In addition since the indices of the 7 nodes are monotonically increasing, we can store offsets between two consecutive nodes. That is, if the indices of the 8 nodes of an element, o_i, are $(n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7)$, we only need to store $(n_0 - i, n_1 - n_0, n_2 - n_1, n_3 - n_2, n_4 - n_3, n_5 - n_4, n_6 - n_5, n_7 - n_6)$. To efficiently store these offsets, we need to estimate how large these offsets can be. We start with a regular grid, i.e., a balanced octree with all octants at $D_{\max}$. Note that any octree that can be generated at the same value of $D_{\max}$ can be obtained by applying a series of local coarsening operations to the regular grid. Since we only store the offsets it is sufficient to analyze the distribution of the offset values for one given direction, say for a neighbor along the x-axis. The expression for all other directions are similar.

For $D_{\max} = 0$, there is only one octant and correspondingly the offset is 1. If we introduce a split in the root octant, $D_{\max}$ becomes 1, the offset increases by 2 for one octant. On introducing further splits, the offset is going to increase for those octants that lie on the boundaries of the original splits, and the general expression for the maximum offset can be written as offset $= 1 + \sum_{i=1}^{D_{\max}} 2^{d \cdot i - 1}$, for a d-tree. In addition, a number of other large offsets are produced for intermediate split boundaries. Specifically for a regular grid at maximum depth $D_{\max}$, we shall have $2^{d \cdot (D_{\max} - x)}$ octants with an offset of $1 + \sum_{i=1}^{x} 2^{d \cdot i - 1}$. As can be clearly seen from the expression, the distribution of the offsets is geometric. With the largest number of octants having small offsets.

For the case of general balanced octrees, we observe that any of these can be obtained from a regular grid by a number of coarsening operations. The only concern is whether the coarsening can increase the offset for a given octant. The coarsening does not affect octants that are greater than the current octant (in the Morton order). For those which are smaller, the effect is minimal since every coarsening operation reduces the offsets that need to be stored.

Golomb-Rice coding [27, 72] is a form of entropy encoding that is optimal for geometric distributions, that is, when small values are vastly more common than large values. Since, the distribution of the offsets is geometric, we expect a lot of offsets with small values and fewer occurrences of large offsets. The Golomb coding uses a tunable parameter M to divide an input value into two parts: q, the result of a division by M, and r, the remainder. In our implementation, the remainder is stored as a `byte`, and the quotient as a `short`. On an average, we observe one large jump in the node indices, and therefore the amortized cost of storing the compressed lookup table, is 8 `bytes` for storing the remainders, 2 `bytes` for the quotient, one `byte` for storing a flag to determine which of the 8 nodes need to use a quotient, and one additional `byte` for storing additional element specific flags. Storing the lookup explicitly would require 8 `ints`, and therefore we obtain a 3`x` compression in storing the lookup table.

3.4 Finite Element Computation on Octrees

In this section, we describe the evaluation of a `MatVec` with the global finite element 'stiffness' matrix. The `MatVec` refers to a function that takes a vector and returns another vector, the result of applying the discretized PDE operator to the the input vector. The key features of our framework for performing finite element computations are listed below.

- We do not store hanging nodes explicitly. They do not represent independent degrees of freedom in a FEM solution. A method to eliminate hanging nodes in locally refined quadrilateral meshes and yet ensure inter-element continuity by the use of special bilinear quadrilateral elements was presented in [112]. We extended that approach to three dimensions. If the i-th vertex of an element/octant is hanging, then the index corresponding to this node will point to the i-th vertex of the parent[32] of this element instead (Figure 3.13a). Thus, if a hanging node is shared between 2 or more elements, then in each element it might point to a different index.

[32]The 2:1 balance constraint ensures that the nodes of the parent can never be hanging.

- Since we eliminate hanging nodes in the meshing stage itself, we don't need to use projection schemes like those used in [2, 42, 43, 109] to enforce conformity. Hence, we don't need multiple tree traversals for performing each MatVec; instead, we perform a single traversal by mapping each octant/element to one of the pre-computed element types, depending on the configuration of hanging nodes for that element.

- To reduce the memory overhead, the linear octree is stored in a compressed form that requires only one byte per octant (the level of the octant). Even the element-to-node mappings can be compressed at a modest expense of uncompressing this on the fly while looping over the elements to perform the finite element MatVecs.

Below, we list some of the properties of the shape functions defined on octree meshes.

- The shape functions are not rooted at the hanging nodes.

- The shape functions are tri-linear.

- The shape functions assume a value of 1 at the vertex at which they are rooted and a value of 0 at all other non-hanging vertices in the octree.

- The support of a shape function can spread over more than 8 elements.

- If a vertex of an element is hanging, then the shape functions rooted at the other non-hanging vertices in that element do not vanish on this hanging node. Instead, they will vanish at the non-hanging vertex that this hanging node is mapped to. For example, in Figure 3.13a the shape function rooted at node a_0 will not vanish at nodes a_1, a_2, a_3, a_4, a_5 or a_6. It will vanish at nodes p_1, p_2, p_3, p_4, p_5, p_6 and a_7. It will assume a value equal to 1 at node a_0.

- A shape function assumes non-zero values within an octant if and only if it is rooted at some non-hanging vertex of this octant or if some vertex of the octant under consideration is hanging, say the i-th vertex, and the shape function in question is rooted at the i-th non-hanging vertex of the parent of this octant. Hence, for any octant there

are exactly eight shape functions that do not vanish within it and their indices will be stored in the vertices of this octant.

- The finite element matrices constructed using these shape functions are mathematically equivalent to those obtained using Projection schemes such as in [43, 108, 109].

To implement finite element MatVecs using these shape functions, we need to enumerate all the permissible hanging configurations for an octant. The following properties of 2:1 balanced linear octrees helps reduce the total number of permissible hanging configurations. Figure 3.13a illustrates these properties.

- Every octant has at least 2 non-hanging nodes and they are:
  - The node that is common to both this octant and its parent.
  - The node that is common to this octant and all its siblings.
- An octant can have a face hanging node only if the remaining nodes on that face are one of the following:
  - Edge hanging nodes.
  - The node that is common to both this octant and its parent.

The 'child number' of an octant is another important property that is used to handle hanging nodes in this framework. This is used both while implementing the MatVec for the finite element matrices and for implementing the inter-grid transfer operations. An octant's position with respect to its parent is identified by specifying the node that it shares with its parent. If an octant shares its k-th node with its parent, then it is said to have a 'child number' equal to k. For convenience, we use the Morton ordering to number the nodes of an octant. Thus, sorting the children of an octant in the Morton order is equivalent to sorting the children according to their child numbers. The child number of an octant is a function of the coordinates of its anchor and its level in the tree. Algorithm 14 is used to compute the child number of an octant.

Algorithm 14 FINDING THE CHILD NUMBER OF AN OCTANT

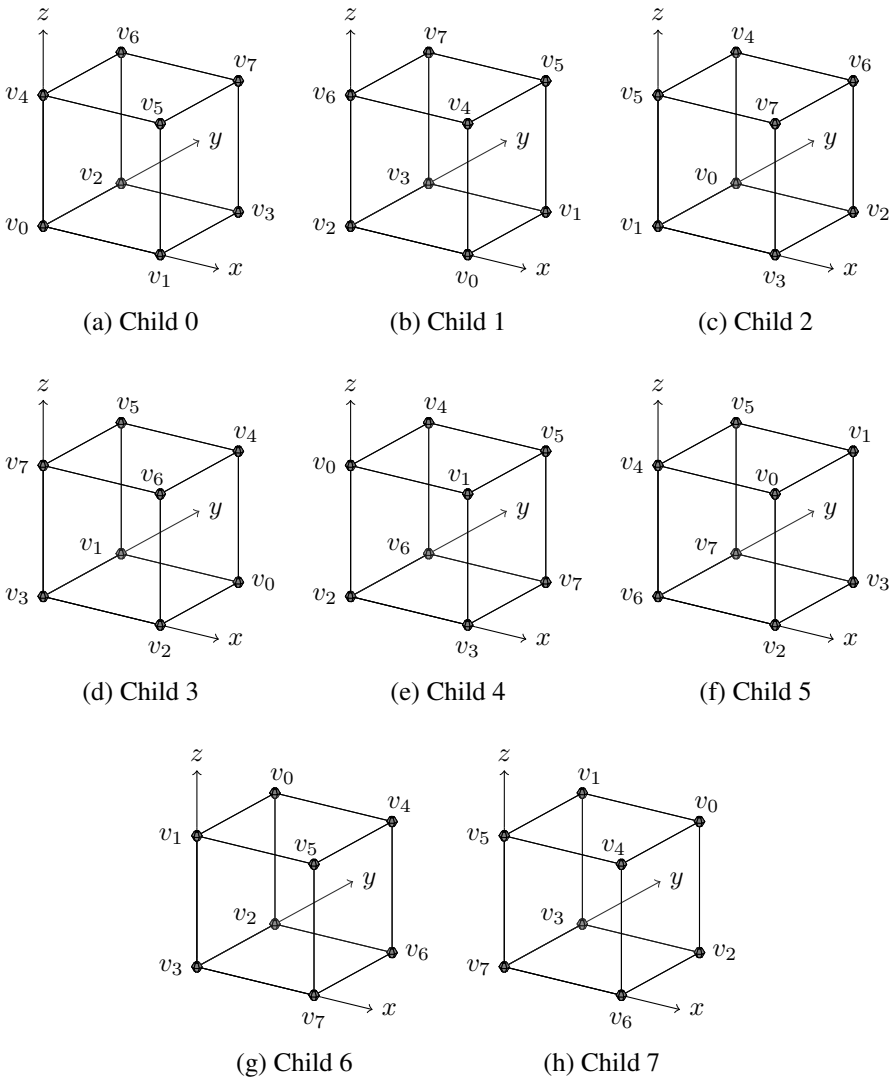
Input: The anchor (x,y,z) and level (d) of the octant and the maximum permissible depth of the tree (D_{max}).
Output: c, the child number of the octant.

1. $l \leftarrow 2^{(D_{max}-d)}$
2. $l_p \leftarrow 2^{(D_{max}-d+1)}$
3. $(i, j, k) \leftarrow (x, y, z) \bmod l_p$
4. $(i, j, k) \leftarrow (i, j, k)/l$
5. $c \leftarrow (4k + 2j + i)$

In each of the 8 child number based configurations (Figures 3.2 - 3.15h), v_0 is the node that this element shares with its parent and v_7 is the node that this element shares with all its siblings. For an element with child number k, v_0 will be the k-th node and v_7 will be the $(7 - k)$-th node. v_0 and v_7 can never be hanging. If v_3, v_5 or v_6 are hanging, they will be face-hanging and not edge-hanging. If v_3 is hanging, then v_1 and v_2 must be edge-hanging. If v_5 is hanging, then v_1 and v_4 must be edge-hanging. If v_6 is hanging, then v_2 and v_4 must be edge-hanging. After factoring in these constraints, there are only 18 potential hanging node configurations for each of the 8 'child number' configurations (Table 3.3).

3.4.1 Overlapping communication with computation

Every octant is owned by a single processor. However, the values of unknowns associated with octants on inter-processor boundaries need to be shared among several processors. We keep multiple copies of the information related to these octants and we term them 'ghost' octants. In our implementation of the finite element MatVec, each processor iterates over all the octants it owns and also loops over a layer of ghost octants that contribute to the nodes it owns. Within the loop, each octant is mapped to one of the above described hanging configurations. This is used to select the appropriate element stencil from a list of pre-computed stencils. We then use the selected stencil in a standard element based assembly



(a) Child 0 (b) Child 1 (c) Child 2

(d) Child 3 (e) Child 4 (f) Child 5

(g) Child 6 (h) Child 7

Figure 3.15: Illustration of the 8 child number configurations. In each of these configurations, v_0 is the node that this element shares with its parent and v_7 is the node that this element shares with all its siblings. For an element with child number k, v_0 will be the k-th node and v_7 will be the $(7-k)$-th node. v_0 and v_7 can never be hanging. If v_3, v_5 or v_6 are hanging, they will be face-hanging and not edge-hanging. If v_3 is hanging, then v_1 and v_2 must be edge-hanging. If v_5 is hanging, then v_1 and v_4 must be edge-hanging. If v_6 is hanging, then v_2 and v_4 must be edge-hanging. All the octants shown are oriented in the global coordinate system.

Element Type	Corresponding Hanging nodes
1	None
2	v_2
3	v_1
4	v_1 and v_2
5	v_4
6	v_4 and v_2
7	v_4 and v_1
8	v_4, v_1 and v_2
9	v_3, v_1 and v_2
10	v_3, v_4, v_1 and v_2
11	v_6, v_4 and v_2
12	v_6, v_4, v_1 and v_2
13	v_6, v_3, v_4, v_1 and v_2
14	v_5, v_4 and v_1
15	v_5, v_4, v_1 and v_2
16	v_5, v_3, v_4, v_1 and v_2
17	v_5, v_6, v_4, v_1 and v_2
18	v_5, v_6, v_3, v_4, v_1 and v_2

Table 3.3: The list of permissible hanging node configurations for any octant.

technique. Although the processors need to read ghost values from other processors they only need to write data back to the nodes they own and do not need to write to ghost nodes. Thus, there is only one communication step within each MatVec. We even overlap this communication with useful computation. We first loop over the elements in the interior of the processor domain since these elements do not share any nodes with the neighbouring processors. While we perform this computation, we communicate the ghost values in the background. At the end of the first loop, we use the ghost values that were recieved from the other processors and loop over the remaining elements.

The `MatVec` can be evaluated either by looping through the elements or by looping through the nodes. Due to the simplicity of the elemental loop, we have chosen to support elemental loops over nodal loops. We first loop over the elements in the interior of the processor domain since these elements do not share any nodes with the neighboring processors. While we perform this computation we communicate the values of the ghost nodes in the background. At the end of the first loop, we use the ghost values from other processors and loop over the remaining elements.

3.5 Summary of Octree Algorithms

The sequence of steps involved in solving a PDE on octrees is summarized below:

1. A 'sufficiently' fine[33] 2:1 balanced complete linear octree is constructed using the algorithms described in Section 3.2
2. The octree at is meshed using the algorithm described in Section 3.3.
3. The discrete system of equations is then solved using the conjugate gradient algorithm.

Table 3.4 gives the parallel time complexity of the various algorithmic components as a function of the problem size, N, and the number of processors, n_p.

[33]Here the term sufficiently is used to mean that the discretization error introduced is acceptable.

Algorithmic Component	Parallel Time Complexity
Octree Construction	$\mathcal{O}(\frac{N}{n_p}\log(\frac{N}{n_p}) + n_p \log n_p)$
2:1 Balancing	$\mathcal{O}(\frac{N}{n_p}\log(\frac{N}{n_p}) + n_p \log n_p)$
Block Partition	$\mathcal{O}(\frac{N}{n_p} + n_p)$
Meshing	$\mathcal{O}(\frac{N}{n_p}\log(\frac{N}{n_p}) + n_p \log n_p)$
Matvecs	$\mathcal{O}(\frac{N}{n_p})$

Table 3.4: Complexity estimates for the various algorithmic components assuming a Hypercube network topology with $\theta(n_p)$ bandwidth. N is the size of the linear octree and n_p is the number of processors.

3.6 Similarity Measures on Octrees

Inter-modality image alignment is a fundamental step in medical image analysis. It is required to bring image data from different modalities to a common coordinate frame in order to accumulate information. It is usually presented as an optimization problem requiring the minimization of a certain objective function. Objective functions, or similarity measures based on information theoretic principles have been successful in aligning images from differing modalities. Mutual Information (MI) was proposed as an image similarity measure by Collignon [18], Viola [110] and Wells [116] and is widely used for rigid inter-modality registration. Several modifications have been proposed to make MI more robust and increase its capture range, including Normalized Mutual Information [95]. However, MI-based methods are very sensitive to the way the probability distributions are estimated and the accuracy of the estimated probability distributions have a great influence in the accuracy and robustness of the registration results [67].

A common assumption made in estimating the probability distribution is that each voxel

is an i.i.d. realization of a random variable. Therefore, the probability distributions (including the joint distribution) can be computed by using all voxels in the reference image and the corresponding voxels in the transformed subject image. In general this can be quite expensive to compute and several multi-resolution and sampling techniques have been proposed for faster estimation of the probability distribution. Downsampling, both uniform and random, have been used quite commonly to speed up the estimation of the distributions [67]. Nonlinear sampling techniques where the local sampling rate is proportional to the gradient magnitude have also been proposed [74].

In general, although these methods have performed quite well for registering different structural modalities (like CT and MR), they have been less successful in being able to register structural modalities to functional modalities,which is important for diagnosis and treatment planning applications. Functional modalities do not image all tissues and are therefore more sensitive to errors in the estimation of probability distributions. We shall use the example of registering Single Photon Emission Computed Tomography (SPECT) with CT images of the same patient to demonstrate this problem.

In this section we present a new method for rigid alignment of multi-modality images using an octree based partitioning of the reference image. Octrees allow us to partition the image into spatially adaptive regions (octants) such that homogeneous regions produce large octants. The octree allows us to define a non-linear sampling of the image that is proportional to the underlying image complexity. The samples thus obtained are closer to the usual i.i.d. assumption in that they tend to be less statistically inter-dependent, which in turn help us obtain more accurate estimates of entropy. The MI is the sum of the entropy of the subject image and the entropy of subject conditional on the target. Consequently, improved entropy estimates provide better estimates of MI.

In Section 3.6.1 we present a brief introduction to estimating entropy in images, and lay the foundation for our arguments in favor of the octree-based estimation of distributions, which is described in Section 3.6.2. Section 3.6.3 discusses the incorporation of the octree-based mutual information metric into a registration framework for inter-modality alignment of images, within a multi-resolution framework. Experimental results and comparisons

with other methods are provided in Section 3.7.

3.6.1 Estimating the entropy of images

Shannon's entropy [85] for a discrete random variable X with a probability distribution $\mathbf{mp}(X) = (p_1, \cdots, p_n)$, is defined as,

$$H(X) \stackrel{\Delta}{=} -\sum_{i=0}^{n} p_i \log p_i. \tag{3.6.1}$$

The definition can be extended to images by assuming the image intensities to be samples from a high dimensional signal. A common assumption made while defining entropy for images is that each voxel is an i.i.d. realization and that the underlying probability of pixel intensities can be estimated via the normalized histogram. The probability of a pixel having an intensity y_i, $p(y_i) = \text{hist}_Y(y_i)/d$, where $\text{hist}_Y(y_i)$ is the number of voxels in image Y with intensity y_i and d is the total number of voxels. Equation 3.6.1 can be then used to estimate the entropy of the image. This however does not capture the spatial complexity of the underlying image. For example, if we shuffle the pixels within the image, we will lose structure and be left with an image which is spatially random. However since the histogram is unchanged, the entropy is the same. This is demonstrated in Figure 3.16, where Figure 3.16(a) is highly structured and the red and blue blocks are placed at regular intervals. It can be seen that although the image in Figure 3.16(b) is produced from that in Figure 3.16(a) by adding random noise to the position of the objects in the scene, the entropy is unaffected. This happens because of our assumption that each voxel intensity is an independent sample. This is not true since voxel intensities depend on the underlying objects, and samples from the same object cannot be assumed to be independent. Also observe that the gradient based approaches will not be able to capture this difference either, because it is not affected by the spatial variation between the two configurations shown in Figures 3.16(a) and (b).

It is widely accepted that images can be successfully modeled as Markov random fields [47, 71]. Now, assuming that instead of an i.i.d. assumption, the samples form a Markov random field, then the error in the estimation of the entropy is lowered if the samples are

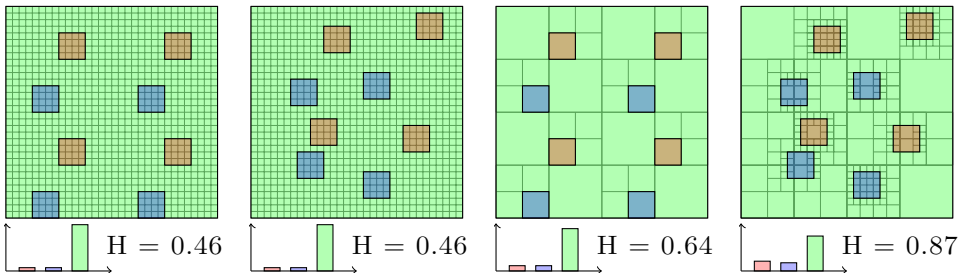


Figure 3.16: The difficulty in estimating the entropy of images. **(a)** An image with the corresponding histogram and entropy estimate, **(b)** An image with objects moved within the image to make it appear more *random* than (a). The standard estimate for the entropy does not change, since the histogram is still the same. **(c)** Octree-based entropy estimation for the image shown in (a), and **(d)** octree-based entropy estimation for the image shown in (b). Note that in this case the increase in entropy was captured by the octree.

largely independent of each other. Sabuncu et al. [74] suggest two non-linear sampling strategies based on the magnitude of the image gradient, in order to make the samples more independent. These suggested sampling methods, however, are for 2D images, expensive to compute and not easily extensible for 3D images. Algorithms have been proposed to make mutual information more robust by incorporating geometric information using gradients [68] and image salience [49]. However, the gradient captures only the local image complexity and it is not straightforward to extend this within a scale independent framework. This suggests using an adaptive sampling strategy, and adaptive image representations have been commonly used in non-linear image registration for representing the transformation [102] and for estimating local image similarities. However, to the best of our knowledge, it has not been used to estimate the global similarity of images.

3.6.2 Octrees based estimation of MI

If we can define a partitioning on the image that is spatially adaptive, and creates largely independent regions, we will be able to use the partition to define a better sampling strategy. Binary space partitions (BSP) [25] allow us to generate such spatially adaptive partitions of data by recursively subdividing a space into convex sets by hyperplanes. BSP trees can be used in spaces with any number of dimensions, but *quadtrees* [22] and *octrees* [58]

are most useful in partitioning 2D and 3D domains, respectively. Quadtrees and octrees use axis aligned lines and planes, respectively, instead of arbitrary hyperplanes and are efficient for the specific domains they are intended to work on.

The octree structure introduces non-stationarity in space [44]. This is important since the resulting octree is not shift-invariant and can create block artifacts. Such artifacts have been reported before [12, 50]. Approaches like generating octrees with overlapping leaves [40] could possibly be applied for estimating shift-invariant octrees. Making the samples shift-invariant is left for future work.

In our sampling method, the same number of samples are used per octant. In other words the density of octants specifies the sampling frequency. The octree-based entropy can be computed by using the following estimate of the probability distribution in (3.6.1),

$$p_i = \frac{\sum_x (T(x) \in \texttt{bin}(i))}{\sum_x 1}, \quad \forall x \in \mathrm{Oct}(T), \tag{3.6.2}$$

where $\texttt{bin}(\cdot)$ defines the histogram binning operator and $\mathrm{Oct}(T)$ is the octree computed for the template image T. In order to understand why an octree-based sampling appears to be better than uniform or gradient based sampling, consider the example discussed earlier, shown in Figure 3.16. We showed earlier that traditional sampling methods (both uniform and gradient based) estimate the same entropy for the images shown in Figures 3.16(a) and (b). However, the octree-based sampling is able to capture this difference, as seen in Figures 3.16(c) and (d). This is because the spatial complexity of the octree increases as a result of the change in the randomness of the scene. The octree captures the spatial complexity of the scene and consequently the entropy of a complex scene is higher as it has a denser octree. It is important to point out that the octree is not necessarily the best estimate of spatial complexity, since it is partial towards objects that are axis aligned and will not be able to capture such variations. A better estimate of the scene complexity would be a connectivity graph of segmented objects. This however would increase the complexity of both computing such a graph and also in using it during the registration. It would be difficult and computationally expensive to define transformations and interpolation functions on a connectivity graph of segmented tissues. The octree is a compromise that provides a

sufficiently accurate and robust estimate of the entropy (see Section 3.7.3) and is also easy to compute and manipulate. Interpolation and transformations (linear and non-linear) can be easily defined on the octree. Importantly, octree-representations are amenable to parallel computing, which can dramatically expedite the performance of algorithms that use it. In all the experiments reported in this section, the octree was computed only for the template image and each octant was sampled at the center of each octant in the template image. Although, it would be better to average over the entire octant instead of sampling at the center, we opted for the latter to improve the computational efficiency of the method.

3.6.3 Rigid inter-modality registration using adaptive MI

Given a template image, $T : \Omega \rightarrow \mathbb{R}^n$ and a subject image $S : \Omega \rightarrow \mathbb{R}^n$, where $\Omega \in \mathbb{R}^d$, the goal is to determine a rigid transformation χ that aligns the two images. The similarity measure is the metric that quantifies the degree of alignment and allows us to present the alignment problem as an optimization problem. We use the octree-based similarity measures as described in Section 3.6.2. We formulate the determination of the optimal transformation, $\hat{\chi}$, as an optimization problem:

$$\hat{\chi} = \arg \max_{\chi} I(T(x), S(\chi(x))), \quad \forall x \in \mathrm{Oct}(T), \tag{3.6.3}$$

where, $I(.;.)$ is the Mutual Information. Powell's multidimensional set method [70] is used to iteratively search for the maxima along each parameter using Brent's method [70]. In order to increase the robustness and to improve speed, we use a multi-resolution framework defined on the octree. The octree at lower (coarser) resolutions are generated by skipping all octants at the finer levels.

3.7 Results

In this section we present performance and scalability analyses for the construction, balancing and meshing of octrees. We also show results for the effectiveness of mutual information computed using octrees.

3.7.1 Octree construction and balancing

The performance of the proposed algorithms is evaluated by a number of numerical experiments, including fixed-size and isogranular scalabilty analysis. The algorithms were implemented in C++ using the MPI library. A variant of the sample sort algorithm was used to sort the points and the octants, which incorporates a parallel bitonic sort to sort the sample elements as suggested in [29]. PETSc [6] was used for profiling the code. All tests were performed on the Pittsburgh Supercomputing Center's TCS-1 terascale computing HP AlphaServer Cluster comprising of 750 SMP ES45 nodes. Each node is equipped with four Alpha EV-68 processors at 1 GHz and 4 GB of memory. The peak performance is approximately 6 Tflops, and the peak performance for the top-500 LINPACK benchmark is approximately 4 Tflops. The nodes are connected by a Quadrics interconnect, which delivers over 500 MB/s of message-passing bandwidth per node and has a bisection bandwidth of 187 GB/s. In our tests, we have used 4 processors per node wherever possible.

We present results from an experiment that we conducted to highlight the advantage of using the proposed two-stage method for intra-processor balancing. Also, we present fixed-size and isogranular scalability analysis results.

Test Data

Data of different sizes were generated for three different spatial distributions of points; Gaussian, Log-normal and Regular. The Regular distribution corresponds to a set of points distributed on a Cartesian grid. Datasets of increasing sizes were generated for all three distributions so that they result in balanced octrees with octants ranging from 10^6(1M) to 10^9(1B). All of the experiments were carried out using the same parameters: $D_{max} = 30$ and $N^p_{max} = 1$. Only the number and distribution of points were varied to produce the various octrees. The fixed size scalability analysis was performed by selecting the 1M, 32M, and 128M Gaussian point distributions to represent small, medium and large problems. We provide the input and output sizes for the construction and balancing algorithms in Table 3.5. The output of the construction algorithm is the input for the balancing algorithm.

Size	Gaussian				Log-Normal				Regular		
	Points	Balancing		Max. Level ($\mathcal{L}^*$)	Points	Balancing		$\mathcal{L}^*$	Points	Leaves	$\mathcal{L}^*$
		Leaves before	Leaves after			Leaves before	Leaves after				
1M	180K	607K	0.99M	14	180K	607K	0.99M	13	0.41M	0.99M	7
2M	361K	1.2M	2M	15	361K	1.2M	2M	14	2M	2M	7
4M	720K	2.4M	3.9M	14	720K	2.4M	3.9M	15	2.4M	4.06M	8
8M	1.5M	4.9M	8.0M	16	1.5M	4.9M	8.1M	16	3.24M	7.96M	8
16M	2.9M	9.7M	16M	16	2.9M	9.7M	16M	16	16.8M	16.8M	8
32M	5.8M	19.6M	31.9M	17	5.8M	19.6M	31.8M	17	19.3M	32.5M	9
64M	11.7M	39.3M	64.4M	18	11.7M	39.3M	64.7M	17	25.9M	63.7M	9
128M	23.5M	79.3M	0.13B	19	23.5M	79.4M	0.13B	19	0.13B	0.13B	9
256M	47M	0.16B	0.26B	19	47M	0.16B	0.26B	19	0.15B	0.26B	10
512M	94M	0.32B	0.52B	20	94M	0.32B	0.52B	20	0.17B	0.34B	10
1B	0.16B	0.55B	0.91B	21	0.16B	0.55B	0.91B	20	1.07B	1.07B	10

Table 3.5: Input and output sizes for the construction and balancing algorithms for the scalability experiments on Gaussian, Log-Normal, and Regular point distributions. The output of the construction algorithm is the input for the balancing algorithm. All the octrees were generated using the same parameters: $D_{max} = 30$ and $N^p_{max} = 1$; differences in the number and distributions of the input points result in different octrees for each case. The maximum level of the leaves for each case is listed. Note that none of the leaves produced were at the maximum permissible depth (D_{max}). This depends only on the input distribution. Regular point distributions are inherently balanced, and so we report the number of octants only once.

Comparison between different strategies for the local balancing stage

In order to assess the advantages of using a two-stage approach for local balancing over existing methods, we compared the runtimes on different problem sizes. Since the comparison was for different local-balancing strategies, it does not involve any communication and hence was evaluated on a shared memory machine. We compared our two-stage approach, discussed in Section 3.2.5, with two other approaches; the first approach is the prioritized ripple propagation idea applied on the entire local domain [108], and the second approach is to use ripple propagation in 2 stages, where the local domain is first split into coarser blocks[34] and ripple propagation is applied first to each local block and then repeated on the boundaries of all local blocks. Fixed size scalability analysis was performed to compare the above mentioned three approaches with problem sizes of 1, 4, 8, and 16 million octants. The results are shown in in Figure 3.17. All three approaches demonstrate good fixed size scalability, but the proposed two-stage approach has a lower absolute runtime.

Scalability analysis

In this Section, we provide experimental evidence of the good scalability of our algorithms. We present both fixed-size and isogranular scalability analysis. Fixed size scalability was performed for different problem sizes to compute the speedup when the problem size is kept constant and the number of processors is increased. Isogranular scalability analysis is performed by tracking the execution time while increasing the problem size and the number of processors proportionately. By maintaining the problem size per processor (relatively) constant as the number of processors is increased, we can identify communication problems related to the size and frequency of the messages as well as global reductions and problems with algorithmic scalability.

One of the important components in our algorithms is the sample sort routine, which has a complexity of $O(\frac{N}{n_p}\log\frac{N}{n_p} + n_p^2 \log n_p)$ if the samples are sorted using a serial sort.

[34] The same partitioning strategy as used in our two-stage algorithm was used to obtain the coarser blocks.

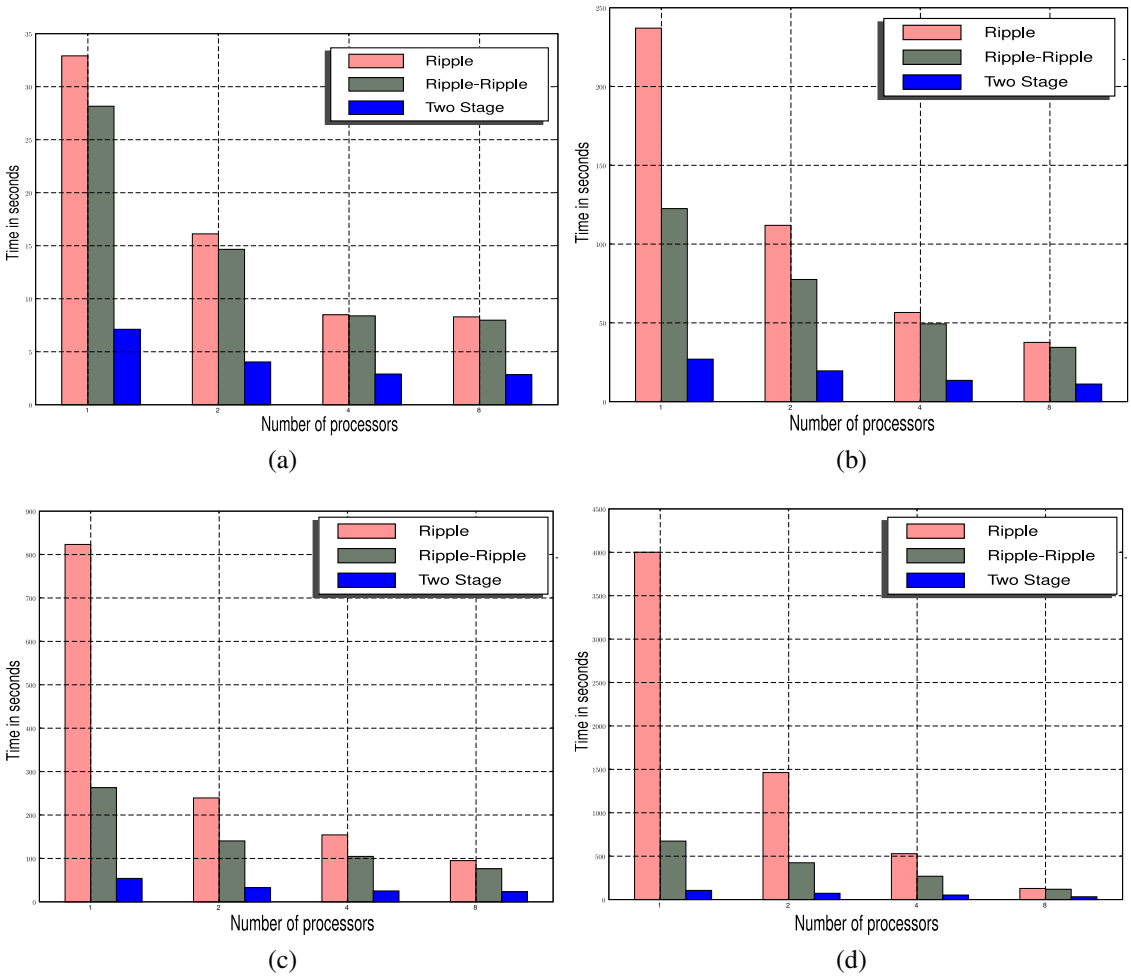


Figure 3.17: Comparison of three different approaches for balancing linear octrees (a) for a Gaussian distribution of 1M octants, (b) for a Gaussian distribution of 4M octants, (c) for a Gaussian distribution of 8M octants, and (d) for a Gaussian distribution of 16M octants.

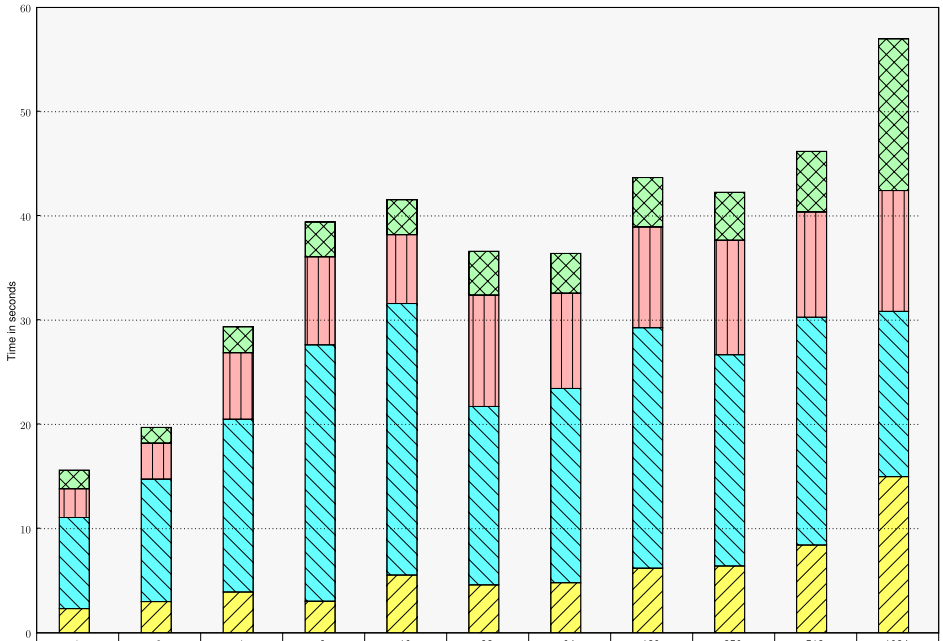
This causes problems when $O(N) < O(n_p^3)$ as the serial sort begins to dominate and results in poor scalability. For example, at $n_p = 1024$ we would require $\frac{N}{n_p} > 10^6$ to obtain good scalability. This presents some problems as it becomes difficult to fit arbitrarily large problems on a single processor. A solution, first proposed in [29], is to sort the samples using the parallel bitonic sort. This approach reduces the complexity of sorting to $O(\frac{N}{n_p} \log \frac{N}{n_p} + n_p \log n_p)$.

Isogranular scalability analysis was performed for all three distributions with an output size of roughly 1M octants per processor, for processor counts ranging from 1 to 1024. Wall-clock timings, speedup, and efficiency for the isogranular analysis for the three distributions are shown in Figures 3.18, 3.19, and 3.20.

Since the regularly spaced distribution is inherently balanced, the input point sizes were much greater for this case than those for Gaussian and Log-normal distributions. Both the Gaussian and Log-normal distributions are imbalanced; and in Table 3.5, we can see that, on average, the number of unbalanced octants is three times the number of input points and the number of octants doubles after balancing. For the regularly spaced distribution, we observe that in some cases the number of octants is the same as the number of input points (2M, 16M, 128M and 1B). These are special cases where the resulting grid is a perfect regular grid. Thus, while both the input and output grain sizes remain almost constant for the Gaussian and LogNormal distributions, only the output grain size remains constant for the Regular distribution. Hence, the trend for the regular distribution is a little different from those for the Gaussian and LogNormal distributions.

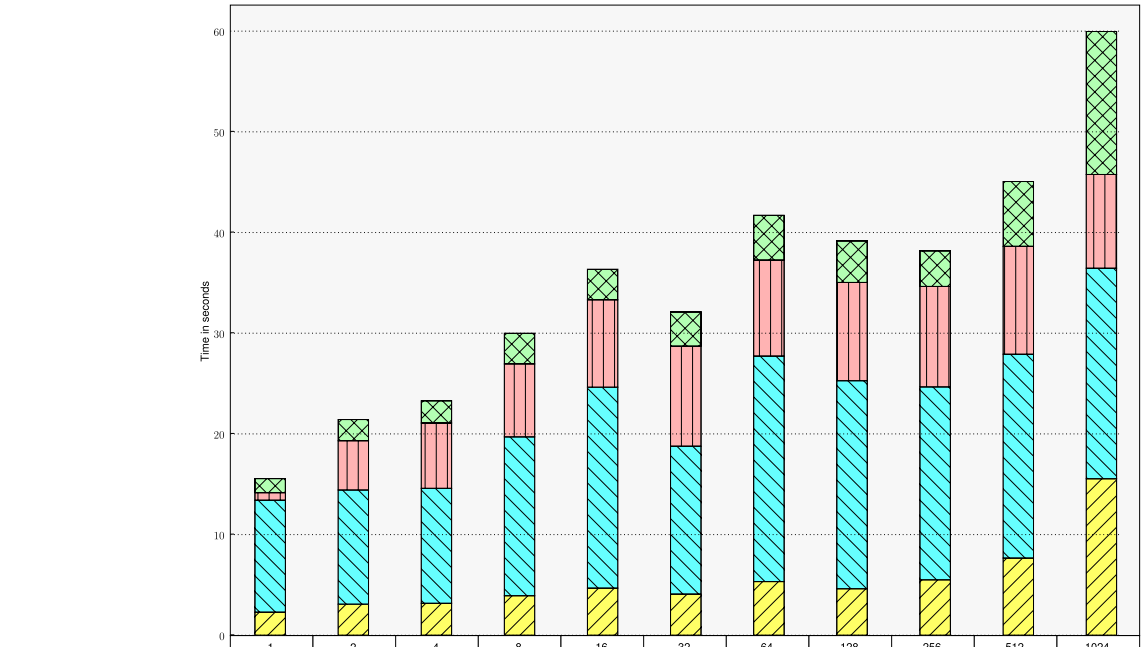
The plots demonstrate the good isogranular scalability of the algorithm. We achieve near optimal isogranular scalability for all three distributions (50s per 10^6 octants per processor for the Gaussian and Log-normal distributions and 25s for the regularly spaced distribution.).

Fixed size scalability tests were also performed for three problem set sizes, small (1 million points), medium (32 million points), and large (128 million points), for the Gaussian distribution. These results are plotted in Figures 3.21, 3.22 and 3.23.



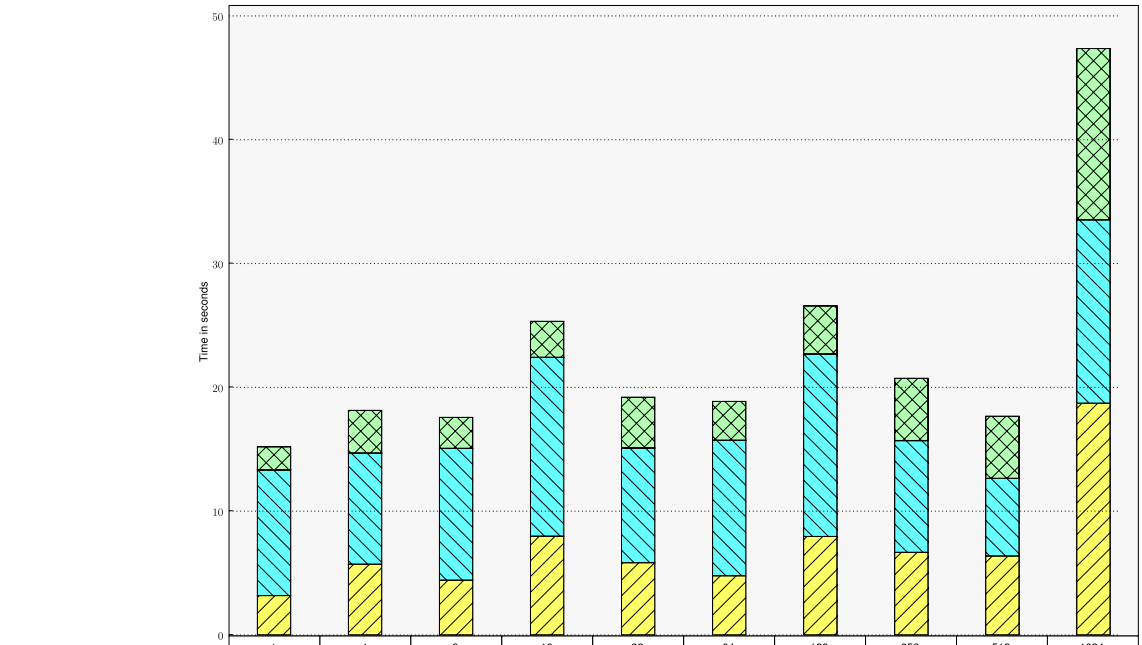
	1	2	4	8	16	32	64	128	256	512	1024
Construction	2.3	3.0	3.9	3.0	5.6	4.6	4.8	6.2	6.4	8.4	15
Internal Balance	8.7	11.8	16.6	24.6	26.0	17.1	18.6	23.1	20.3	21.8	15
Boundary Balance	2.8	3.4	6.4	8.4	6.6	10.7	9.1	9.7	11.0	10.1	11
Communication for Balance	1.8	1.5	2.5	3.3	3.4	4.2	3.8	4.7	4.6	5.8	14

Figure 3.18: Isogranular scalability for a Gaussian distribution of `1M` octants per processor. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `4` sections. From top to bottom, the sections represent the time taken for `(1)` communication (including related pre-processing and post-processing) during balance refinement `(Algorithm 12)`, `(2)` balancing across intra and inter processor boundaries `(Algorithm 10)`, `(3)` balancing the blocks `(Algorithm 8)`, and `(4)` construction from points `(Algorithm 6)`.



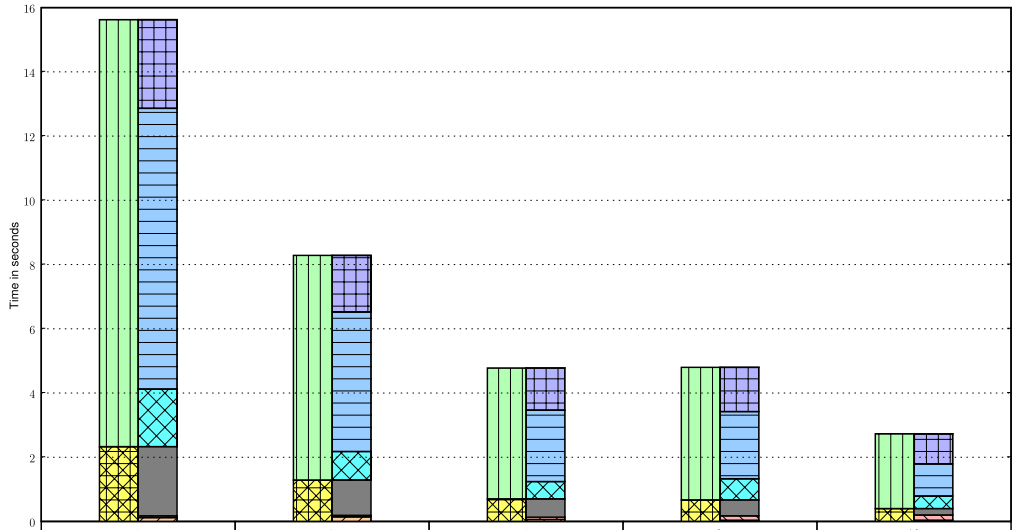
	1	2	4	8	16	32	64	128	256	512	1024
Construction	2.3	3.1	3.2	3.9	4.7	4.1	5.3	4.6	5.5	7.7	15
Internal Balance	11.1	11.3	11.4	15.8	20.0	14.7	22.4	20.7	19.2	20.2	20
Boundary Balance	0.7	4.9	6.5	7.2	8.7	9.9	9.5	9.8	10.0	10.7	9
Communication for Balance	1.4	2.1	2.2	3.1	3.0	3.4	4.5	4.1	3.5	6.4	14

Figure 3.19: Isogranular scalability for a Log-normal distribution of `1M` octants per processor. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `4` sections. From top to bottom, the sections represent the time taken for `(1)` communication (including related pre-processing and post-processing) during balance refinement `(Algorithm 12)`, `(2)` balancing across intra and inter processor boundaries `(Algorithm 10)`, `(3)` balancing the blocks `(Algorithm 8)`, and `(4)` construction from points `(Algorithm 6)`.



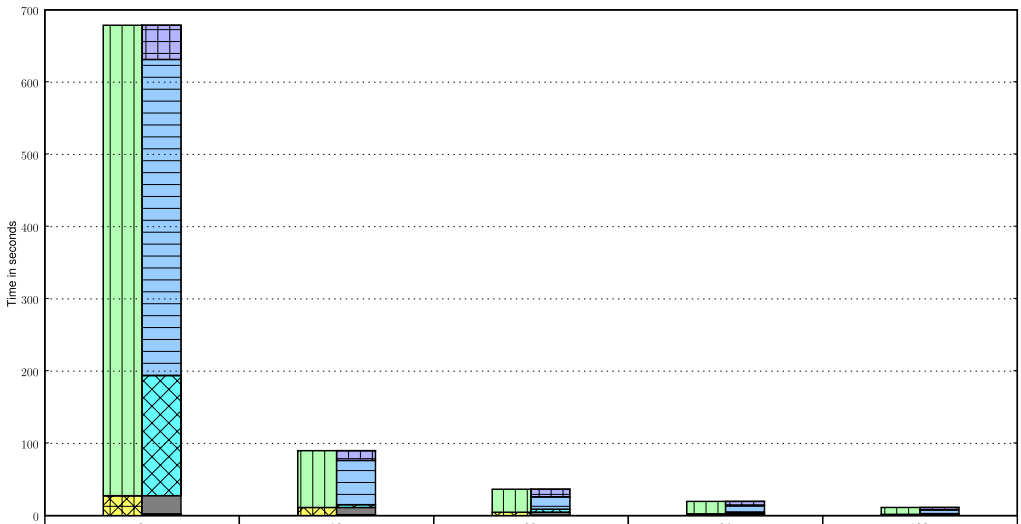
	1	4	8	16	32	64	128	256	512	1024
Construction	3.2	5.7	4.4	8.0	5.8	4.8	8.0	6.7	6.4	18
Internal Balance	10.1	9.0	10.6	14.4	9.3	10.9	14.7	9.0	6.3	14
Boundary Balance	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Communication for Balance	1.9	3.5	2.5	2.9	4.1	3.1	3.9	5.0	5.0	19

Figure 3.20: Isogranular scalability for a Regular distribution of `1M` octants per processor. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `4` sections. From top to bottom, the sections represent the time taken for `(1)` communication (including related pre-processing and post-processing) during balance refinement `(Algorithm 12)`, `(2)` balancing across intra and inter processor boundaries `(Algorithm 10)`, `(3)` balancing the blocks `(Algorithm 8)` and `(4)` construction from points `(Algorithm 6)`. While both the input and output grain sizes remain almost constant for the Gaussian and LogNormal distributions, only the output grain size remains constant for the Uniform distribution. Hence, the trend seen in this study is a little different from those for the Gaussian and LogNormal distributions.



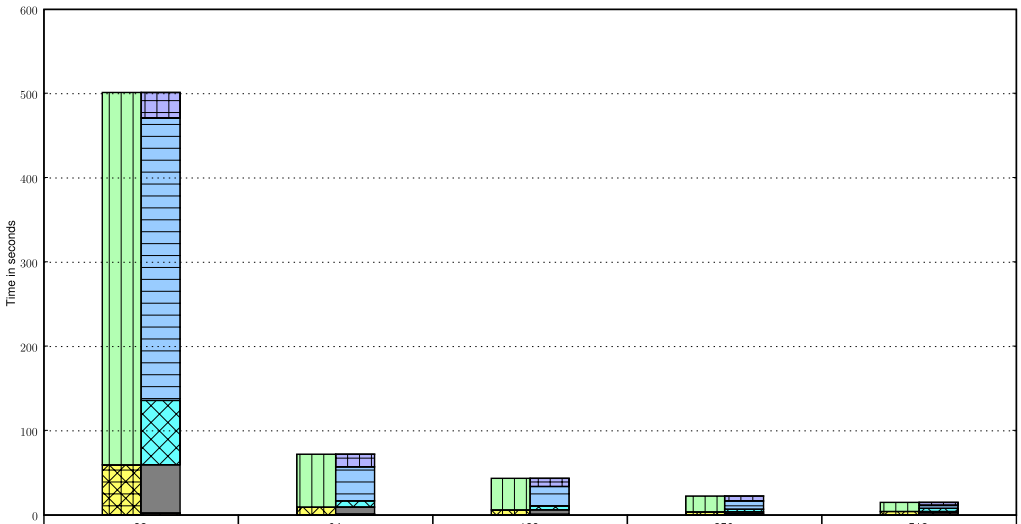
	1	2	4	8	16
Construction	2.3	1.3	0.7	0.7	0.4
Sort - Construction	0.1	0.1	0.1	0.0	0.0
BlockPartition	0.1	0.0	0.1	0.1	0.2
Balance	13.3	7.0	4.1	4.1	2.3
Commn - Balance	1.8	0.9	0.5	0.7	0.4
Internal Balance	8.7	4.3	2.2	2.1	1.0
Boundary Balance	2.8	1.8	1.3	1.4	0.9

Figure 3.21: Fixed size scalability for a Gaussian distribution of `1M` octants. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `2` columns, which are further subdivided. The left column is subdivided into `2` sections and the right column is subdivided into `6` sections. The top and bottom sections of the left column represent the total time taken for `(1)` balance refinement `(Algorithm 12)` and `(2)` construction `(Algorithm 6)`, respectively. From top to bottom, the sections of the right column represent the time taken for `(1)` balancing across intra and inter processor boundaries `(Algorithm 10)`, `(2)` balancing the blocks `(Algorithm 8)`, `(3)` communication (including related pre-processing and post-processing) during balance refinement, `(4)` local processing during construction, `(5)` `BlockPartition`, and `(6)` `Sample Sort`.



	8	16	32	64	128
Construction	27.7	11.3	4.6	2.5	1.7
Sort - Construction	1.7	0.8	0.4	0.2	0.2
BlockPartition	0.8	0.5	0.4	0.4	0.6
Balance	651.2	78.7	32.0	17.2	9.6
Commn - Balance	166.2	3.9	4.2	2.3	1.1
Internal Balance	437.5	61.2	17.1	9.5	5.0
Boundary Balance	47.4	13.6	10.7	5.3	3.4

Figure 3.22: Fixed size scalability for a Gaussian distribution of `32M` octants. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `2` columns, which are further subdivided. The left column is subdivided into `2` sections and the right column is subdivided into `6` sections. The top and bottom sections of the left column represent the total time taken for `(1)` balance refinement `(Algorithm 12)` and `(2)` construction `(Algorithm 6)`, respectively. From top to bottom, the sections of the right column represent the time taken for `(1)` balancing across intra and inter processor boundaries `(Algorithm 10)`, `(2)` balancing the blocks `(Algorithm 8)`, `(3)` communication (including related pre-processing and post-processing) during balance refinement, `(4)` local processing during construction, `(5)` `BlockPartition`, and `(6)` `Sample Sort`.



	32	64	128	256	512
Construction	59.7	9.5	6.2	3.9	4.3
Sort - Construction	2.0	0.8	0.4	0.6	1.4
BlockPartition	0.8	0.7	0.9	1.4	2.2
Balance	441.8	62.8	37.5	18.8	10.8
Commn - Balance	76.4	7.5	4.7	3.1	3.8
Internal Balance	335.1	40.3	23.1	9.7	3.7
Boundary Balance	30.3	15.0	9.7	5.9	3.3

Figure 3.23: Fixed size scalability for a Gaussian distribution of `128M` octants. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `2` columns, which are further subdivided. The left column is subdivided into `2` sections and the right column is subdivided into `6` sections. The top and bottom sections of the left column represent the total time taken for `(1)` balance refinement `(Algorithm 12)` and `(2)` construction `(Algorithm 6)`, respectively. From top to bottom, the sections of the right column represent the time taken for `(1)` balancing across intra and inter processor boundaries `(Algorithm 10)`, `(2)` balancing the blocks `(Algorithm 8)`, `(3)` communication (including related pre-processing and post-processing) during balance refinement, `(4)` local processing during construction, `(5)` `BlockPartition`, and `(6)` `Sample Sort`.

3.7.2 Octree Meshing

In this section we present numerical results for the tree construction, balancing, meshing and matrix vector multiplication for a number of different cases. The algorithms were implemented in C++ using the MPI library. PETSc [6] was used for profiling the code. We consider two point distribution cases: a regular grid one, to directly compare with structured grids; and a Gaussian distribution which resembles a generic non-uniform distribution. In all examples we discretized a variable-coefficient linear elliptic operator. We used piecewise constant coefficients for both the Laplacian and Identity operators. Material properties for the element were stored in an independent array, rather than within the octree data structure.

First we test the performance of the code on a sequential machine. We compare with a regular grid implementation with direct indexing (the nodes are ordered in lexicographic order along the coordinates). The results are presented in Table 3.6. We report construction times, and the total time for 5 matrix vector multiplications. Overall the code performs quite well. Both the meshing time and the time for performing the `MatVecs` are not sensitive to the distribution. The `MatVec` time is only 50% more than that for a regular grid with direct indexing, about five seconds for four million octants.

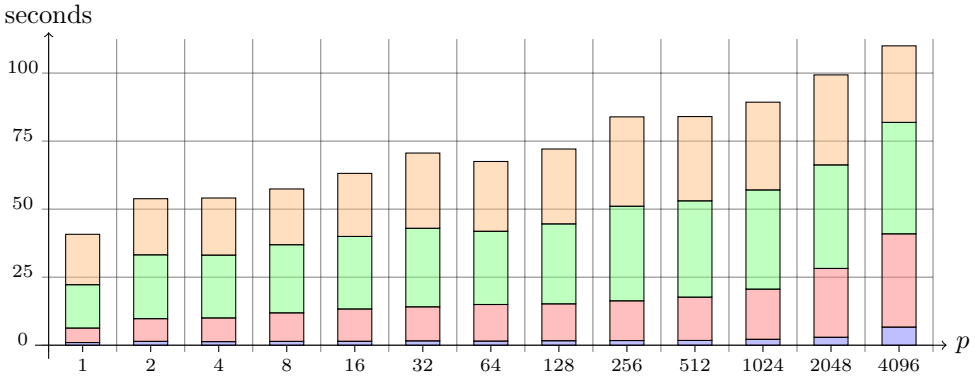
In the second set of experiments we test the isogranular scalability of our code. Again, we consider two point distributions, a uniform one and a Gaussian. The size of the input points, the corresponding linear and balanced octrees, the number of vertices, and the runtimes for the two distributions are reported in Figures 3.24 and 3.25. All the runs took place on a Cray XT3 MPP system equipped with 2068 compute nodes (two 2.6 GHz AMD Opteron and 2 GBytes of RAM per node) at the Pittsburgh Supercomputing Center. We observe excellent scalability for the construction and balancing of octrees, meshing and the matrix-vector multiplication operation. For example, in Figure 3.24 we observe the expected complexity in the construction and balancing of the octree (there is a slight growth due to the logarithmic factor in the complexity estimate) and we observe a roughly size-independent behavior for the matrix-vector multiplication.The results are even better for

Problem Size	Regular Grid MatVec	Octree Mesh			
		Uniform		Gaussian	
		Meshing	MatVec	Meshing	MatVec
256K	1.08	4.07	1.62	4.34	1.57
512K	2.11	8.48	3.18	8.92	3.09
1M	4.11	17.52	6.24	17.78	6.08
2M	8.61	36.27	11.13	37.29	12.33
4M	17.22	73.74	24.12	76.25	24.22

Table 3.6: The time to construct (**Meshing**) and perform 5 matrix-vector multiplications (**MatVec**) on a single processor for increasing problem sizes. Results are presented for Gaussian distribution and for uniformly spaced points. We compare with matrix-vector multiplication on a regular grid (no indexing) having the same number of elements and the same discretization (trilinear elements). We discretize a variable coefficient (isotropic) operator. All wall-clock times are in seconds. The runs took place on a 2.2 GHz, 32-bit Xeon box. The sustained performance is approximately 400 MFlops/sec for the structured grid. For the uniform and Gaussian distribution of points, the sustained performance is approximately 280 MFlops/sec.

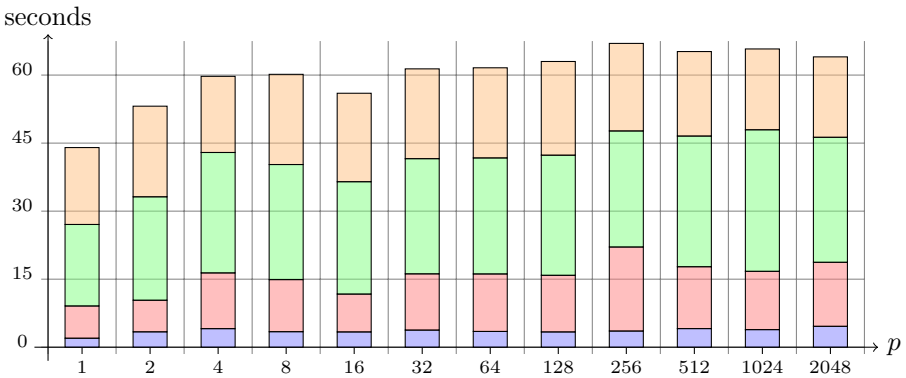
Problem Size	Uniform Distribution		Gaussian Distribution		
	Points	Octants	Points	Unbalanced	Balanced
1M	405.2K	994.9K	179.9K	607.3K	995.37K
2M	2.1M	2.1M	360.9K	1.21M	2M
4M	2.41M	4.07M	720K	2.43M	3.97M
8M	3.24M	7.96M	1.47M	4.91M	8.03M
16M	16.77M	16.77M	2.89M	9.69M	16.03M
32M	19.25M	32.53M	5.8M	19.61M	31.89M
64M	25.93M	63.67M	11.72M	39.28M	64.39M
128M	134.22M	134.22M	23.5M	79.29M	130.62M
256M	153.99M	260.24M	47M	158.63M	256.78M
512M	207.47M	509.39M	94M	315.19M	519.11M
1B	1.07B	1.07B	188M	635.08M	1.04B
2B	1.96B	2B	376M	1.26B	2.05B
4B	-	-	752M	2.52B	4.16B

Table 3.7: Input and output sizes for the construction and balancing algorithms for the iso-granular scalability experiment on Gaussian and uniform point distributions.



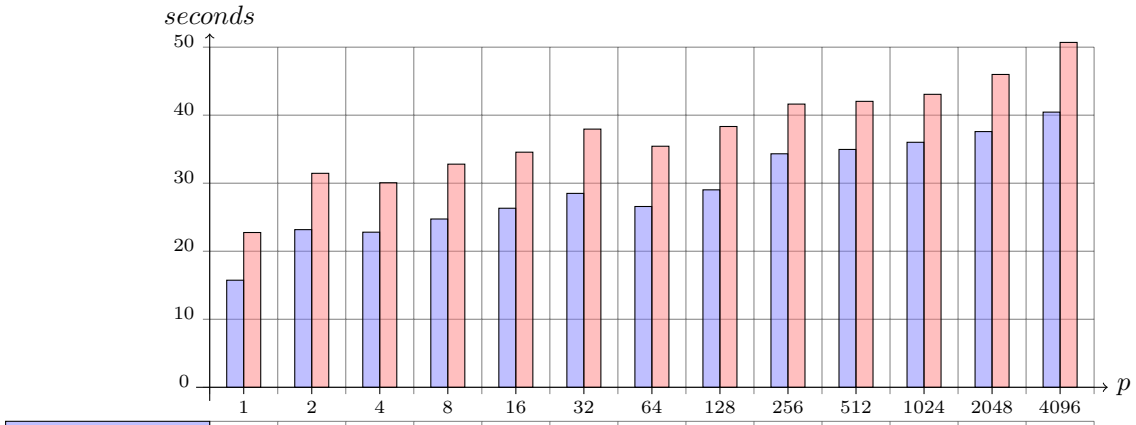
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
Octree Construction	0.99	1.41	1.32	1.40	1.44	1.60	1.51	1.65	1.72	1.75	2.17	2.92	6.68
Octree Balancing	5.34	8.39	8.75	10.55	11.94	12.57	13.52	13.62	14.67	15.99	18.53	25.42	34.44
Meshing	16.01	23.57	23.19	25.15	26.77	28.99	27.03	29.52	34.91	35.56	36.63	38.23	41.14
`MatVec` (5)	18.61	20.72	21.09	20.59	23.29	27.79	25.78	27.65	33.01	31.12	32.40	33.24	28.27
Points	180K	361K	720K	1.47M	2.89M	5.8M	11.7M	23.5M	47M	94M	188M	376M	752M
Unbalanced Octants	607K	1.2M	2.4M	4.9M	9.7M	19.6M	39.3M	79.3M	158M	315M	635M	1.26B	2.52B
Balanced Octants	996K	2M	4M	8M	16M	31.9M	64.4M	131M	257M	519M	1.04B	2.05B	4.16B
Independent Vertices	660K	1.3M	2.7M	5.2M	10.5M	21.5M	42M	87.8M	172M	339M	702M	1.36B	2.72B

Figure 3.24: Isogranular scalability for Gaussian distribution of `1M` octants per processor. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `4` sections. From top to bottom, the sections represent the time taken for `(1)` performing 5 Matrix-Vector multiplications, `(2)` Construction of the octree based mesh, `(3)` balancing the octree and `(4)` construction from points .



Octree Construction	2.01	3.44	4.15	3.47	3.42	3.83	3.52	3.42	3.62	4.16	3.93	4.67
Octree Balancing	7.21	7.06	12.46	11.64	8.46	12.57	12.86	12.65	18.78	13.83	13.03	14.31
Meshing	18.21	23.12	26.91	25.72	25.11	25.74	25.93	26.85	25.94	29.21	31.64	27.94
`MatVec` (5)	17.19	20.25	17.02	20.16	19.77	20.07	20.14	20.94	19.56	18.88	18.06	17.97
Points	1M	2.05M	3.94M	8M	15.8M	31.8M	63.5M	127.3M	248.8M	504.5M	1B	1.96B
Unbalanced Octants	1.06M	2.09M	4.14M	8.29M	16.1M	32.3M	64.3M	128M	250.5M	505.6M	1B	2B
Balanced Octants	1.06M	2.09M	4.14M	8.29M	16.1M	32.3M	64.3M	128.5M	250.5M	505.6M	1B	2B
Vertices	1.07M	2.15M	4.14M	8.29M	16.1M	32.3M	64.3M	128.5M	250.5M	505.6M	1B	2B

Figure 3.25: Isogranular scalability for uniformly spaced points with `1M` octants per processor. From left to right, the bars indicate the time taken for the different components of our algorithms for increasing processor counts. The bar for each processor is partitioned into `4` sections. From top to bottom, the sections represent the time taken for `(1)` performing 5 Matrix-Vector multiplications, `(2)` Construction of the octree based mesh, `(3)` balancing the octree and `(4)` construction from points.



	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
4-way	16.01	23.57	23.19	25.16	26.77	28.99	27.03	29.52	34.91	35.56	36.63	38.23	41.14
Exhaustive	23.14	31.99	30.58	33.36	35.15	38.60	36.04	38.99	42.34	42.75	43.80	46.77	51.56

Figure 3.26: Comparison of meshing times using exhaustive search with using a hybrid approach where only the first layer of octants uses exhaustive search and the rest use the 4-way search to construct the lookup tables. The test was performed using a Gaussian distribution of 1 million octants per processor. It can be seen that the 4-way search is faster than the exhaustive search and scales upto 4096 processors.

the uniform distribution of points in Figure 3.25, where the time for 5 matrix-vector multiplications remains nearly constant at approximately 20 seconds.

Finally, we compare the meshing time for the two search strategies presented in Section 3.3.1. The improvement in meshing time as a result of using the 4-way search is shown in Figure 3.26, for the Gaussian distribution. As can be seen, there is a significant improvement in meshing time at all processor counts.

3.7.3 Octree-based MI

In this section we describe experiments that were carried out to test the effectiveness of octree-based MI in the rigid registration of inter-modality images. We first describe the similarity profiles when an artificial transformation is introduced between two registered images. We compared the octree-based method with uniform sampling based estimation of mutual information. The first experiment was performed using simulated MR datasets obtained from the BrainWeb database [19]. The second experiment was performed with 13

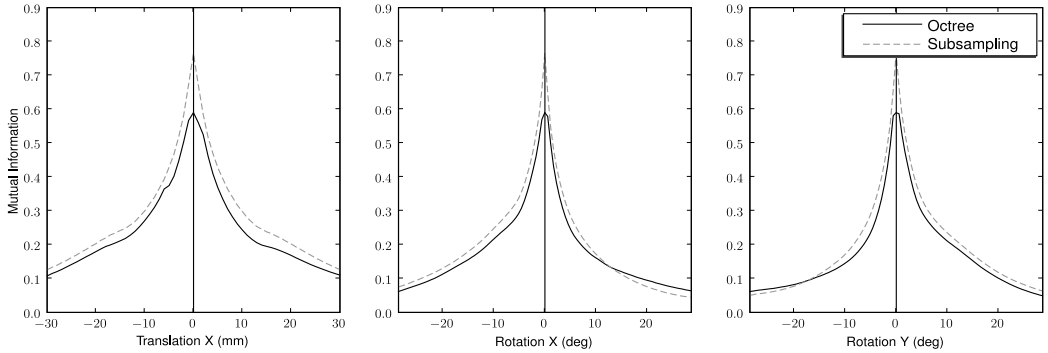


Figure 3.27: Comparison of the mutual information computed via uniform sampling (dotted lines) and using the proposed octree-based sampling (solid lines), on BrainWeb datasets. The plots shown are for a comparison between a T1-weighted (T1) image and a proton density (PD) image with 9% noise and 40% intensity non-uniformity.

CT datasets with corresponding SPECT images. These images were all acquired using a Siemens Symbia^{TM} T, a TruePoint SPECT-CT system and are assumed self registered. We analyzed the mutual information profiles while varying the transformation. The transformation parameters were varied one at a time, and the similarity profiles were plotted. The plots for translation along the x-axis, and for rotation about the x and y axes are shown in Figures 3.27 and 3.28, for T1-PD MR images and CT-SPECT images, respectively[35]. The profiles for translation and rotation along the other axes were similar. In all cases we compare the octree-based sampling with uniform sampling, where the total number of samples are similar. The octree reduced the number of samples by a factor of 8 on an average, therefore we subsampled by a factor of 2, along each direction, for the uniform sampling strategy, to have the same number of samples in both cases. As can be seen from Figure 3.27, both methods perform equally well on the BrainWeb datasets. Both sampling techniques have smooth curves with sharp peaks and very good capture ranges. However, when we look at the results from the CT-SPECT comparison, shown in Figure 3.28, we observe that the octree-based sampling performs much better. Although, both approaches have good profiles subject to translation, for the profiles subject to rotation, the uniform sampling approach exhibits a weak maxima at the optimal value with a very small capture

[35]The profiles generated from using all the voxels in the image were almost identical to those obtained by uniform subsampling, and are not presented here for clarity.

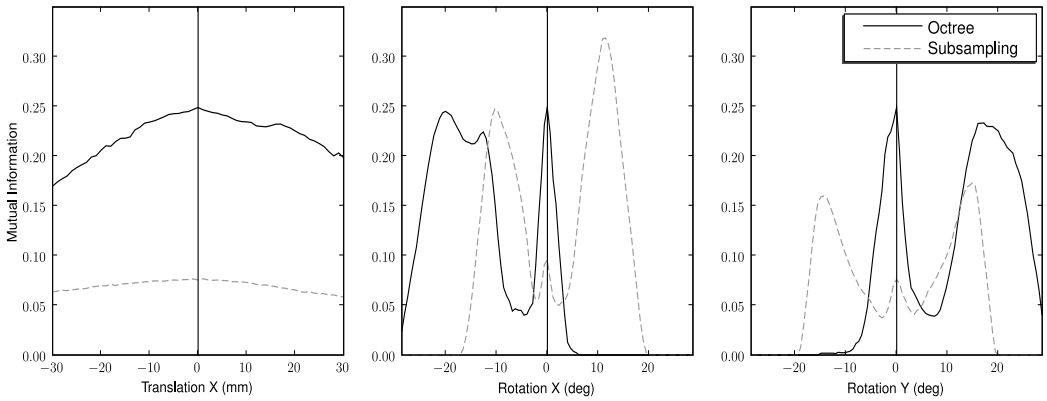


Figure 3.28: Comparison of the mutual information computed via uniform sampling (dotted lines) and using the proposed octree-based sampling (solid lines), with CT-SPECT datasets. The plots shown are for a comparison between a CT cardiac image (512×512×25) and a SPECT image ($128 \times 128 \times 128$).

	Uniform sampling			Octree-based		
Dataset	Success	Trans. error (mm)	Rot. error (deg)	Success	Trans. error (mm)	Rot. error (deg)
T1 - T2	82.4%	0.48 ± 0.63	0.17 ± 0.24	86.1%	0.53 ± 0.59	0.21 ± 0.19
T1 - PD	79.7%	0.57 ± 0.66	0.2 ± 0.33	81.3%	0.59 ± 0.62	0.22 ± 0.23
CT - SPECT	31.1%	0.73 ± 0.69	0.23 ± 0.28	68.5%	0.64 ± 0.57	0.21 ± 0.31

Table 3.8: Means and standard deviations of the registration errors for the different test cases.

range. In contrast the octree-based approach exhibits a strong maximum at the optimal value and also has a much larger capture range. The fact that the neighboring maxima in the vicinity of the optimum are lower further implies that it is likely that a multi-resolution approach can potentially be used to increase the capture range. The uniform sampling approach will in most cases converge to the wrong result since the neighboring maxima are much larger in that case.

Registration was performed on a number of datasets to quantitatively assess the performance of the octree-based mutual information within the registration framework. We selected a T1-weighted image with no noise and uniform intensity as the template image. T2-weighted and Proton Density (PD) images with varying levels of noise (0 – 9%) and intensity non-uniformity (0 – 40%) were registered to the template image, with a pseudo-random initial transform applied. The random transform was selected such that the initial

translation was at most half the size of the image (to ensure overlap) and the rotational components were less than 60°. The same set of pseudo-random transformations were used for both methods. The registration was considered successful if the final error was less than 2mm for the translational parameters and less than 2° for the rotational parameters. Similar experiments were performed for the CT-SPECT dataset and are summarized in Table 3.8. The error in the estimation of the translation and rotation parameters is calculated using only the cases in which a successful registration was performed. We can see from Table 3.8, that the octree-based sampling performs slightly better than uniform sampling in case of T1-T2 and T1-PD registration. We would like to emphasize that the success rate for the CT-SPECT registration is much better using the octree-based sampling as compared to uniform sampling, owing mainly to the broader capture range of the octree-based method. The average time to perform the registration was 13 seconds using the octree-based approach, 14 seconds for uniformly sampled approach and 85 seconds when using all voxels. All experiments were performed on an Intel Xeon 2.8GHz with 2GB of RAM. The time for the octree-based method includes the time to compute the octree.

3.8 Conclusions and future work

We have presented a set of algorithms for the parallel construction, balancing, and meshing of linear octrees. Our mesh data structure is interfaced with PETSc [6], thus allowing us to use its linear and non-linear solvers. Our target applications include elliptic, parabolic, and hyperbolic partial differential equations. We presented results that verify the overall scalability of our code. The overall meshing time is in the order of one minute for problems with up to four billion elements. Thus, our scheme enables efficient execution of applications which require frequent remeshing.

We need to consider the following factors to improve the performance of the proposed algorithms. In order to minimize communication costs, it is desirable to have as large coarse blocks as possible since the communication cost is proportional to the area of the inter-processor boundaries. However, too coarse blocks will increase the work for the local block

balancing stage (Section 3.2.5). If additional local splits are introduced, then the intra-block boundaries increase causing the work load for the first ripple balance to increase. The local balancing step of the algorithm can be made more efficient by performing the local balancing recursively by estimating the correct size of the block that can be balanced by the search-free approach. Such an approach should be based on low-level architecture details, like the cache size.

There are two important extensions: multigrid schemes, and higher-order discretizations. For the former, restriction and prolongation operators need to be designed, along with refinement and coarsening schemes. Higher-order schemes will require additional bookkeeping and longer lookup tables as the inter-element connectivity will increase.

Chapter 4

Inverse Problem

The problem of motion estimation from cardiac cine images is ill-posed, and relying solely on image similarity, even with very accurate similarity measures, is not sufficient to capture the true motion of the heart. Current cardiac motion estimation methods rely on image similarity measures to drive the motion estimation, and typically incorporate a regularizer to smooth the deformation field. Instead, we propose to maximize the image similarity, subject to the motion estimate constrained by a mechanical model of the heart, which has been discussed in Chapter 2.

The result of a 4D scan of a normal beating heart is an almost periodic sequence of N 3-D images,

$$I(x,t) = \{I_t(x), 0 \leq t < N\},$$

where I_0 is the end-diastolic image. We define the motion field as the transformation $\chi(x,t)$ defined over the image space. The transformation maps a point x in the end-diastole frame I_0 of the image to its corresponding point in frame I_t at time t. This is illustrated in Figure 4.1. The displacement field $U(x,t)$ defines the mapping from the coordinate system of the end-diastole image I_0 to the image at time t, I_t. The transformation and the displacement are related as, $\chi(x,t) = x + U(x,t)$.

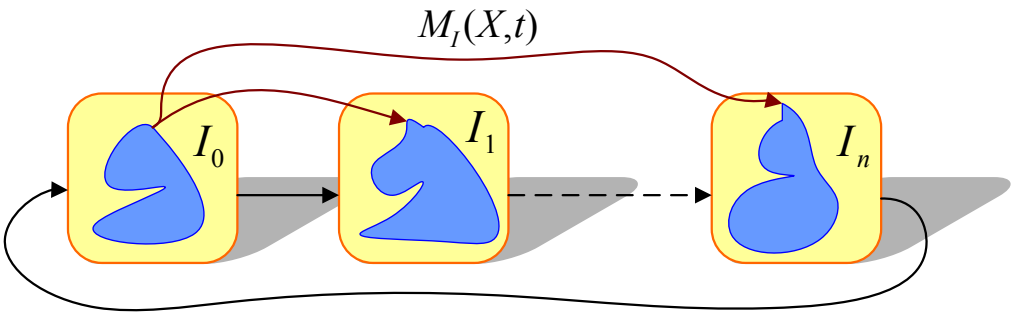


Figure 4.1: The formulation of the motion estimation problem.

4.1 Formulation of the inverse problem

The basic premise of our formulation is the following: The heart motion is induced by the active forces in the myocardium. If we knew the exact biomechanical model for the myocardial tissue (constitutive law, geometry, fiber orientations, material properties for the heart and surrounding tissues, endocardial tractions due to blood flow) and the active stretching time-space profile, then we could solve the so-called "*forward problem*" for the displacements of the myocardial tissue. Similarly, if we knew the displacements at certain locations in the myocardium, we could solve the so-called "*inverse problem*" to reconstruct active forces so that the motion due to the reconstructed forces matches the observed one. More generally, we have cine-MRI data but not the displacements. Our long term goal is to recover the displacements from cine-MR data by solving a biomechanically-constrained image registration problem.

In this context, an abstract formulation of the myocardium motion estimation problem is given by

$$\min_{u,s} \mathcal{J}(I_t, I_0, u) \quad \text{subject to} \quad C(u, s) = 0. \tag{4.1.1}$$

Here, $I_t := I_t(x,t)$ is the cine-MR image sequence with x, t denoting the space-time coordinates, $I_0 := I(x,0)$ is the initial frame (typically end-diastole), $u := u(x,t)$ is the *displacement* (motion), $s = s(x,t)$ is the active fiber contraction, and C is the forward problem

operator. Also, $\mathcal{J}$ is an image similarity measure functional. This is a classical PDE-constrained inverse problem [2]. Notice that there is no need for elastic, fluid, or any kind of regularization for u. It is constrained through the biomechanical model C.[1]

4.1.1 Objective function ($\mathcal{J}$)

To access the feasibility of the approach, we will simplify the problem. We will replace the highly non-convex image-similarity functional with a simpler one that is convex: we will assume that the true displacement field is partially observed (with noise) and we will try to recover the displacements everywhere. In practice, this information (partial displacement fields) can be extracted by the Cine-MR data by a preprocessing step, in which the point correspondences are manually computed for all frames, i.e., $d_j(t) := u(x_j, t)_{i=1}^M$ at M points. Then, the objective function is given by

$$\mathcal{J} := \int_0^1 (Qu - d)^2 \, d\, t := \int_0^1 \sum_{i=1}^{M} (u(x_j, t) - d_j(t))^2 \, d\, t, \tag{4.1.2}$$

where Q is the so called spatial observation operator.

4.1.2 Inverse problem

The inverse problem is stated by (4.1.1) where $\mathcal{J}$ is given by (4.1.2) and C is given by (2.6.2). By introducing Lagrange multipliers p, the first-order optimality conditions for (4.1.1) can be written as:

$$\begin{aligned} M\ddot{u}(t) + C\dot{u}(t) + Ku(t) + As(t) = 0, \quad \dot{u}(0) = u(0) = 0, \\ M\ddot{p}(t) - C\dot{p}(t) + Kp(t) + Q^T(Qu - d) = 0, \quad \dot{p}(1) = p(1) = 0, \\ A^T p(t) = 0. \end{aligned} \tag{4.1.3}$$

The second equation is the so-called "*adjoint problem*". Equation (4.1.3) consists of a system of partial-differential equations for u (cardiac motion), p (adjoints), and s (active fiber contraction). It is a 4D boundary value problem since we have conditions prescribed at both $t = 0$ and $t = 1$.

[1] However, one can show that the problem is ill-posed on s. Here we regularize by discretization of s.

4.1.3 Discretization and solution algorithms

We discretize the forward and adjoint problems in space using a Ritz-Galerkin formulation. We have developed a parallel data-structure and meshing scheme, discussed in [96]. The basis functions are trilinear, piecewise continuous polynomials. In time, we discretize using a Newmark scheme. The overall method is second-order accurate in space and time. The implicit steps in the Newmark scheme are performed using Conjugate Gradients combined with a domain-decomposition preconditioner in which the local preconditioners are incomplete factorizations. The solver and the preconditioner are part of the PETSc package [6].

For these particular choices of objective function and forward problem, the inverse problem (4.1.3) is linear in p, u, and s. We use a reduced space approach in which we employ a matrix-free Conjugate-Gradients algorithm for the Schur-complement of s—also called the (reduced) Hessian operator. Each matrix-vector multiplication with the Hessian requires one forward and one adjoint cardiac cycle simulation. Furthermore, one can show that the Hessian is ill-conditioned. Thus, the overall computational cost of the inversion is high. We are developing efficient multigrid schemes for the inverse problem. Details of this approach can be found in [2]. To reduce the computational cost for the calculations, we used a reduced-order model for s in which ψ is a product of B-splines in time and radial functions in space (Gaussians). This discretization not only does it allow acceleration of the inverse problem but it introduces a model error since the synthetic "ground truth" is generated using a full resolution fiber model and the inversion is done using the reduced resolution fiber model. This allows to perform preliminary tests on the sensitivity of our method to model errors.

4.1.4 Active Force Models

We consider two models for the active forces: the first being the generic case where arbitrary forces can be specified and the second model which accounts for the contractility of the myocytes. For the generic case the force term is given by,

$$F(t) = M\boldsymbol{f}(t), \tag{4.1.4}$$

where, $\boldsymbol{f}(t)$ are the nodal forces at time t.

To model the contractility of the myocytes given the fiber contractility $\boldsymbol{s}$ as a function of space and time and the myocyte orientation n, we define the active stretch tensor $U = 1 + \boldsymbol{s}\, n \otimes n$, whose divergence results in a distributed active force of the form $\mathrm{div}(\boldsymbol{s}\, n \otimes n)$. The force term in this case is given by,

$$F(t) = A\boldsymbol{s}(t), \quad A_{ij} = \int (n \otimes n)\nabla\phi_i\phi_j. \tag{4.1.5}$$

To reduce the computational cost for the calculations, we used a reduced-order model for $\boldsymbol{s}$ and $\boldsymbol{f}$ as a combination of B-splines bases in time and radial basis functions in space (Gaussians). The forces can then be written in terms of the B-spline basis, B and the radial basis, G, and the control parameters $\boldsymbol{\mu}$,

$$\boldsymbol{f}(\boldsymbol{x}, t) = \sum_{k=1}^{3} \boldsymbol{e}_k \sum_i G_i^k(\boldsymbol{x}) \sum_j \boldsymbol{\mu}_{ij} B_{ij}(t), \tag{4.1.6}$$

$$\boldsymbol{s}(\boldsymbol{x}, t) = \sum_i G_i(\boldsymbol{x}) \sum_j \boldsymbol{\mu}_{ij} B_{ij}(t). \tag{4.1.7}$$

In the matrix form this defines the parametrization matrix C is given by,

$$C_{xt,ij} = G_i(x)B_{ij}(t). \tag{4.1.8}$$

We can write the active forces in terms of the parametrization matrix C, as

$$F = M\boldsymbol{f} = MC\boldsymbol{\mu},$$

$$F = A\boldsymbol{s} = AC\boldsymbol{\mu}.$$

Additionally we need to define the transpose of A and C since they appear in the reduced gradient and reduced Hessian operators. The transpose of the myocyte parametrization matrix A is given by,

$$A_{ij}^{T} = \int \nabla\phi_j \phi_i (n \otimes n). \tag{4.1.9}$$

The transpose of the parametrization matrix C is given by,

$$C_{ij,xt}^{T} = G_i B_{ij} \tag{4.1.10}$$

4.2 Results

In this section, we describe experiments conducted to validate the motion estimation algorithm. We conducted experiments using synthetic models to determine the numerical correctness and robustness of the estimation with respect to noise and the estimation parameters. These are described in Section 4.2.1. We also tested the estimation using tagged MR datasets to validate the effectiveness of the method on real data. These experiments are described in Section 4.2.2.

4.2.1 Synthetic Datasets

In order to assess the fiber-orientation parametrized model of the forces (2.6.2), we use an ellipsoidal model of the left ventricle. The fiber orientations are generated by varying the elevation angle between the fiber and the short axis plane between $+60^\circ$ and -60° from the endocardium to the epicardium [76, 84]. The model along with the fiber orientations is shown in Figure 2.11. For this model we selected a Poisson's ratio $\nu = 0.45$ and a Young's modulus of 10 kPa for the myocardial tissue and 1 kPa for the surrounding tissue and ventricular cavity. Raleigh damping ($C = \alpha M + \beta K$) was used with parameters $\alpha = 0$ and $beta = 7.5 \times 10^{-4}$. In order to drive the forward model, we generated forces by propagating a synthetic activation wave from the apex to the base of the ventricles. Snapshots of this activation wave at different phases of the cardiac cycle are shown in Figure 2.14.

The number of time steps were set to be 50 and the data was interpolated to achieve this resolution. We used the generalized minimal residual method (GMRES) for the inverse

Basis Size	Relative Error $\|\cdot\|_2$	Time
2^3	1.31×10^{-1}	36 mins
4^3	5.67×10^{-2}	$\approx$ 5 hrs
4^3	1.12×10^{-1}	108 mins
8^3	9.66×10^{-2}	141 mins

Table 4.1: Error in recovery of activation for increasing number of radial basis functions. By changing the inversion solver accuracy, we can accelerate the calculation without compromising accuracy (e.g., the 4^3 calculation).

Observations	Relative Error $\|\cdot\|_2$
Full	5.36×10^{-2}
1/8	6.21×10^{-2}
1/64	8.51×10^{-2}

Table 4.2: Error in the recovery of activation with partial observations of the displacements. Errors are reported on the cylinder model for a grid size of 32 with 4^3 basis functions.

solver. The relative residual tolerance for the forward solver (CG) was set to 10^{-8} and to 10^{-4} for the inverse solver. For this mechanical model we selected a Poisson's ratio $\nu = 0.45$ and a Young's modulus of 10 kPa for the myocardial tissue and 1 kPa for the surrounding tissue and ventricular cavity. Raleigh damping ($C = \alpha M + \beta K$) was used with parameters $\alpha = 0$ and $beta = 7.5 \times 10^{-4}$.

For the inverse problem, we validate the error in the estimation of the activations for different degrees of parametrization using the radial basis. In all cases, the time dependence of the force was parametrized by five B-splines. The relative error in the estimation of the activation for a 64^3 grid was computed for spatial parametrizations of 2^3, 4^3 and 8^3 and is tabulated in Table 4.1. Ground-truth activations are generated using a wave propagating from the apex to the base of the ventricle 2.14. In addition, we investigated the error in the estimation when only partial observations are available. We compared estimations based on full and sparse observations with 12% and 6% samples against the analytical solutions. These results are tabulated in Table 4.2. In order to assess the sensitivity of the motion estimation framework, we estimated the motion for the synthetic model of the heart at a grid size of 64 with a radial basis parametrization of 4^3 by adding noise to the system. We added a 5% random error on the estimates of the fiber orientation and to the material

properties of the myocardium. In addition, we added a 1% noise to the true displacements. The system converged and the relative error, in the L^2 norm, increased from 5.67×10^{-2} to 9.43×10^{-2}.

4.2.2 Validation using tagged MR images

Data

We acquired cine MR sequences with and without myocardial tagging for 5 healthy volunteers in order to validate our motion estimation algorithm. The cine MR and tagged images were acquired on a Siemens Sonata 1.5T^{TM} scanner during the same scanning session at end-expiration, thus the datasets are assumed self-registered. Short and long axis segmented k-space breath-hold cine TrueFISP (SSFP) images were acquired for three healthy volunteers. The image dimensions were 156x192 pixels with a pixel size of 1.14x1.14 mm^2. A total of 11 slices were acquired with a slice thickness of 6mm and a slice gap of 2mm. For validation purposes, we also acquired grid tagged, segmented k-space breath-hold cine TurboFLASH images. The image dimensions were 156x192 pixels with a pixel size of 1.14x1.14 mm^2. The slice thickness was 8mm. Three short axis (apical, mid-ventricular and basal) and one long-axis images were acquired. The displacements (observations) at the tag intersection points within the myocardium were computed manually using the markTags GUI, which allows users to manually place points at the tag intersections. The program allows users to define point correspondences over time, which can then be used to drive the motion estimation algorithm. A screenshot of the markTags program is shown in Figure 4.2. An average of 70 tag intersection points were selected over the left and right ventricles on each plane resulting in around 300 observations in space. Three independent observers processed all five datasets to get three sets of observations. The average displacements were used as the true observations for the motion estimation algorithm.

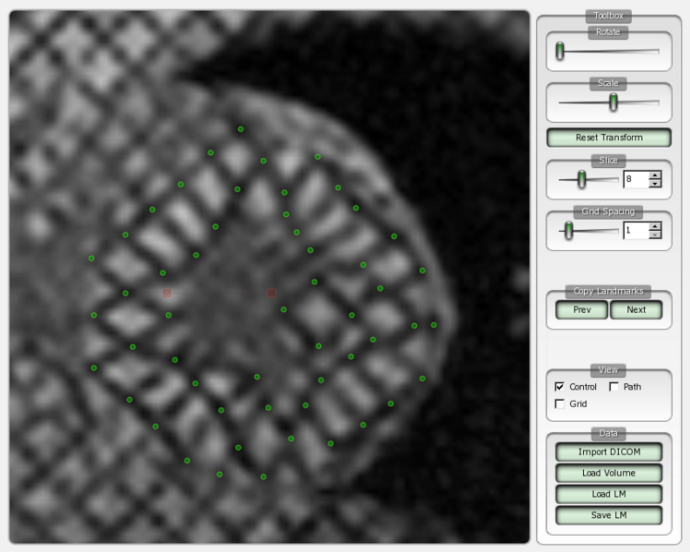


Figure 4.2: The `markTags` program that is used to manually track the tag intersection points.

Pre-processing

The patient specific fiber orientations are required for the motion estimation algorithm. The end-diastolic MR image was segmented manually using the Insight SNAP application. The segmented end-diastolic image was registered to the MR image of an ex-vivo human heart (atlas) for which diffusion tensor imaging was performed. We used the in-house HAMMER algorithm [86] for performing the non-rigid registration with the atlas and template warping was used to estimate the patient specific myocardial fiber orientation as explained in Section 2.3. The segmentations were also used to assign material properties to the myocardial tissue, the blood and the surrounding tissue.

Experiments

For each of the 5 subjects, we had observations from 3 independent observers in the form of manually placed landmarks over the entire image sequences. To account for observer variability, we treated the mean location of the landmark as the ground truth observations. We used a grid size of 64^3 for all inversions. In three separate experiments, we used 70%,

	Error (%) All obs.	Error (%) Control obs.	Error (*mm*) All obs.	Error (*mm*) Control Obs.
Observers	9.21	8.41	1.39	1.26
Algorithm 70%	12.46	14.22	1.88	2.13
Algorithm 50%	16.37	21.02	2.47	3.15
Algorithm 20%	41.61	51.19	6.28	7.67

Table 4.3: Results from tagged data using a 64^3 forward mesh size and an inverse parametrization of 4^3 spatial parameters and 5 temporal B-spline parameters.

50%, and 20% of the ground truth observations as the data for the inversion. The observations that are not used during the inversion are the control observations and are used for assessing the goodness of the inversion.

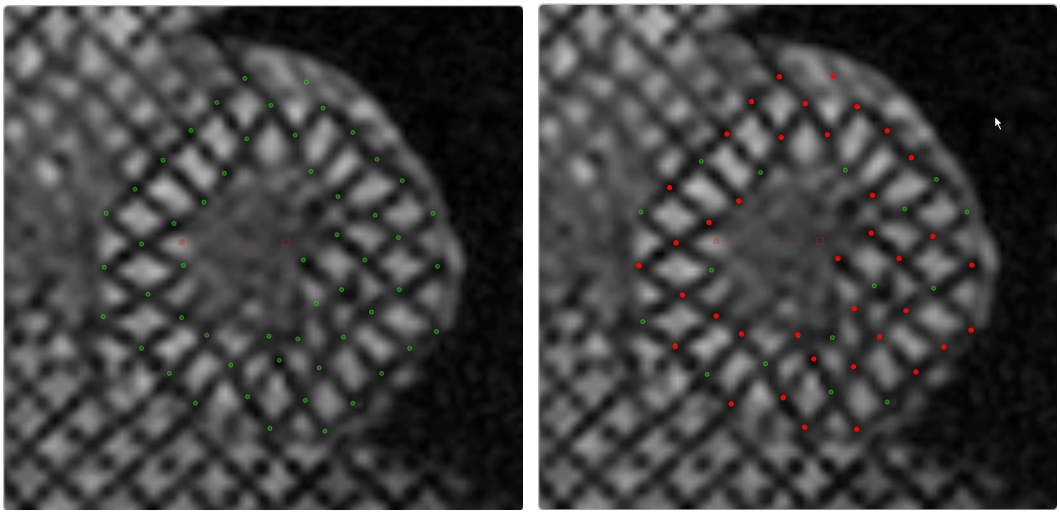
We used the fiber orientation parametrized force model 2.6.2 along with the B-spline and the radial bases to reduce the paramter space. We used a total of 4^3 spatial parameters and for each 5 B-spline temporal parameters, giving us a total of 320 parameters.

After inversion, one additional forward solve was performed using the dense estimates of the fiber contractions to obtain dense estimates of myocardial displacements. These displacements were compared with the ground truth displacements at the tag-intersection points. The relative error (as a percentage) and the absolute error in millimetres for all observations and restricted to only the control observations are shown in Table 4.3. A second experiment, on the same datasets performed using a grid size of 128^3, and using 8^3 spatial parameters and for each 8 B-spline temporal parameters is shown in Table 4.4.

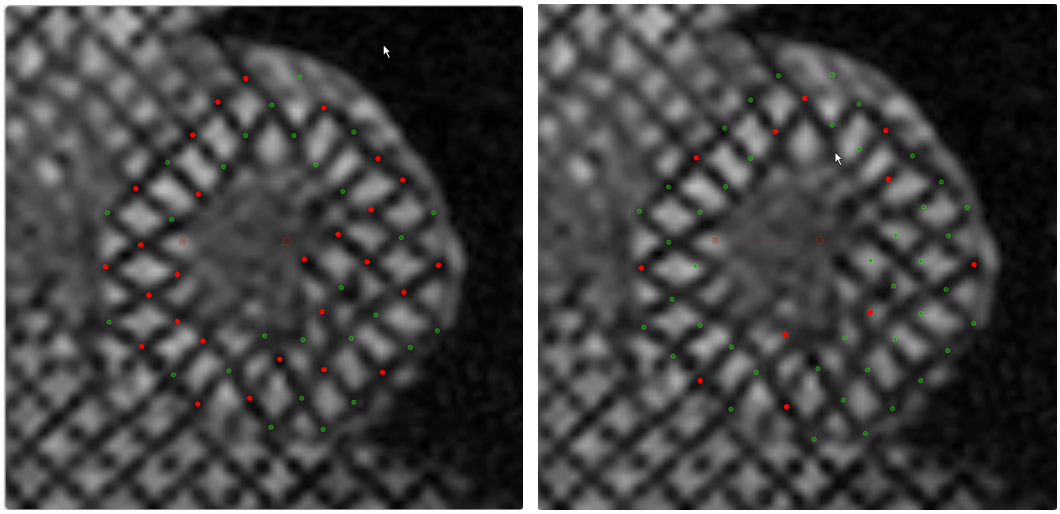
A visual comparison of the estimation with different levels of partial observations overlaid on the original tagged images are shown in Figure 4.3. The partial observations used in each case are marked in red, and the control observations in green. The RMS errors over time for the landmarks on 4 selected (3 short axis and 1 long axis) slices are shown in Figures 4.4-4.6.

The deformation fields at end-diastole, mid-systole and end-systole for the mid ventricular slice from one subject is shown in Figure 4.7.

An alternate way of interpreting the results is to analyze the left-ventricular volume,



(a) Manual (b) Estimate with 70%



(c) Estimate with 50% (d) Estimate with 20%

Figure 4.3: Comparison of the motion estimation algorithm using partial observations. Motion estimation was performed using 70%, 50% and 20% partial observations, which are marked in red. The remaining points (Controls) are shown in green.

	Error (%) All obs.	Error (%) Control obs.	Error (*mm*) All obs.	Error (*mm*) Control Obs.
Observers	9.21	8.41	1.39	1.26
Algorithm 70%	12.08	13.96	1.81	2.09
Algorithm 50%	19.28	22.75	2.89	3.41
Algorithm 20%	44.92	54.86	6.73	8.22

Table 4.4: Results from tagged data using a 128^3 forward mesh size and an inverse parametrization of 8^3 spatial parameters and 8 temporal B-spline parameters.

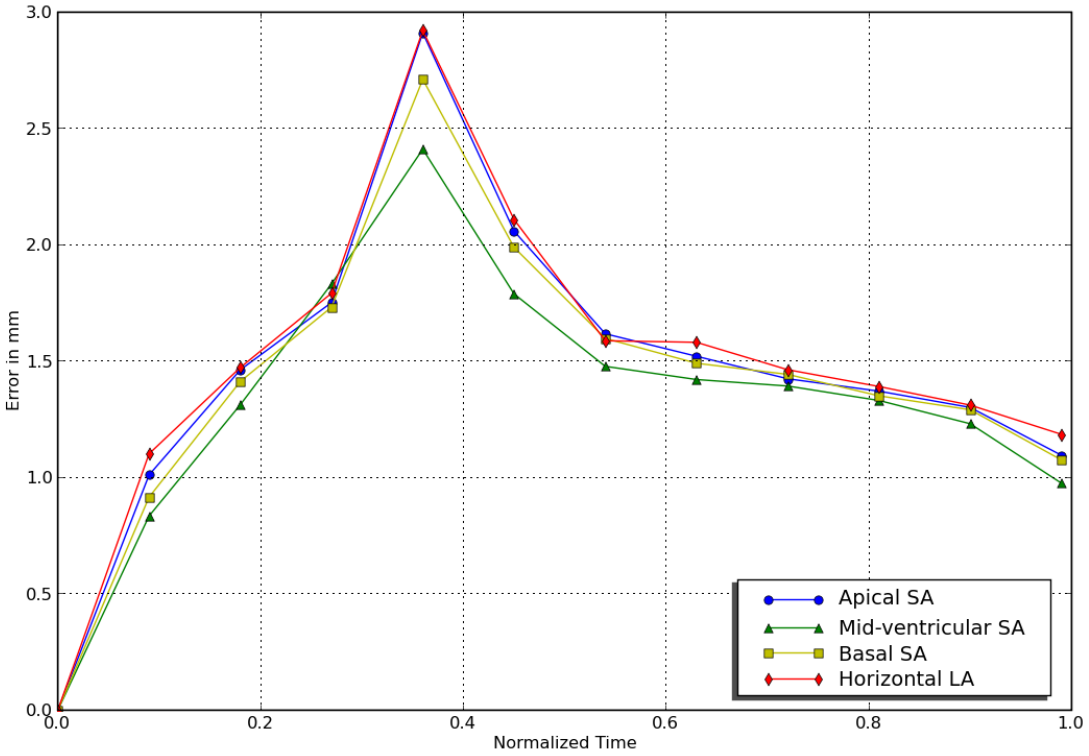


Figure 4.4: Comparison of the RMS errors over time for the motion estimation algorithm using partial observations. Motion estimation was performed using 70% partial observations, and the errors from the basal short axis (SA), mid-ventricular SA, apical SA and the horizontal long axis (LA) slices are plotted as a function of time for each of the partial observation cases.

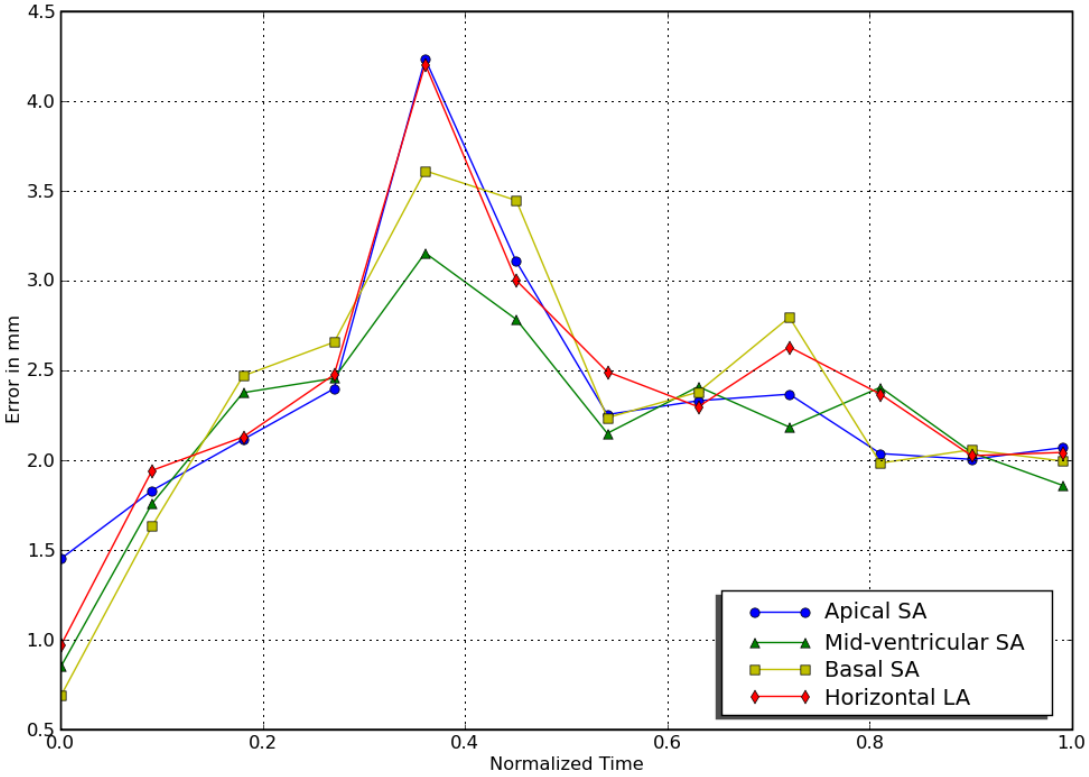


Figure 4.5: Comparison of the RMS errors over time for the motion estimation algorithm using partial observations. Motion estimation was performed using 50% partial observations, and the errors from the basal short axis (SA), mid-ventricular SA, apical SA and the horizontal long axis (LA) slices are plotted as a function of time for each of the partial observation cases.

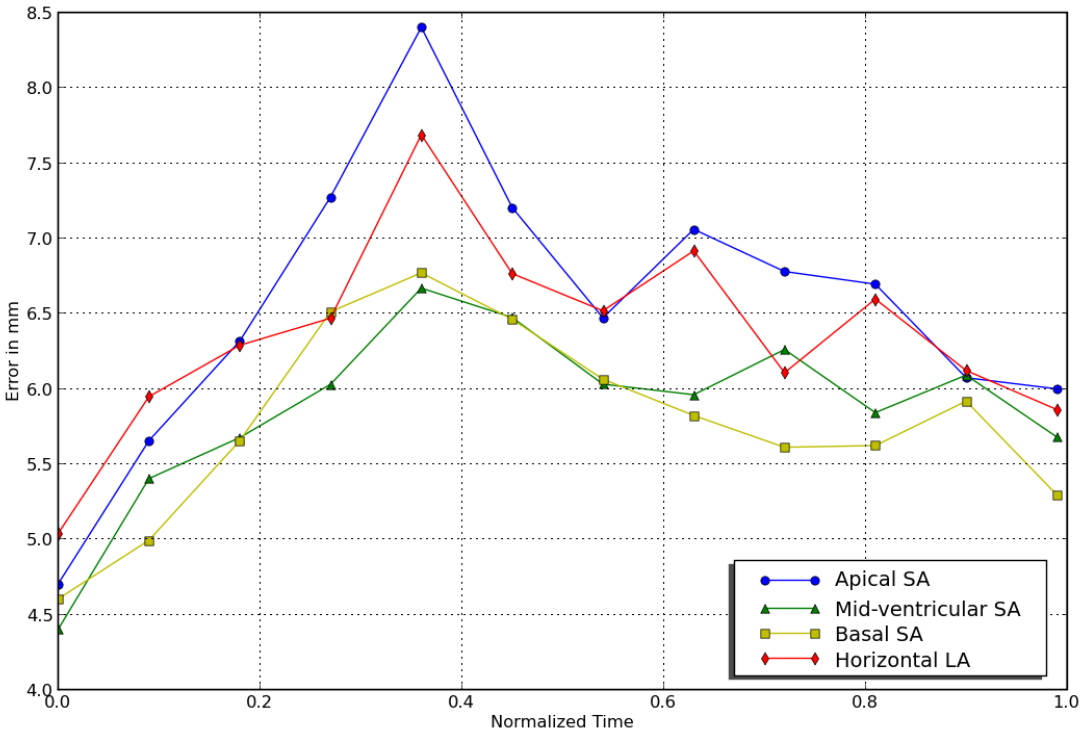


Figure 4.6: Comparison of the RMS errors over time for the motion estimation algorithm using partial observations. Motion estimation was performed using 20% partial observations, and the errors from the basal short axis (SA), mid-ventricular SA, apical SA and the horizontal long axis (LA) slices are plotted as a function of time for each of the partial observation cases.

(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) end-diastole (k) mid-systole (l) end-systole

Figure 4.7: The deformation field shown for selected slices during end-diastole, mid-systole, and end-systole. The top row is an apical slice, the last row is the basal slice, and the middle two are slices from the mid-ventricular region.

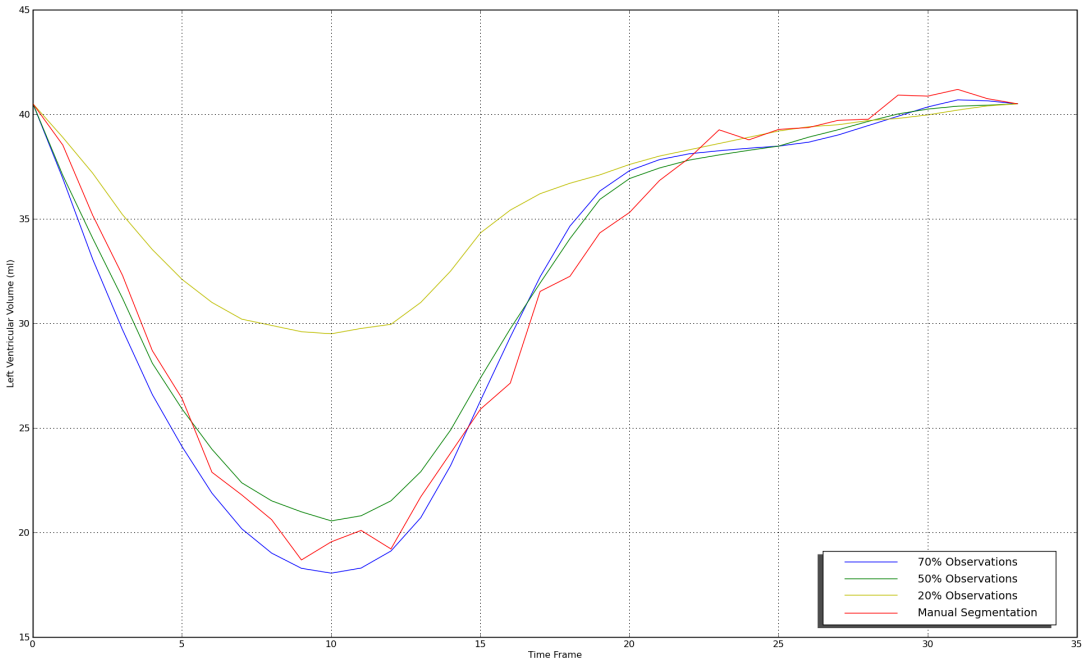


Figure 4.8: Left ventricular volume of a selected subject, segmented automatically and by manually over all frames in a cardiac cycle. The automatic segmentation was performed by propagating the end-diastole segmentation to the other frames using the motion estimates using 70%, 50% and 20% partial observations. These results were obtained using a 4^3 spatial parametrization and 5 B-spline parameters.

obtained using manual segmentation and by the using the motion estimates[2], as seen for one particular subject in Figure 4.8. The motion estimation based volume curve is smoother than the one obtained by manual segmentation, but is in general agreement with the latter. We manually segmented the left ventricle for the 5 subjects and evaluated the difference in ventricular volume over the entire cardiac cycle. The average volume error was 3.37%, with standard deviation 2.56%; and average volume overlap error was 7.04%, with standard deviation 3.28%.

[2]using 70% observations

4.3 Conclusions

We presented a method for cardiac motion reconstruction. We integrate cine-MR images and a biomechanical model that accounts for inhomogeneous tissue properties, fiber information, and active forces. We presented an inversion algorithm. We will able to solve problems that involve 300 million unknowns (forward and adjoint in space-time) in a couple of hours on 64 processors—a modest computing resource. The potential of the method as multicore platforms become mainstream is significant.

The limitations of our current implementation (but not the method) is the assumptions of linear geometric and material response and the potential bias due to template-based fibers that does not account for anatomical variability, that is still requires some preprocessing of the initial frame to assign material properties and fiber orientation, that assumes zero residual stresses and initial conditions, and that it does not include an electrophysiology model.

Our on-going work includes transition to an intensity-based image-registration inversion (in which case we need to solve a nonlinear inversion), further verification of the method by comparing to manually-processed real data, and its clinical validation by reconstructing motions of normal and abnormal populations and conducting statistical analysis. Among the many open problems are the level of required model complexity for clinically relevant motion reconstructions, the bias of the fibers, the sensitivity to the values of the material properties, and the sensitivity to the image similarity functional.

Chapter 5

Conclusions

In this thesis we have presented algorithms for incorporating bio-mechanical models as constraints for cardiac motion estimation. The methods presented are a first attempt at achieving this goal and several extensions to the work presented in this thesis are already being worked on. This chapter aims to summarize the major contributions of this thesis and to provide a glimpse of future work to give an idea of the larger picture that this work is intended to be a part of. The contributions shall be highlighted in context of their applicability and relevance to future work and extentions.

5.1 Fiber Orientations

The proposed algorithm for myocardial fiber estimation works reasonably well, but it can never be expected to fully capture subtle patient specific variations in fiber orientation. It is also unlikely that additional work along similar lines will provide additional improvements. In the long run, it is expected that improvements in cardiac DTI will allow us to acquire *in-vivo* diffusion tensor images, providing us with highly accurate and patient-specific fiber orientations. The algorithms presented in this paper can however still be used to compare diffusion tensor images across patients in that case.

5.2 Octree Meshing

We have presented a set of algorithms for the parallel construction, balancing, and meshing of linear octrees. Our mesh data structure is interfaced with PETSc [6], thus allowing us to use its linear and non-linear solvers. Our target applications include elliptic, parabolic, and hyperbolic partial differential equations. We presented results that verify the overall scalability of our code. The overall meshing time is in the order of one minute for problems with up to four billion elements. Thus, our scheme enables efficient execution of applications which require frequent remeshing.

We need to consider the following factors to improve the performance of the proposed algorithms. In order to minimize communication costs, it is desirable to have as large coarse blocks as possible since the communication cost is proportional to the area of the inter-processor boundaries. However, too coarse blocks will increase the work for the local block balancing stage (Section 3.2.5). If additional local splits are introduced, then the intra-block boundaries increase causing the work load for the first ripple balance to increase. The local balancing step of the algorithm can be made more efficient by performing the local balancing recursively by estimating the correct size of the block that can be balanced by the search-free approach. Such an approach should be based on low-level architecture details, like the cache size.

There are two important extensions: multigrid schemes, and higher-order discretizations. For the former, restriction and prolongation operators need to be designed, along with refinement and coarsening schemes. Higher-order schemes will require additional bookkeeping and longer lookup tables as the inter-element connectivity will increase.

5.3 Image Registration

The first direct extension of the methods described in this thesis is the development of an elastic image registration algorithm using the parallel octree framework. The additional work entailed in this is the incorporation of an image similarity measure within the parallel

octree framework. Preliminary work in this direction was done in the computation of the Octree based Mutual Information, described in Section 3.6. This work was done using a sequential octree mesh and assuming an affine transformation. This has to be extended to arbitrary transformation fields within the parallel octree framework.

5.4 Motion Estimation using Cine MR

The main extension to this work should follow from the image registration case, wherein we can incorporate the image registration measure to replace the objective function (4.1.2) being minimized during motion estimation. This will allow us to use Cine MR images as the data for estimating motion instead of the manually annotated tagged MR images, as proposed in this thesis. Since Cine MR imaging is in far more widespread use in clinical environments as compared to tagged MR images, the benefits of this extension shall be substantial.

Incorporating the image similarity measure will increase the computational complexity of the motion estimation problem and therefore additional work shall be required to improve the convergence of the inverse solvers. Currently work is being done to incorporate octree based multi-grid solvers that should greatly improve the convergence and consequently make the overall method more tangible.

Bibliography

[1] M. F. Adams, H. Bayraktar, T. Keaveny, and P. Papadopoulos. Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In *Proceedings of SC2004*, The SCxy Conference series in high performance networking and computing, Pittsburgh, Pennsylvania, 2004. ACM/IEEE.

[2] V. Akcelik, J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E. J. Kim, J. Lopez, D. R. O'Hallaron, T. Tu, and J. Urbanic. High resolution forward and inverse earthquake modeling on terascale computers. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. ACM, 2003.

[3] D. Alexander, C. Pierpaoli, P. Basser, and J. Gee. Spatial transformations of diffusion tensor magnetic resonance images. *Medical Imaging, IEEE Transactions on*, 20:1131–1139, 2001.

[4] A. H. Association. Heart disease and stroke statistics. *American Heart Association, Dallas, Texas*, 2004.

[5] D. Ayala, P. Brunet, R. Juan, and I. Navazo. Object representation by means of nonminimal division quadtrees and octrees. *ACM Trans. Graph.*, 4(1):41–59, 1985.

[6] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc.

[7] R. Becker and M. Braack. Multigrid techniques for finite elements on locally refined meshes. *Numerical Linear Algebra with applications*, 7:363–379, 2000.

[8] B. Bergen, F. Hulsemann, and U. Rude. Is 1.7×10^{10} Unknowns the Largest Finite Element System that Can Be Solved Today? In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 5, Washington, DC, USA, 2005. IEEE Computer Society.

[9] M. W. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry and Applications*, 9(6):517–532, 1999.

[10] G. Berti. Image-based unstructured 3-d mesh generation for medical application. In *European Congress on Computational Methods in Applied Sciences and Engineering*, Jyväskylä, Finland, 2004.

[11] Bluemke, D. A. and Krupinski, E. A. and Ovitt, T. and Gear, K. and Unger, E. and Axel, L. and Boxt, L. M. and Casolo, G. and Ferrari, V. A. and Funaki, B. and Globits, S. and Higgins, C. B. and Julsrud, P. and Lipton, M. and Mawson, J. and Nygren, A. and Pennell, D. J. and Stillman, A. and White, R. D. and Wichter, T. and Marcus, F. MR imaging of arrhythmogenic right ventricular cardiomyopathy: Morphologic findings and interobserver reliability. *Cardiology*, 99(3):153–162, 2003.

[12] C. A. Bouman and M. Shapiro. A multiscale random field model for Bayesian image segmentation. *IEEE Trans. on Image Processing*, 3(2):162–177, 1994.

[13] P. Brunet and I. Navazo. Solid representation and operation using extended octrees. *ACM Trans. Graph.*, 9(2):170–197, 1990.

[14] P. Cachier. How to trade off between regularization and image similarity in non-rigid registration? In W. Niessen and M. Viergever, editors, *4th Int. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI'01)*, volume 2208 of *LNCS*, pages 1285–1286, Utrecht, The Netherlands, October 2001.

[15] P. Cachier and N. Ayache. Isotropic energies, filters and splines for vectorial regularization. *J. of Math. Imaging and Vision*, 20(3):251–265, May 2004.

[16] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.

[17] Y. Chen, Y.-P. Wang, and A. A. Amini. *Measurement of Cardiac Deformation from MRI: Physical and Mathematical Models*, chapter 8: Tagged MRI Image Analysis from Splines. Kluwer Academic Publishers, 2001.

[18] A. Collignon and *et al.* Automated multimodality medical image registration using information theory. In *Proc. Information Processing in Medical Imaging*, volume 3, pages 263–274, 1995.

[19] D. L. Collins and *et al.* Design and construction of a realistic digital brain phantom. *IEEE Trans. on Medical Imaging*, 17(3):463–468, 1998.

[20] T. Corman, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[21] F. Epstein and W. Gilson. Displacement-Encoded Cardiac MRI Using Cosine and Sine Modulation to Eliminate (CANSEL) Artifact-Generating Echoes. *Magnetic Resonance in Medicine*, 52(4):774–781, 2004.

[22] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[23] L. Freitag and R. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *ACM/IEEE 1999 Conference on Supercomputing*, 13-18 Nov. 1999.

[24] S. Frisken and R. Perry. Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools*, 7(3):1–11, 2002.

[25] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Proc. SIGGRAPH*, pages 124–133, New York, NY, USA, 1980. ACM Press.

[26] E. Gladilin. *Real-time Simulation of Physically Realistic Global Deformations*. PhD thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin, 2003.

[27] S. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[28] G. Golub and P. Basser. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1983.

[29] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *An Introduction to Parallel Computing: Design and Analysis of Algorithms*. Addison Wesley, second edition, 2003.

[30] H. Gray. *Anatomy of the Human Body*. Lea & Febiger, Philadelphia, 1918.

[31] D. M. Greaves and A. G. L. Borthwick. Hierarchical tree-based finite element mesh generation. *International Journal for Numerical Methods in Engineering*, 45(4):447–471, 1999.

[32] L. Greengard and W. Gropp. A parallel version of the fast multipole method-invited talk. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 213–222, Philadelphia, PA, USA, 1989. Society for Industrial and Applied Mathematics.

[33] M. Griebel and G. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and U. Trottenberg, editors, *Parallel Computing: Fundamentals, Applications and New Directions, Proceedings of the Conference ParCo'97, 19-22 September 1997, Bonn, Germany*, volume 12, pages 589–600, Amsterdam, 1998. Elsevier, North-Holland.

[34] M. E. Gurtin. *An Introduction to Continuum Mechanics*. Academic Press, 1981.

[35] B. Hariharan, S. Aluru, and B. Shanker. A scalable parallel fast multipole method for analysis of scattering from perfect electrically conducting surfaces. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–17, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[36] B. V. Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 103–110, New York, NY, USA, 1987. ACM Press.

[37] E. Hsu and V. Henriquez. Myocardial fiber orientation mapping using reduced encoding diffusion tensor imaging. *Journal of Cardiovascular Magnetic Resonance*, 3:339–347, 2001.

[38] Z. Hu, D. Metaxas, and L. Axel. In vivo strain and stress estimation of the heart left and right ventricles from MRI images. *Medical Image Analysis*, 7(4):435–444, 2003.

[39] T. Hughes. *The Finte Element Method. Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[40] W. W. Irving and *et al.* An overlapping tree approach to multiscale stochastic modeling and estimation. *IEEE Trans. on Image Processing*, 6(11):1517–1529, 1997.

[41] G. J., B. D.A., O. N.F., and et al. Fast determination of regional myocardial strain fields from tagged cardiac images using harmonic phase mri. *Circulation*, 101:981–988, 2000.

[42] A. Jones and P. Jimack. An adaptive multigrid tool for elliptic and parabolic systems. *International Journal for Numerical Methods in Fluids*, 47:1123–1128, 2005.

[43] E. Kim, J. Bielak, O. Ghattas, and J. Wang. Octree-based finite element method for large-scale earthquake ground motion modeling in heterogeneous basins. *AGU Fall Meeting Abstracts*, Dec. 2002.

[44] J. Laferté, P. Perez, and F. Heitz. Discrete Markov image modeling and inference on the quadtree. *IEEE Trans. on Image Processing*, 9(3):390–404, 2000.

[45] M. Ledesma-Carbayo, J. Kybic, M. Desco, A. Santos, and M. Unser. Cardiac motion analysis from ultrasound sequences using non-rigid registration. In W. Niessen and M. Viergever, editors, *MICCAI*, volume 2208 of *Lecture Notes in Computer Science*, pages 889–896, Utrecht, The Netherlands, October 14-17, 2001. Springer.

[46] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19(3):261–296, 1987.

[47] S. Z. Li. *Markov random field modeling in computer vision*. Springer-Verlag, 1995.

[48] J. Liu, B. Vemuri, and J. Marroquin. Local frequency representations for robust multimodal image registration. *Medical Imaging, IEEE Transactions on*, 21(5):462–469, May 2002.

[49] H. Luan, F. Qi, and D. Shen. Multi-modal image registration by quantitative-qualitative measure of mutual information Q-MI. In *Proc. CVBIA*, pages 378–387, 2005.

[50] M. R. Luettgen, W. Clem Karl, and A. S. Willsky. Efficient multiscale regularization with applications to the computation of optical flow. *IEEE Trans. on Image Processing*, 3(1):41–64, 1994.

[51] G. M.A., P. J.L., and M. E.R. Tag and contour detection in tagged mr images of the left-ventricle. *IEEE Transactions on Medical Imaging*, 13:74–88, 1994.

[52] H. MAGISTRALE, S. DAY, R. CLAYTON, and R. GRAVES. The SCEC Southern California reference three-dimensional seismic velocity model version 2. *Bulletin of the Seismological Soceity of America*, 2000.

[53] T. Makela, P. Clarysse, O. Sipila, N. Pauna, Q. C. Pham, T. Katila, and I. Magnin. A review of cardiac image registration methods. *Medical Imaging, IEEE Transactions on*, 21(9):1011–1021, Sep 2002.

[54] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(8):837–842, 1996.

[55] D. J. Mavriplis, M. J. Aftosmis, and M. Berger. High resolution aerospace applications using the nasa columbia supercomputer. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 61, Washington, DC, USA, 2005. IEEE Computer Society.

[56] A. McCulloch, J. Bassingthwaighte, P. Hunter, and D. Noble. Computational biology of the heart: from structure to function. *Prog Biophys Mol Biol*, 69(2-3):153–5, 1998.

[57] J. McEachen, II, A. Nehorai, and J. Duncan. Multiframe temporal estimation of cardiac nonrigid motion. *Image Processing, IEEE Transactions on*, 9(4):651–665, Apr. 2000.

[58] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19:129–147, 1982.

[59] P. Moireau, D. Chapelle, and P. Le Tallec. Joint state and parameter estimation for distributed mechanical systems. *Computer Methods in Applied Mechanics and Engineering*, 197:659–677, 2008.

[60] D. Moore. The cost of balancing generalized quadtrees. In *Symposium on Solid Modeling and Applications*, pages 305–312, 1995.

[61] S. Narasimhan, R.-P. Mundani, and H.-J. Bungartz. An octree and a graph-based approach to support location aware navigation services. In *Proceedings of the 2006 International Conference on Pervasive Systems & Computing, (PSC 2006)*, pages 24–30, Las Vegas, Nevada, USA, 2006. CSREA Press.

[62] N. Osman, W. Kerwin, E. McVeigh, and J. Prince. Cardiac motion tracking using CINE harmonic phase(HARP) magnetic resonance imaging. *Magnetic Resonance in Medicine*, 42(6):1048–1060, 1999.

[63] X. Papademetris, A. Sinusas, D. Dione, and J. Duncan. Estimation of 3D left ventricular deformation from echocardiography. *Medical Image Analysis*, 5(1):17–28, 2001.

[64] X. Papademetris, A. J. Sinusas, D. P. Dione, R. T. Constable, and J. S. Duncan. Estimation of 3-D left ventricular deformation from medical images using biomechanical models. *IEEE Transactions on Medical Imaging*, 21(7):786, 2002.

[65] D. Perperidis, R. Mohiaddin, and D. Rueckert. Spatio-temporal free-form registration of cardiac mr image sequences. In *MICCAI*, pages 911–919, 2004.

[66] C. Pierpaoli, P. Jezzard, P. Basser, A. Barnett, and G. Chiro. Diffusion tensor mr imaging of the human brain. *Radiology*, 201(3):637–648, 1996.

[67] J. Pluim, J. Maintz, and M. Viergever. Mutual-information-based registration of medical images: a survey. *IEEE Trans. on Medical Imaging*, 22(8):986–1004, 2003.

[68] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever. Image registration by maximization of combined mutual information and gradient information. *IEEE Trans. on Medical Imaging*, 19(8):809–814, 2000.

[69] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, 190:572–600(29), 20 September 2003.

[70] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[71] P. Prez. Markov random fields and images. *CWI Quarterly*, 11(4):413–437, 1998.

[72] R. F. Rice. Some practical universal noiseless coding techniques. Technical Report JPL Publication 79-22, Jet Propulsion Laboratory, Pasadena, California, 1979.

[73] A. Roche, X. Pennec, G. Malandain, and N. Ayache. Rigid registration of 3D ultrasound with mr images: a new approach combining intensity and gradient information. *Medical Imaging, IEEE Transactions on*, 20(10):1038–1049, Oct. 2001.

[74] M. R. Sabuncu and P. J. Ramadge. Gradient based nonuniform sampling for information theoretic alignment methods. In *Proc. Int. Conf. of the IEEE EMBS*, San Francisco, CA, 2004.

[75] F. Sachse, C. Henriquez, G. Seemann, C. Riedel, C. Werner, R. Penland, D. Davis, and E. Hsu. Modeling of fiber orientation in the ventricular myocardium with mr diffusion imaging. In *Proc. Computers in Cardiology*, volume 28, pages 617–620, 2001.

[76] F. B. Sachse. *Computational Cardiology: Modeling of Anatomy, Electrophysiology, and Mechanics*, volume 2966 of *Lecture Notes in Computer Science*. Springer, 2004.

[77] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.

[78] R. Schneiders. An algorithm for the generation of hexahedral element meshes based on an octree technique, 1997.

[79] R. Schneiders, R. Schindler, and F. Weiler. Octree-based generation of hexahedral element meshes, 1996.

[80] D. Scollan, A. Holmes, R. Winslow, and J. Forder. Histological validation of myocardial microstructure obtained from diffusion tensor magnetic resonance imaging. *Am. J. Physiol. (Heart Circulatory Physiol.*, 275:2308–2318, 1998.

[81] M. Sermesant, Y. Coudiére, H. Delingette, N. Ayache, J.Sainte-Marie, F. C. D. Chapelle, and M. Sorine. Progress towards model-based estimation of the cardiac electromechanical activity from ECG signals and 4D images. In *Modeling and Simulation for Computer-aided Medicine and Surgery*, 2002.

[82] M. Sermesant, H. Delingette, and N. Ayache. An electromechanical model of the heart for image analysis and simulation. *IEEE Transactions in Medical Imaging*, 25(5):612–625, 2006.

[83] M. Sermesant, P. Moireau, O. Camara, J. Sainte-Marie, R. Andriantsimiavona, R. Cimrman, D. Hill, D. Chapelle, and R. Razavi. Cardiac function estimation from MRI using a heart model and data assimilation: Advances and difficulties. *Medical Image Analysis*, 10(4):642–656, 2006.

[84] M. Sermesant, K. Rhode, G. Sanchez-Ortiz, O. Camara, R. Andriantsimiavona, S. Hegde, D. Rueckert, P. Lambiase, C. Bucknall, E. Rosenthal, H. Delingette, D. Hill, N. Ayache, and R. Razavi. Simulation of cardiac pathologies using an electromechanical biventricular model and xmr interventional imaging. In *MICCAI*, pages 467–480, 2004.

[85] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27, 1948.

[86] D. Shen and C. Davatzikos. Hammer: hierarchical attribute matching mechanism for elastic registration. *Medical Imaging, IEEE Transactions on*, 21(11):1421–1439, Nov. 2002.

[87] M. Shephard and M. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 26:709–749, 1991.

[88] J. R. Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, pages 86–95, Minneapolis, Minnesota, USA, June 1998. Association for Computing Machinery.

[89] P. Shi, A. Sinusas, R. T. Constable, and J. Duncan. Volumetric deformation analysis using mechanics–based data fusion: Applications in cardiac motion recovery. *International Journal of Computer Vision*, 35(1):65–85, Nov. 1999.

[90] S. Smart, T. Knickelbine, F. Malik, and K. Sagar. Dobutamine-Atropine Stress Echocardiography for the Detection of Coronary Artery Disease in Patients With Left Ventricular Hypertrophy Importance of Chamber Size and Systolic Wall Stress, 2000.

[91] A. K. Somani and A. M. Sansano. Minimizing overhead in parallel algorithms through overlapping communication/computation. Technical report, Institute for Computer Applications in Science and Engineering (ICASE), 1997.

[92] S. Song and R. Leahy. Computation of 3d velocity fields from 3d cine ct images of a human heart. *Medical Imaging, IEEE Transactions on*, 10(9):295–306, Sep 1991.

[93] K. Strasters and J. Gerbrands. 3-dimensional image segmentation using a split, merge and group-approach. *Pattern Recognition Letters*, 12(5):307–325, May 1991.

[94] D. D. Streeter Jr. and W. T. Hanna. Engineering Mechanics for Successive States in Canine Left Ventricular Myocardium: II. Fiber Angle and Sarcomere Length. *Circ Res*, 33(6):656–664, 1973.

[95] C. Studholme, D. Hill, and D. Hawkes. An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition*, 32(1):71–86, 1999.

[96] H. Sundar, R. S. Sampath, S. S. Adavani, C. Davatzikos, and G. Biros. Low-constant parallel algorithms for finite element simulations using linear octrees. In *ACM/IEEE SCXY Conference Series*, 2007.

[97] H. Sundar, R. S. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008.

[98] H. Sundar, D. Shen, G. Biros, H. Litt, and C. Davatzikos. Estimating myocardial fiber orientation by template warping. In *IEEE International Symposium on Biomedical Imaging, Arlington, Virginia, USA*, Apr 2006.

[99] H. Sundar, D. Shen, G. Biros, C. Xu, and C. Davatzikos. Robust computation of mutual information using spatially adaptive meshes. In *Medical Image Computing and Computer-Assisted Intervention MICCAI 2007*, pages 950–958, 2007.

[100] C. Teh and R. Chin. On image analysis by the methods of moments. *IEEE Trans. PAMI*, 10:496–513, 1988.

[101] S.-H. Teng. Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. *SIAM Journal on Scientific Computing*, 19, 1998.

[102] S. Timoner. *Compact Representations for Fast Nonrigid Registration of Medical Images*. PhD thesis, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, 2003.

[103] F. M. Todd Constable, et al. Tracking myocardial deformation using phase contrast mr velocity fields: a stocastic approach. *IEEE Transactions on Medical Imaging*, 15(4):453–465, 1996.

[104] H. Tropf and H. Herzog. Multidimensional range search in dynamically balanced trees. *Angewandte Informatik*, 2:71–77, 1981.

[105] W.-Y. Tseng, T. Reese, R. Weisskoff, and V. Wedeen. Cardiac diffusion tensor mri in vivo without strain correction. *Magnetic Resonance in Medicine*, 42(2):393–403, 1999.

[106] T. Tu and D. R. O'Hallaron. Balance refinement of massive linear octree datasets. *CMU Technical Report*, CMU-CS-04(129), 2004.

[107] T. Tu and D. R. O'Hallaron. Extracting hexahedral mesh structures from balanced linear octrees. In *13th International Meshing Roundtable*, pages 191–200, Williamsburg, VA, Sandia National Laboratories, September 19-22 2004.

[108] T. Tu, D. R. O'Hallaron, and O. Ghattas. Scalable parallel octree meshing for terascale applications. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 4, Washington, DC, USA, 2005. IEEE Computer Society.

[109] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'Hallaron. From mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 91, New York, NY, USA, 2006. ACM Press.

[110] P. Viola. *Alignment by Maximization of Mutual Information*. PhD thesis, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, 1995.

[111] J. C. Walker, J. M. Guccione, Y. Jiang, P. Zhang, A. W. Wallace, E. W. Hsu, and M. B. Ratcliffe. Helical myofiber orientation after myocardial infarction and left ventricular surgical restoration in sheep. *J Thorac Cardiovasc Surg*, 129(2):382–390, 2005.

[112] W. Wang. Special bilinear quadrilateral elements for locally refined finite element grids. *SIAM Journal on Scientific Computing*, 22:2029–2050, 2001.

[113] Y.-P. Wang, Y. Chen, and A. A. Amini. Efficient lv motion estimation using subspace approximation techniques. *Medical Image Analysis*, 20(6):499–513, Jun. 2001.

[114] M. S. Warren and J. K. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 570–576, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[115] M. S. Warren and J. K. Salmon. A parallel hashed octree N-body algorithm. In *Proceedings of Supercomputing '93*, 31 March 1993.

[116] W. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis. Multi-modal volume registration by maximization of mutual information. *Medical Image Analysis*, 1(1):35–51, 1996.

[117] D. Xu, S. Mori, D. Shen, P. C. v. Zijl, and C. Davatzikos. Spatial normalization of diffusion tensor fields. *Magnetic Resonance in Medicine*, 50:175–182, 2003.

[118] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J. Comput. Phys.*, 196(2):591–626, 2004.

[119] L. Ying, G. Biros, D. Zorin, and H. Langston. A new parallel kernel-independent fast multipole method. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 14, Washington, DC, USA, 2003. IEEE Computer Society.

[120] E. Zerhouni, D. Parish, W. Rogers, A. Yang, and E. Shapiro. Human heart: tagging with mr imaging–a method for noninvasive assessment of myocardial motion. *Radiology*, 169(1):59–63, 1988.

[121] B. Zitová and J. Flusser. Image registration methods: a survey. *Image Vision Comput.*, 21(11):977–1000, 2003.