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

2010

Christos Davatzikos, Supervisor of Dissertation	Susan Margulies, Graduate Group Chair
George Biros, Committee Member	Victor Ferrari, Committee Member
Harold Litt, Committee Member	Dinggang Shen, Committee Member

Quidquid id est,

timeo Danaos et dona ferentes.

-Vergil, Aeneid II.49

Acknowledgements

ABSTRACT

SPATIO-TEMPORAL DEFORMATION ANALYSIS OF CARDIAC MR IMAGES

Hari Sundar

Supervisor: Christos Davatzikos

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. Consequently, it is important for any cardiac diagnosis method to consider both these aspects. 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 [12]. Studies also suggest that standardizing acquisition protocols and objective analysis especially of regional myocardial function will help improve the accuracy and reduce inter-observer variability [53]. 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 structural and functional changes in the heart. The development of such methods is the primary goal of this project.

Aim: Develop methods to measure myocardial function from MR Cine images, specifically myocardial wall motion. We expect that the use of a model of cardiac motion to constrain the motion estimation problem shall improve the accuracy of motion estimation.

Our hypotheses are that 1) these methods will improve the accuracy and robustness of estimating myocardial motion from MR Cine sequences (Aim 1), 2) these methods will detect subtle and spatio-temporally distributed structural and functional differences between patients and healthy individuals, thereby enabling the construction of sensitive and specific

diagnostic tools for the early detection of ARVC and other diffuse cardiomyopathies that currently remain undetectable, especially at early stages (Aims 2 and 3).

Although the algorithms and methods developed as part of this work should apply in general to the whole class of diffuse cardiomyopathies, we restrict the scope to the characterization of Arrhythmogenic right ventricular cardiomyopathy (ARVC). Future work shall focus on how the methods developed as part of this work can be generalized to other diffuse cardiomyopathies.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

According to WHO estimates, 16.7 million people around the world die of cardiovascular diseases (CVD) each year [4]. Of the total CVD deaths annually, about 8.6 million are of women. Heart attack and stroke deaths are responsible for twice as many deaths in women as all cancers combined. The US health care system is facing serious access and quality issues. The current access and quality issues will be compounded in the coming years by three factors that will serve to accelerate the rate of cardiovascular disease and its complications.

First, aging of the population will undoubtedly result in a concomitant increase in the incidence of chronic diseases, including coronary artery disease, heart failure and stroke. Second, we are experiencing an explosive increase in the prevalence of obesity and type 2 diabetes and their related complications of hypertension, hyperlipidemia, and artherosclerotic vascular disease [4]. Finally, there is an alarming increase in unattended risk factors in the younger generations that will continue to fuel the cardiovascular epidemic for years to come. These factors include obesity and smoking. CVD is the leading cause of mortality

in every region in the world except sub-Saharan Africa, and it is anticipated that cardiovascular disease will eclipse the present the current leader in that region, infectious disease, within the next few years. By 2020 the WHO estimates nearly 25 million CVD deaths worldwide. By 2020, cardiovascular diseases, injury and mental illnesses will be responsible for about one half of all deaths and one half of all healthy years lost, worldwide. The socio-economic impact of cardiovascular disease is too great to be ignored. In 2004 the estimated direct and indirect cost of CVD was $368.4 billion [4].

It is noteworthy that the principal cardiovascular disorder responsible for the global rise in mortality is no longer rheumatic heart disease, but rather artherosclerotic vascular disease. Ischemic heart disease is the leading cause of death in the world, and cerebrovascular disease is the second leading cause [4]. It is often assumed that artherosclerosis is a disease of the affluent, industrialized countries. However, 80% of these deaths occur in low-to-middle income countries of varying size like China, Russia, Poland, Mauritius, Argentina, and India [90]. In many countries, the need for care already outstrips the ability to provide it to its citizens. Throughout the world, even in economically advanced societies, there are deficiencies in preventive and acute care that might stem the tide of this epidemic. Because of these reasons, it is important to stop think of cardiovascular diseases as a rich-man's disease and start dealing with it as an epidemic. Consequently, it is extremely important to develop cheap, non-invasive early detection methods to be able to identify the onset of cardiovascular diseases and treat them before they cause excessive damage.

There are a number of invasive and non-invasive procedures that are currently used for diagnosis of CVD [9]. To reduce trauma for the patient it is preferable to use non-invasive procedures. Important noninvasive techniques are plain radiography, radionucleotide imaging, positron emission tomography (PET), Magnetic Resonance Imaging (MRI) and Ultrasound [9]. Of these, MRI can provide much cardiac information during a single examination and may thus be more cost-effective than several other studies. In the last few years

MRIs have become cheaper and more accessible and given the human and economic cost of CVD, it is important that high-risk populations be screened regularly for CVD.

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. 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 [12]. Studies also suggest that standardizing acquisition protocols and objective analysis especially of regional myocardial function will help improve the accuracy and reduce inter-observer variability [53]. 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 structural and functional changes in the heart. The main contribution of this thesis is the development of computational methods to characterize myocardial function from MR images.

Although the algorithms and methods developed as part of this work should apply in general to the whole class of diffuse cardiomyopathies, we restrict the scope to the characterization of Arrhythmogenic right ventricular cardiomyopathy (ARVC). Future work shall focus on how the methods developed as part of this work can be generalized to other diffuse cardiomyopathies.

1.2 Arrhythmogenic Right Ventricular Cardiomyopathy

1.2.1 Clinical Features and Relevance

Arrhythmogenic right ventricular cardiomyopathy is generally accepted as the most common cause of sudden cardiac death in young patients, and despite over 25 years of study remains a poorly understood disease [22, 79]. Recent genetic studies have elucidated both autosomal dominant and recessive inheritance mechanisms [54]. Arrhythmogenic right ventricular cardiomyopathy (ARVC), also known as arrhythmogenic right ventricular dysplasia is characterized by progressive fibrofatty replacement of right ventricular myocardium, initially with typical regional and later global right and some left ventricular involvement, with relative sparing of the septum [79]. As the underlying pathophysiology of ARVC remains unknown, there is no consensus regarding a gold standard for diagnosis [78]. Diagnosis of ARVC is based on presence of major and minor criteria that include structural, histological, electrocardiographic, arrhythmic, and genetic factors. At its early stages, the diagnosis of ARVC remains a clinical challenge. The different imaging modalities play a limited role, as there is no single non- invasive method that helps to establish or exclude this diagnosis.

1.2.2 MR Imaging of ARVC

While Magnetic Resonance imaging (MRI) findings were not included in the original Task Force criteria because of a lack of evidence of efficacy, it has been evaluated as a method for demonstration of the structural and functional abnormalities listed, including myocardial fatty replacement, RV dilation, wall thinning, and aneurysm formation, and evaluation of RV function [12]. However, the diagnostic performance for detection of these abnormalities, as measured by sensitivity, specificity, and inter-observer variability remains some-

what uncertain. A recent single center study compared MR findings in 12 patients with definitive diagnosis of ARVC by Task Force criteria, with 10 age and sex matched controls [75]. The study found evidence of intramyocardial fat in 75% of the patients and none of the controls, a greater incidence of RV hypertrophy and statistically significant increases in quantitative measures of RV dimensions and decreases in RV function. In another study [12], 13 readers at multiple institutions reviewed images from MR evaluations of 7 patients with a diagnosis of ARVC by Task Force criteria, 6 controls, and 32 patients with suspected ARVC. While the presence of reported RV enlargement and other morphological abnormalities were significantly higher in the definitive ARVC patients, the percentages with reported intramyocardial fat were equal amongst the three groups. Overall diagnostic quality was poor and there was wide inter-observer variability for all parameters evaluated. Limitations of this study included non-standardization of MR acquisition techniques and criteria for interpretation, as well as lack of inclusion of functional cine images in the interpretations. The findings suggest that standardizing acquisition protocols and standardized, objective analysis especially of regional myocardial function will improve the accuracy and reduce variability.

1.3 Assessing Myocardial Function

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 [58], moments [66, 76], Gabor features [46], and local frequency representations [41]. 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 [70]. 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 [95]. 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 [69, 18]. 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 [42], the displacement field can be estimated and intramyocardial strain can be computed in a variety of ways [95]. 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 [51], 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 [38]. 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 [80], 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) [20] and harmonic phase imaging (HARP) [51] 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. However, data-acquisition in single-breath-hold DENSE MR imaging has been limited to only one cardiac phase. Multiple breath-hold 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 6-10mm. The temporal resolution varies between 25-70ms. A lot of work has been done in extracting cardiac motion fields from MR and Ultrasound image sequences [69, 40, 48, 52, 55, 72, 87]. 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 [69, 52, 87]. 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 [69, 48]. 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 [40, 55, 72]. Perperidis et al. [55] 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 [15, 16]. 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 [47]. 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 [21] and a general review of image registration methods can be found in [96].

1.4 Biomechanical Modeling of the Heart

1.5 Contributions

1.6 Organization of this Thesis

Chapter 2

Image Registration based Motion Estimation

For any image based diagnosis algorithm it is important that we are able to robustly extract features that characterize the pathology and use these to be able to classify the subjects. The difference between feature extraction and classification is somewhat arbitrary: An ideal feature extractor would yield a representation that makes the job of the classifier trivial; conversely, an ideal classifier would not need the help of a sophisticated feature extractor. However, feature extractors play another important role which is particularly important in our case, that of dimension reduction.

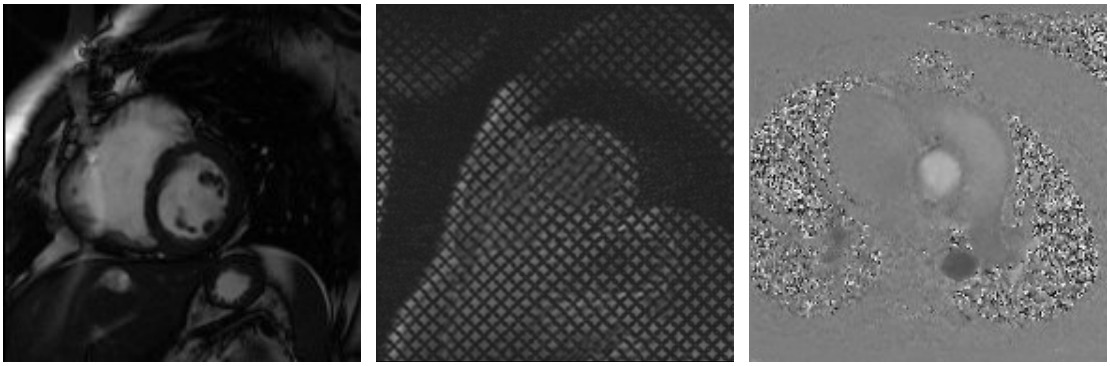
Most of the work done under CAD has focused on pathologies that can be characterized by structural changes, like tumors. The only case where diagnosis of pathologies has utilized functional information invloves functional imaging modalities like fMRI or PET. Structural information, albeit important, is not sufficient to characterize cardiomyopathies. Cardiomyopathies present themselves in different forms, both by structural changes like plaque formation, fat deposits, etc., and also 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. Work has been done on extracting image features like intensities and gradients [58], moments [66][76], Gabor features [46][44] and local frequency representations [41]. 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 ischaemic heart diseases (ischaemia, 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 [60]. 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. Identifying changes in the myocardium based on both structural and functional changes makes the system more robust. Of course quantization of the myocardial wall motion represents a challenge in itself.

2.1 Using MR Cine Sequences for Structural and Functional Characterization

Since the ultimate goal is to be able to classify patient scans in order to diagnose cardiomyopathies, the following factors need to be considered for the choice of imaging modality.

- CAD applications are supposed to aid physicians during diagnosis and not really work independently. Therefore the modality being used by the CAD program should be convenient for the physician to use for diagnosis. In a typical setting the CAD program will filter our most of the patients scanned. The physician reevaluates these patients and removes the false positives from the filtered list. If the physician is not able to diagnose using the same images as that used by the CAD program then an



(a) MR Cine (b) MR Tagged (c) MR Phase Contrast

Figure 2.1: Different types of MR images

additional set of images will need to be acquired for the physician to confirm the diagnosis

- The modality should be capable of providing both structural and functional information about the myocardium. This means that the modality should be able to provide 4D datasets.

- It is also preferable that we have a large enough database of available scans for normal and pathological cases so that we can train our classifier. It would be very difficult and expensive to acquire new datasets for different kinds of patients. This suggests the use of MR Cine sequences as it is very routine to acquire these for patients with most cardiomyopathies.

- Since patients need to be scanned on a regular basis, the scan modality should be simple and patient discomfort and trauma should be minimal. This rules out modalities requiring intervention. This also implies minimizing the time spent inside the scanner by the patient and also the number of breathholds required for the scan.

Most clinical modalities used to image myocardial function evaluate passive wall motion (ventriculography) or wall thickening (echocardiography, gated single-photon emis-

sion 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. The two most widely used MR imaging methods for the quantification of intramyocardial wall motion are myocardial tagging [95] and phase contrast imaging [80]. Each technique has advantages and disadvantages, which shall now be described.

MR Tagging was developed to provide non-invasive virtual markers inside the myocardium, which deform with myocardial motion [95]. 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 with accuracy [69][1][18]. 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 [42], the displacement field can be estimated and intramyocardial strain can be computed in a variety of ways [95]. 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 [1], 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 breathold acquisitions, the resolution has been relatively poor [38].

The *second* approach is that of MR phase contrast imaging [80], 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.

Both of these methods do not satisfy the requirements expected from the suitable modality. Therefore we look at the alternate approach of extracting functional characteristics, i.e., myocardial wall motion from MR Cine sequences, which satisfies most of the requirements. A lot of work has been done in extracting cardiac motion fields from MR Cine sequences [69][52][87][48][72][40][55]. There are again two approaches to estimating the motion fields. The first method 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 [69][52][87]. 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 [69][48]. These are very sensitive to the accuracy with which the myocardium can be segmented. Also they do a poor job in regions within the myocardium, and mainly try to align the myocardial boundaries. The other approach uses energy-based warping or optical flow techniques to compute the displacement of the myocardium [40][72][55]. Perperidis et al. [55] 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. There are two major shortcomings of this method. Firstly, the similarity function is evaluated over the entire image and this fails to capture subtle localized deformations. Since our ultimate goal is to be able to characterize cardiomyopathies these subtle variations are very important. Secondly, MR Cine sequences have very large inter slice spacing and slice thicknesses. This makes the resolution along

this axis extremely poor and the heart has substantial deformation in that direction. By comparison, the temporal resolution is very good. Therefore it is important to utilize the temporal information while detecting the correspondence or the similarity between two images. The B-spline based method evaluates the similarity over 3-D volumes, and uses only the spatial information for evaluating the similarity. The temporal factors only come in because of the smoothing term that happens because of the use of the 4-D B-spline grid. This is a major problem with this approach. Using temporal information to aid correspondence detection can improve motion field estimation. We use the mechanical model for the heart that was described in Section 3 to constrain the motion estimation process. This allows us to better capture the subtle twisting action of the myocardium that the other methods are unable to do. A more detailed and thorough review of cardiac image registration methods can be found in [21] and a more general review of image registration methods can be found in [96].

We shall now describe our motion field estimation algorithm. This work builds up on work done earlier for deformable registration [66] and correspondence detection [92] in the brain. We first define the wavelet attribute vector that we use in order to detect correspondence. Then we define our registration framework that solves for the motion fields over the entire sequence.

2.2 Wavelet Attribute Vectors

The accuracy and robustness of correspondence detection depends on how effectively an attribute vector can capture the local and global properties of the a given point. In a registration context it is desirable that these attributes are scale and rotation invariant. We use wavelet-based attribute vectors to determine correspondence in the MR images. This work builds up on [92], where the authors justify the use of wavelet based attribute vec-

tors for determining correspondence in 3D MR brain images. The heart does not have many geometrically discernible points and most local operators like intensity, gradient information, fail to capture the uniqueness of a given voxel effectively. The wavelet based attribute vector is extracted from the wavelet subimages and reflects the image structure in a large neighborhood around the voxel in a multi-scale fashion. The use of the wavelet based attribute vectors allow us to uniquely identify points in the cardiac sequence and also identify *focus* points within the sequence that are better suited for registration at different resolutions.

2.2.1 Construction of Wavelet Attribute Vectors

We construct the wavelet attribute vectors in a multi-resolution manner by taking the wavelet transform in a neighborhood around each voxel. The process is shown graphically in Figure 2.2. To construct the attribute vector of a voxel $\mathbf{x}_0$ under consideration, the discrete wavelet transform (DWT) decomposition of the image data within a sliding window centered at $_0$, $I_o(\mathbf{x})$ is performed. We use length-2 Daubechies wavelet, which has orthogonal and symmetric basis functions. The low-pass and high-pass filters are $[1/\sqrt{2}, 1/\sqrt{2}]$ and $[1/\sqrt{2}, -1/\sqrt{2}]$, respectively. The wavelet decomposition is invariant to translations, since we compute it in a window centered at the voxel of interest. However, these are not rotation invariant, and we need to make these rotation invariant. In order to do this we combine three high-pass sub-images at each level into one subimage. The feature subimage $L^{(j)}_{\mathbf{x}_0}(\mathbf{x})$ at level j is calculated as

$$L^{(j)}_{\mathbf{x}_0}(\mathbf{x}) = \sqrt{\left|I^{(j)}_{LLH}(\mathbf{x})\right|^2 + \left|I^{(j)}_{LHL}(\mathbf{x})\right|^2 + \left|I^{(j)}_{HLL}(\mathbf{x})\right|^2}$$

where $I^{(j)}_{LLH}(\mathbf{x})$, $I^{(j)}_{LHL}(\mathbf{x})$ and $I^{(j)}_{HLL}(\mathbf{x})$ are the selected high-pass sub-images at DWT level j. Radial profiling is used to compute the attribute vector from the DWT image, $L^{(j)}_{\mathbf{x}_0}(\mathbf{x})$ at

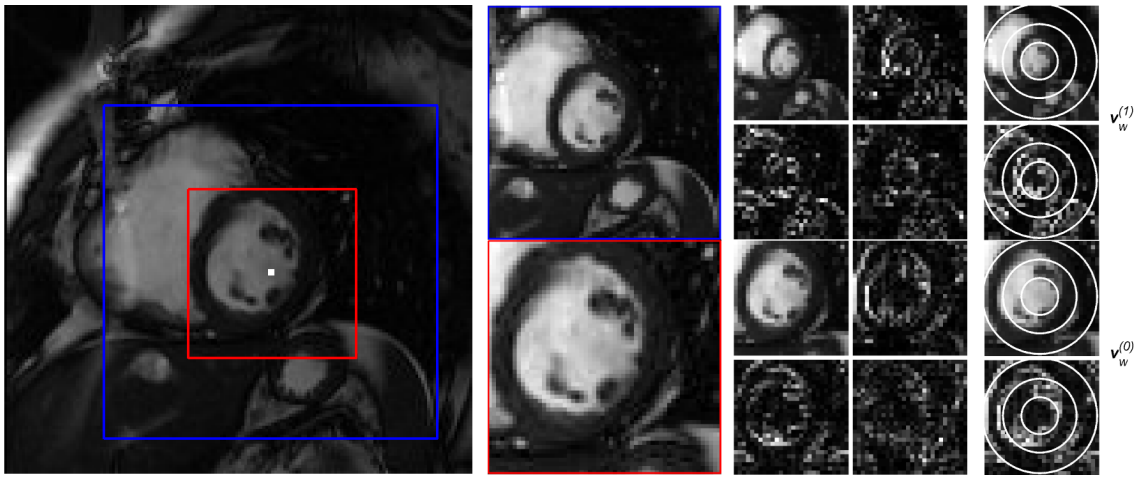


Figure 2.2: *Two dimensional illustration of the construction of the wavelet attribute vectors in a multiresolution framework. (a) Input image: the rectangles refer to the sliding windows centered at the current voxel; the large neighborhood corresponds to the relatively low resolution, and the smaller neighborhood to the relatively higher resolution. (b) the image data inside the sliding windows; top: the down-sampled image inside the large window (low-resolution image); bottom: the image inside the small window (high-resolution image). (c) Wavelet decompositions of the images shown in (b); (d) Radial profiling is used to construct the attribute vectors at different resolutions,* $\mathbf{v}_w^{(0)}$ *and* $\mathbf{v}_w^{(1)}$*, and the wavelet attribute vector is* $\mathbf{v}_w = [\lambda_0' \mathbf{v}_w^{(0)}, \lambda_1' \mathbf{v}_w^{(1)}]$

level j. Combining all the J-level attribute vectors $\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_J$, the wavelet attribute vector of voxel $\mathbf{x}_0$ is.

$$\mathbf{W} = [\eta_1 \mathbf{w}_1^T, \eta_2 \mathbf{w}_2^T, \cdots, \eta_J \mathbf{w}_J^T, \eta_D \mathbf{w}_D^T]^T$$

where $\mathbf{w}_D$ is calculated from $I_{LLL}^{(j)}(\mathbf{x})$ using radial profiling. $\eta_1, \cdots, \eta_J$ and η_D are weighting coefficients for the attribute vectors at different levels.

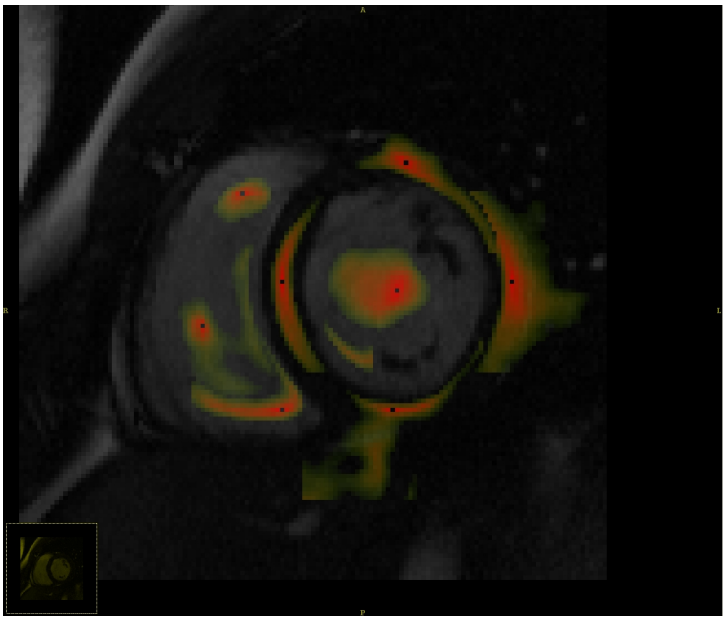


Figure 2.3: *Wavelet-based attribute vector similarity. The matchmaps, or the similarity in the neighborhood of select points are shown.*

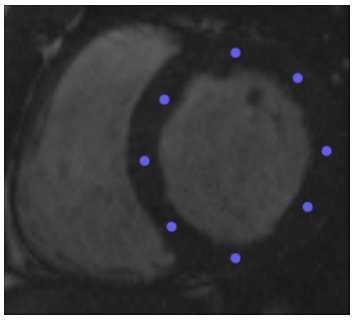
2.2.2 Wavelet attribute vector similarity function

The similarity between two wavelet attribute vectors $\mathbf{A}$ and $\mathbf{B}$ is given by the sum over all resolutions of the angle between the wavelet attributes. We can write this as,

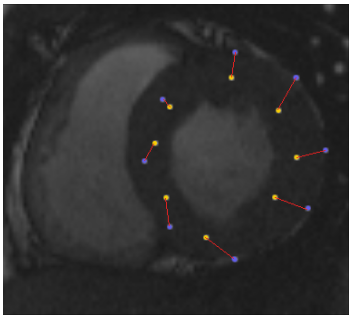
$$\varphi(\mathbf{A}, \mathbf{B}) = \sum_{r} \eta_r \frac{\langle \mathbf{a}_r, \mathbf{b}_r \rangle}{\|\mathbf{a}_r\| \, \|\mathbf{b}_r\|}$$

where, η_r is the weight applied to the attribute vector $\mathbf{w}_r$ at a given resolution r.

The effectiveness of the wavelet attribute in capturing the uniqueness of a given voxel can be seen in Figure 2.3. We evaluate the similarity of the attribute vector with the voxels in its neighborhood. As can be seen the attribute vectors are able to localize the voxel effectively. The neighboring voxels show high similarity to the given voxel. The similarity drops as we move away from the voxel. Another thing to notice is that the similarity also drops as we cross over to the blood from the myocardium and vice-versa. Thus the wavelet



(a) Original Frame on which the points were selected, with the points shown in blue.



(b) The Target Frame, showing the original points in blue. The best correspondence is shown in yellow

Figure 2.4: *Wavelet-based attribute vectors for evaluating deformations within the same subject.*

attribute vectors effectively capture the local and global features of a given voxel. One problem with the wavelet attribute that can be seen in Figure 2.3 is that we seem to get *reflections* about blood-myocardium boundaries. This happens to a large part because the wavelet attributes calculated by [92] are rotation invariant. For cardiac motion estimation it is actually preferable not to have rotation invariant attributes. This will be done in the future.

2.2.3 Selecting the best correspondence point

Instead of using inverse consistency for determining the best match for a given voxel as done in [92], we instead use an alternate strategy. We compute the similarity φ of the point p defined on the end-diastole frame, I_0, to points within a search radius, r, in the image I_t. We test the similarity between the point $\mathbf{p}$ on I_0 and the neighboring points, $\mathbf{p} + \boldsymbol{\epsilon}$, on I_t, where $\|\boldsymbol{\epsilon}_i\| \leq r$. The points in I_t together with their similarity values to the point $\mathbf{p}$ define a *matchmap* [34]. This can be seen in Figure 2.3. We select the best corresponding point by integrating over the matchmap. We want to weight the points by the similarity, therefore

we assign weights to the points based on their similarity,

$$\gamma_i = \varphi(\mathbf{W}_{I_0}(\mathbf{p}), \mathbf{W}_{I_t}((\mathbf{p} + \boldsymbol{\epsilon}_i)))$$

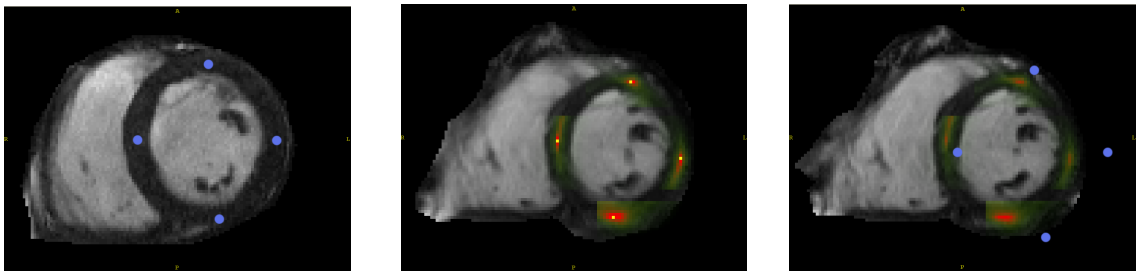
where, $\mathbf{W}_I(\mathbf{x})$ is the wavelet attribute vector at point $\mathbf{x}$ on image I, and $\mathbf{p} + \boldsymbol{\epsilon}_i$ is a point within the search radius. Using this, the best corresponding point, $\mathbf{x}_c$, is given by the weighted mean of the points, where the weights are γ_i defined above,

$$\mathbf{x}_c = \frac{\sum \gamma_i(\mathbf{p} + \boldsymbol{\epsilon}_i)}{\sum \gamma_i}$$

the summation being evaluated over a neighborhood of the point $\mathbf{p}$.

This makes the correspondence detection more robust and also imposes smoothness constraints. The results from this formulation can be seen in Figure 2.4. The best correspondence points are shown in yellow. These typically lie in another plane and are thus shown in two frames, one with the original points in blue and the other with the detected points. The twist within the myocardium is clearly visible. These results make us believe that along with smoothness constraints, the wavelet attributes along with the weighted correspondence estimation are best suited to extract cardiac motion fields.

We also observe that the wavelet attribute vectors are very effective in detecting correspondence across subjects. This can be seen in Figure 2.5. Here the points in the template image are shown in blue. The detected corresponding points are shown in yellow. As we can see, the attribute vectors are able to identify the corresponding points both the in the end-diastole image, i.e., the one with no deformation and the end-systole image, the one with maximum deformation, Figure 2.6. Thus we see that the wavelet-based attribute vectors are both robust and accurate across time and across subjects. Also, the similarity computation between the attribute vectors, described in Section 2.2.2 is very fast and thus makes the actual registration very efficient.

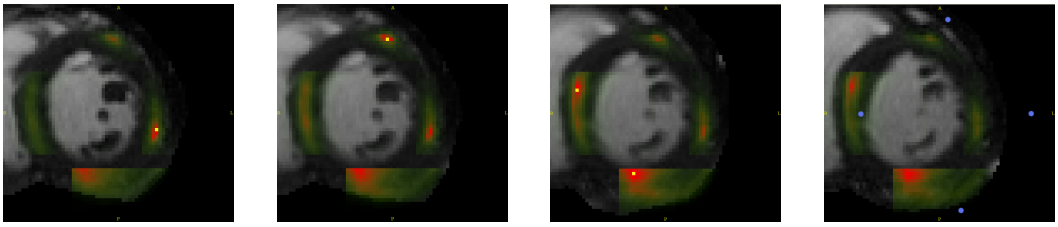


(a) The template image on which the points are selected. Shown here in blue.

(b) The subject image, same frame, adjacent slice. The best correspondence is shown in yellow.

(c) The subject image, same frame, same slice. We show the selected points in the template space, overlaid in blue.

Figure 2.5: *Attribute similarity across different Subjects*



(a) The subject image shown further along cardiac cycle, slice - 3. One of the points is detected on this slice

(b) The subject image shown further along cardiac cycle, slice - 2. One of the points is detected on this slice

(c) The subject image shown further along cardiac cycle, slice - 1. Two of the points are detected on this slice

(d) The subject image shown further along cardiac cycle, the same slice. The points picked in the template space are shown in blue.

Figure 2.6: *Attribute similarity across different Subjects and different time frames*

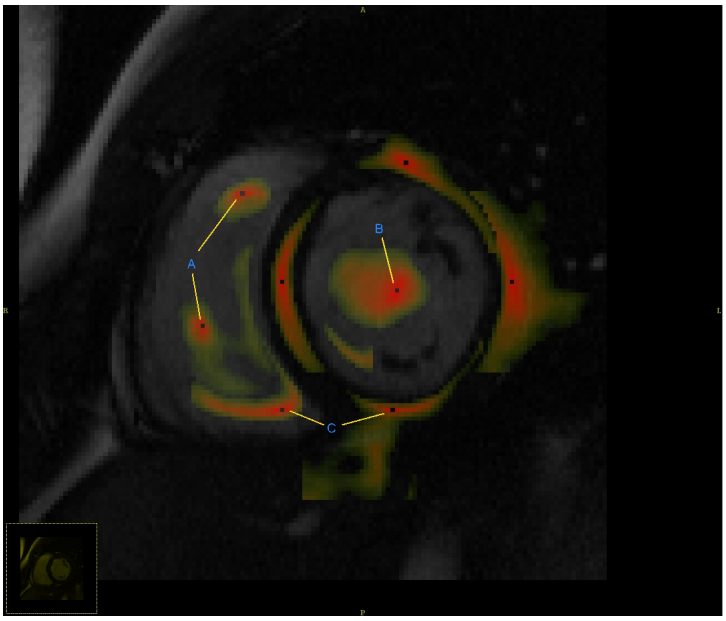
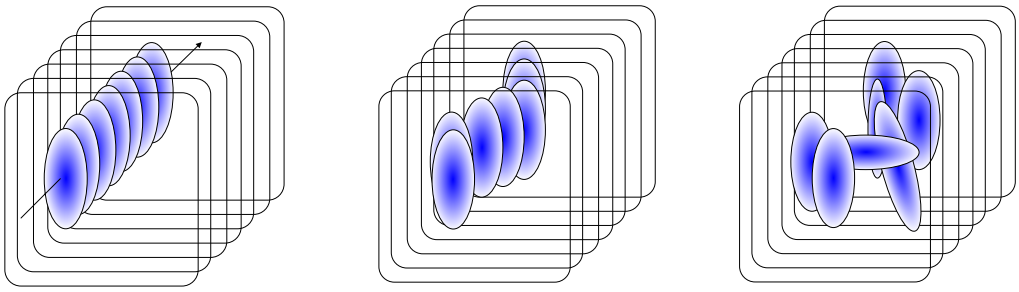


Figure 2.7: *Different kinds of matchmaps. Points A are points with high distinctiveness as their spread is small. Point B has low distinctiveness as it has a large spread. Points C also have relatively low distinctiveness, since the spread is large along one of the axes*

2.2.4 Distinctiveness of attribute vectors

In order to be able to pick focus points to drive the registration, we need to be able to select attribute vectors based on how distinctive they are. Doing this will allow us to robustly perform the registration in a fast hierarchical manner. This also allows us to select focus points automatically based on the distinctiveness of the attribute vectors. Also we weight the similarity of the points, during the energy function evaluation by the distinctiveness of the point. If we perturb each point in the end-diastole image I_0 and evaluate the best correspondence point for each of these points, we will get a scatter of best correspondence points. The distribution of this scatter of points provides information about the robustness of each match to image I_t [34].

To understand how we define the distinctiveness of a given voxel based on the wavelet attributes, consider Figure 2.7. We can see that different voxels produce different scatters



(a) Distinct point that has low spatial spread and is static, i.e., has no motion

(b) Distinct point, that has low spatial spread and has temporally smooth motion

(c) Non distinct point that has large spatial spread and haphazard temporal motion

Figure 2.8: Distinctiveness of a point over a sequence

based on the local and global characteristics of the underlying image. Although we get sharp peaks for all three types, (A,B, and C), we observe that for **A** the scatter is localized and the spread is uniform and small in all directions. In case of **B**, the spread is more or less uniform but it is large. The distinctiveness of such points is low, because there are a lot of points which are relatively far from the actual point that have sufficiently high similarity. The confidence in the similarity of such points is lower since we could be sufficiently far from the best point (in mm) and still have a fairly high similarity. Such points can increase the error in registration, and should therefore be avoided. The third set of points, **C** is also not very distinct, because although they are heavily localized in one direction, there is a very large spread along the principal axes. Again such points can increase the error in registration and should be assigned low *distinctiveness*.

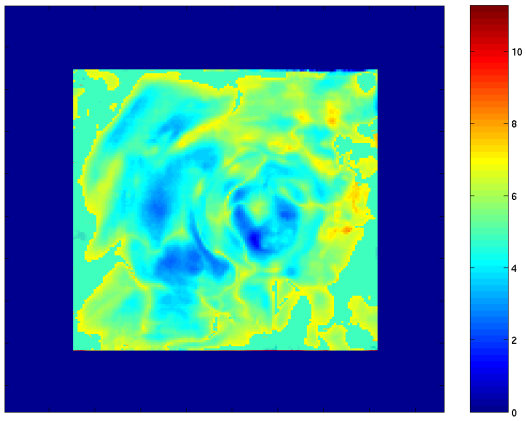
The situation is a bit different when we consider a sequence of such images, and their matchmaps over time. Some of the common cases that can occur are shown in Figure 2.8. One common case is of points that are localized spatially and temporally consistent, as shown in Figure 2.8(a). These kinds of points typically can be found outside the pericardium, i.e., tissues and bones which do not move much during the cardiac cycle. Figure 2.8(b) shows the case that is ideally preferred for points within the myocardium. These are

points whose spatial spread is low, and temporally smooth motion can be observed. Figure 2.8(c) shows the case of a point whose matchmaps are scattered across slices, and the confidence in estimating the location of this point across slices is low. Therefore, we assign a low distinctiveness to such points. This amounts to a distinctive point having a large spread along the time axis, and much smaller localized spreads along the spatial dimensions.

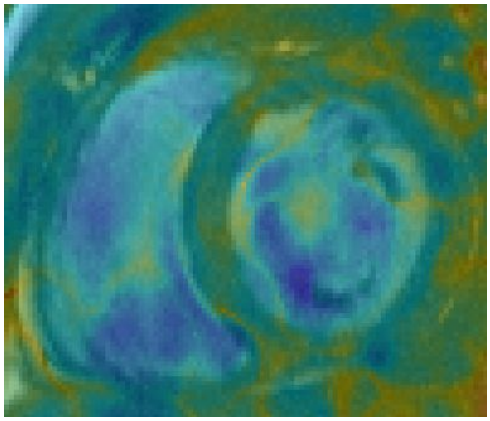
We extend this to the 4D case, where the only difference is that we wish for the spread along the time axis to be large, and the other three to be small. In order to compute the distinctiveness of the selected point, we calculate the eigenvalues and eigenvectors from the moment of inertia matrix obtained from the 4D matchmap. The 4D moment of inertia matrix of the matchmap is defined as,

$$I = \sum_i \gamma_i \begin{bmatrix} y_i^2 + z_i^2 + t_i^2 & -x_i y_i & -x_i z_i & -x_i t_i \\ -x_i y_i & x_i^2 + z_i^2 + t_i^2 & -y_i z_i & -y_i t_i \\ -x_i z_i & -y_i z_i & x_i^2 + y_i^2 + t_i^2 & -z_i t_i \\ -x_i t_i & -y_i t_i & -z_i t_i & x_i^2 + y_i^2 + z_i^2 \end{bmatrix} \tag{2.2.1}$$

We prefer points whose similarity matchmaps are spatially localized and compact. They should also be temporally smooth. This means that the 4D scatter in a 4D neighborhood for distinct points should be a 4D cylinder. We therefore compute the eigenvalues of the moment of inertia matrix, defined in (2.2.1), and assign high distinctiveness to all points which have one large eigenvalue and three small eigenvalues. Here we are basically trying to keep the spatial spread low. Because of the way the 4D matchmap is evaluated, the temporal spread will be large. For points which do not move much, the principal eigenvector will be aligned with the *time* axis. Correspondingly as the motion of the point in question increases, the angle between the principal eigenvector and the *time* axis will increase. Since we wish to give more importance to points with large motion. Therefore we scale the



(a) Distinctiveness calculated over the whole image. We can see that it is low along the peripheries and also within the blood pool.



(b) Overlay of the distinctiveness to show the distinctiveness on the myocardial region

Figure 2.9: The distinctiveness of voxels, based on the wavelet attribute vectors

distinctiveness factor by the angle between the principal eigenvector and the *time* axis.

The distinctiveness of the voxels as computed by this method is shown in Figure 2.9. As can be seen this is proportional to both the level of structural as well as motion relation features, which is what we are trying to extract. We can use this these *distinctiveness* values to compute a multiresolution set of *focus* points.

2.3 Estimating the Cardiac Motion Field

Cardiac motion estimation is the problem of determining a transformation that captures the motion of every point in the myocardium over the cardiac cycle. If we can register 3D images acquired at different phases of the cardiac cycle to each other, then we have an estimate of cardiac motion. Thus we can formulate the problem as one of being able to track (detect correspondence) of a small set of points on the myocardium over the entire cardiac cycle.

These points are the ones whose motion can be estimated most reliably. These are selected based on their distinctiveness, Section 2.2.4. This makes the registration more robust

as it is not affected by points with low distinctiveness, i.e., those points whose correspondence cannot be determined with high confidence. Another advantage of using the *focus* points is that it makes the similarity computation faster. It also reduces the dimensionality of the cost function, making the optimization more robust and faster. All these factors together make the current formulation robust and fast. Since the myocardium moves as one connected object and the motion is smooth, we can interpolate the transformation at the focus points to get the transformation over the whole myocardium.

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 measure to drive the motion estimation, and typically incorporate a regularizer to smooth the deformation field. We propose to instead maximize the similarity between the wavelet attributes, subject to the motion estimate constrained by a mechanical model of the heart, which has been discussed in Section 3.

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

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

where I_0 is the end-diastolic image. We define the motion field as the transformation $\boldsymbol{\chi}(\mathbf{x}, t)$ defined over the image space. The transformation maps a point $\mathbf{x}$ 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 2.10. The displacement field $\mathbf{U}(\mathbf{x}, t)$ 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, $\boldsymbol{\chi}(\mathbf{x}, t) = \mathbf{x} + \mathbf{U}(\mathbf{x}, t)$.

A sparse set of focus points $\mathbf{p} \in \Omega$ needs to be defined on the end-diastole image I_0. This can either be done manually, by the user, or using some feature extraction policy.

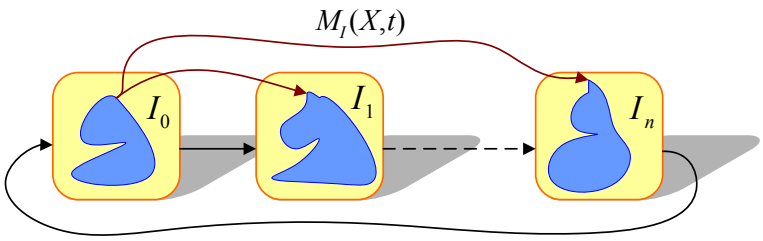


Figure 2.10: Formulation of the cardiac motion extraction problem.

Automatic focus point selection as described in Section 2.2.4 is used to select focus points at different resolutions to be used with the multiresolution approach. These points are tracked through the entire sequence $I(\mathbf{x}, t)$ in order to determine the correspondences and estimate the motion field $\boldsymbol{\chi}(\mathbf{x}, t)$. We solve an energy minimization problem where we solve for the deformation vectors at each of these points at each time frame.

To allow the registration algorithm to focus on different sets of points adaptively during different stages of image registration, each point should have it's own energy term and the overall energy function should be a weighted summation of energy terms of all the points. Therefore, by hierarchically assigning weights, $\eta(\mathbf{x})$, according to the distinctiveness of the attribute vectors (Section 2.2.4), we can focus on the most suitable points to actively drive the image registration. This increases the reliability of the motion estimation as it is not affected by the similarity estimates of unreliable points. The other benefit is that of improved speed of the motion estimation. If we were to solve for the displacement at every grid point on the image, then the dimensionality of the cost function that we are minimizing will be extremely high. Also the cost function evaluation would be much more expensive. Solving for the displacements at only the focus points, allows us to reduce the dimensionality of the problem and also makes the cost function evaluation much faster. Thus the procedure approximates a very high-dimensional cost function (equal to the number of points in the image sequence) by a significantly lower-dimensional cost function of only the ac-

tive points. This makes the minimization process much faster and also less susceptible to getting trapped in local minima, as it is a function of the focus points for which we can find relatively unambiguous matches. The method also performs much better than a simple subsampling of the original grid to reduce dimensionality, since in essence we create an adaptive grid that is dense in areas with more distinctive features and sparse in areas which are relatively static or where the confidence in the similarity is low.

We present the motion problem as a non-linear pde constrained optimization problem. Most systems can be thought of as having certain control parameters g, which directly or indirectly affect the process' state ϕ. An optimization problem can be thought of as one of finding controls g and states ϕ such that a cost functional $\mathcal{C}(\phi, g)$ is minimized subject to $\mathcal{F}(\phi, g) = 0$. Here the cost functional $\mathcal{C}(\phi, g)$ is a measure of the how close the current state is to the desired one. $\mathcal{F}(\phi, g) = 0$ is a constraint on the relationship between the states and the controls. For the problem of cardiac motion estimation, we solve for the forces τ in the myocardial fibers and obtain the states $\mathbf{u}$, the displacements produced as a result of these forces. The constraint, $\mathcal{F}(\mathbf{u}, \tau) = 0$ is the mechanical model described in Section 3,

$$\mathcal{F}(\mathbf{u}, \tau) = \rho_0 \frac{\partial^2 \mathbf{u}}{\partial t^2} - \text{Div}(\lambda \ln \mathcal{J}\mathbf{F}^{-T} + \mu(\mathbf{F} - \mathbf{F}^{-T})) - \sigma \mathbf{F}\mathbf{n} = 0 \quad \text{in } \Omega$$

The cost functional that we minimize, is the wavelet similarity term discussed earlier in Section 2.2.2. The similarity is computed between the wavelet attribute vectors at the point $\boldsymbol{\chi}(\mathbf{x}, t)$, $\mathbf{W}(\boldsymbol{\chi}(\mathbf{x}, t))$ and at the point $\boldsymbol{\chi}(\mathbf{x}, t+1)$, $\mathbf{W}(\boldsymbol{\chi}(\mathbf{x}, t+1))$. The Energy function for point similarity, summed over all the focus points, $\mathbf{p}$, over all time frames is,

$$\mathcal{C}(\mathbf{u}, \tau) = \sum_{t=0}^{N-1} \sum_{\mathbf{x} \in \{\mathbf{p}\}} \eta(\mathbf{x}) \left(\sum_{\mathbf{z} \in n(\mathbf{x}, 0)} \varphi \left(\mathbf{W}(\boldsymbol{\chi}(\mathbf{z}, t)), \mathbf{W}(\boldsymbol{\chi}(\mathbf{z}, t+1)) \right) \right) \tag{2.3.1}$$

The importance of each point $\mathbf{x}$, in the registration is determined by the corresponding

parameter $\eta(\mathbf{x})$, which is designed to be proportional to the distinctiveness of the point's wavelet-based attribute vector. The match for each point $(\mathbf{x}, t)$ is evaluated in it's 3-D neighborhood $n(\mathbf{x}, 0)$, by integrating the similarity measure between the attribute vector $\mathbf{W}(\boldsymbol{\chi}(\mathbf{z}, t))$ of every neighboring point $(\mathbf{z}, t)$ and the attribute vector $\mathbf{W}(\boldsymbol{\chi}(\mathbf{z}, t+1))$ of the corresponding point in the adjacent time frame of the sequence. The is equivalent to matching image patches or volumes between the two images because of the following two reasons,

- The attribute vector is calculated in a local neighborhood and therefore represents information in the neighborhood of the voxel in question.
- We sum over the similarities in the neighborhood of $\mathbf{x}$ when evaluating the wavelet similarity term (2.3.1).

The size of the neighborhood is large initially and decreases gradually with the progress of the deformation, thereby increasing robustness and accuracy of the motion estimation. As the neighborhood size is decreased we also change the weights for the multi-resolution attribute vectors similarity, increasing the weights for the fine features and decreasing the weights on the global features. This is equivalent to reducing the size of the image patches being matched as we move to the higher resolutions.

The optimization algorithm can be solved by using the following algorithm,

2.4 Periodicity Constraint

Since the image sequence is periodic, i.e., the first image I_0 and the last image I_{N-1} are also temporal neighbors, as shown in Figure 2.10. We impose a hard constraint on the deformations to return a point $\mathbf{x}$ to itself after the full cardiac cycle, i.e., $(\mathbf{x}, 0) = \chi(\mathbf{x}, N -$

1). This is done by ensuring that during each step of the minimization,

$$\sum_{\mathbf{x}\in\{\mathbf{p}\}} \|\boldsymbol{\chi}(\mathbf{x}, N-1) - (\mathbf{x}, 0)\| = 0 \tag{2.4.1}$$

s

Chapter 3

Mechanical Model of the Heart

3.1 Introduction

In this chapter we describe the anatomical structure of the human heart and describe how we translate that information to build a simple mechanical model of the heart. This mechanical model is used to constrain the problem of cardiac motion estimation, as shall be explained in detail in Section 5. 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. A comprehensive review of the field can be found in [59]. Muscle fiber orientations need to be considered while modeling cardiac electro-mechanics. 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 along the fiber direction. Therefore knowledge of the diffusion tensor or at least the fiber orientations is very important for modeling purposes, especially if the models are patient specific. It is not possible to obtain diffusion tensor images in vivo currently, and as a result most modeling approaches use synthetic data for the fiber orientations. These models for fiber orientations are very simple and capture only the basic

trends in the orientation. Since our goal is to estimate the motion of the heart, we ignore the electrical stimulation that activates the heart muscles and develops the forces in the myo-fibers. Instead we solve for the forces directly, and only model the mechanical aspects of the heart. For this we solve a linear elasticity equation at all the fibers. The model parameters are the fiber orientations, and the material properties. We first describe how these parameters are obtained, followed by the governing equations for the model.

3.2 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 [43, 59].

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 3.1. 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.

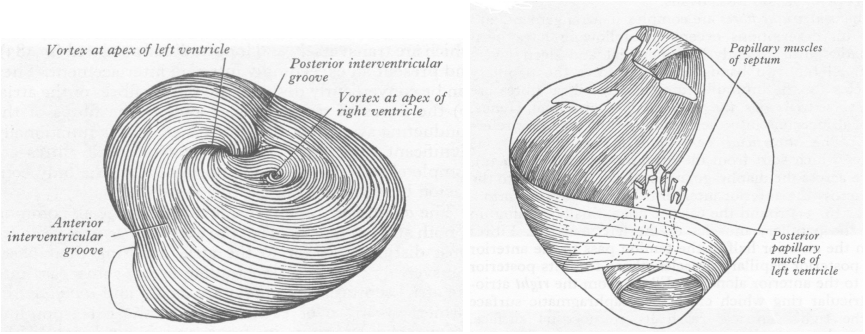


Figure 3.1: *Fiber orientations in the Human Heart showing the helical structure of the muscles (from [30])*

3.3 Mechanical Modeling

$$\rho\ddot{\mathbf{u}} - \mu\Delta\mathbf{u} + (\lambda+\mu)\nabla\mathbf{div}\ \mathbf{u} = \mathbf{f}R(\mathbf{u})\vec{\eta_0}$$

We model the heart as a linear elastic solid occupying a bounded region ω, with Dirichlet boundary conditions. Its displacement is described by

$$\nabla\cdot\left[\lambda(\mathbf{x})\left(\boldsymbol{\nabla}\cdot\mathbf{U}\right)\mathbf{I}+\mu(\mathbf{x})\left(\boldsymbol{\nabla}\mathbf{U}+\left(\boldsymbol{\nabla}\mathbf{U}\right)^{T}\right)\right]+\mathbf{f}R(\mathbf{U})\mathbf{N}_0=0\quad\text{in }\omega,\quad\mathbf{U}=\mathbf{g}\quad\text{on }\gamma. \tag{3.3.1}$$

Here $\mathbf{u}$ is the displacement field, and $\lambda(\mathbf{x})$ and $\mu(\mathbf{x})$ are the Lamé parameters which are related to the Young's Modulus $E(\mathbf{x})$ and Poisson's ratio $\nu(\mathbf{x})$. $\mathbf{R}(\mathbf{U})$ is the rotational component of the local displacement field, by which the fiber orientation $\mathbf{N}_0$ must be rotated. To solve (3.3.1) we embed ω in a regular domain Ω. We use trilinear finite elements to discretize (3.3.1), 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[1]. Gladilin [27] 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.

3.3.1 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:

$$\begin{aligned} \rho\frac{\partial^2\mathbf{u}}{\partial t^2} &= \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \mathbf{f} && \text{in } \Omega\times]0,T[, \\ \boldsymbol{\sigma}\mathbf{n} &= 0, && \text{in } \Gamma\times]0,T[, \\ \mathbf{u}(t=0) &= \mathbf{u}_0, && \text{in } \Omega \\ \dot{\mathbf{u}}(t=0) &= \mathbf{v}_0, && \text{in } \Omega \end{aligned} \tag{3.3.2}$$

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

$$\boldsymbol{\sigma} = \lambda \operatorname{tr}\boldsymbol{\varepsilon}\mathbf{I} + 2\mu\boldsymbol{\varepsilon}, \tag{3.3.3}$$

[1] 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 (3.3.1); see [37] for details.

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

$$\boldsymbol{\varepsilon} := \frac{1}{2}\left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^T\right] := \boldsymbol{\nabla}_s\mathbf{u}\ , \tag{3.3.4}$$

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

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

Using equation (3.3.4), the components of the strain tensor are

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

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

$$\int_\Omega \mathbf{w}\cdot(\rho\frac{\partial^2\mathbf{u}}{\partial t^2} - \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} - \mathbf{f})\ d\Omega + \int_\Gamma \mathbf{w}\cdot\boldsymbol{\sigma}\mathbf{n}\ d\Gamma = 0. \tag{3.3.5}$$

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

$$
\begin{aligned}
0 &= \int_\Omega w_i(\rho u_{i,tt} - \sigma_{ij,j} - f_i)\, d\Omega + \int_\Gamma w_i \sigma_{ij} n_j \, d\Gamma \\
&= \int_\Omega w_i \rho_i u_{i,tt}\, d\Omega - \left(\int_\Omega w_i \sigma_{ij,j}\, d\Omega \right) - \int_\Omega w_i f_i \, d\Omega + \int_\Gamma w_i \sigma_{ij} n_j \, d\Gamma \\
&= \int_\Omega w_i \rho_i u_{i,tt}\, d\Omega - \left(\int_\Omega (w_i \sigma_{ij})_{,j}\, d\Omega - \int_\Omega w_{i,j} \sigma_{ij}\, d\Omega \right) - \int_\Omega w_i f_i \, d\Omega + \int_\Gamma w_i \sigma_{ij} n_j \, d\Gamma \\
&= \int_\Omega w_i \rho_i u_{i,tt}\, d\Omega - \left(\int_\Gamma w_i \sigma_{ij} n_j \, d\Gamma - \int_\Omega w_{i,j} \sigma_{ij}\, d\Omega \right) - \int_\Omega w_i f_i \, d\Omega + \int_\Gamma w_i \sigma_{ij} n_j \, d\Gamma \\
&= \int_\Omega w_i \rho_i u_{i,tt}\, d\Omega + \int_\Omega w_{(i,j)} \sigma_{ij}\, d\Omega - \int_\Omega w_i f_i \, d\Omega, \qquad (3.3.6)
\end{aligned}
$$

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

$$
\int_\Omega \mathbf{w} \cdot \rho \ddot{\mathbf{u}}\, d\Omega - \int_\Omega \boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma}\, d\Omega = \int_\Omega \mathbf{w} \cdot \mathbf{f}\, d\Omega\,, \qquad (3.3.7)
$$

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 (3.3.7) motivates us to define the following bilinear forms:

$$
\begin{aligned}
a(\mathbf{w}, \mathbf{u}) &= -\int_\Omega \boldsymbol{\nabla}_s \mathbf{w} : \boldsymbol{\sigma}\, d\Omega, && (3.3.8) \\
(\mathbf{w}, \mathbf{f}) &= \int_\Omega \mathbf{w} \cdot \mathbf{f}\, d\Omega, \text{ and} && (3.3.9) \\
(\mathbf{w}, \rho\ddot{\mathbf{u}}) &= \int_\Omega \mathbf{w} \cdot \rho \ddot{\mathbf{u}}\, d\Omega. && (3.3.10)
\end{aligned}
$$

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

$$\begin{aligned} (\mathbf{w}, \rho\ddot{\mathbf{u}}) + a(\mathbf{w}, \mathbf{u}) &= (\mathbf{w}, \mathbf{f}), \\ (\mathbf{w}, \rho\mathbf{u}(0)) &= (\mathbf{w}, \rho\mathbf{u}_0), \\ (\mathbf{w}, \rho\dot{\mathbf{u}}(0)) &= (\mathbf{w}, \rho\boldsymbol{v}_0). \end{aligned} \tag{3.3.11}$$

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,

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \left\{ \begin{array}{c} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{12} \\ 2\epsilon_{23} \\ 2\epsilon_{31} \end{array} \right\} = \left\{ \begin{array}{c} u_{1,1} \\ u_{2,2} \\ u_{3,3} \\ u_{2,3} + u_{3,2} \\ u_{1,3} + u_{3,1} \\ u_{1,2} + u_{2,1} \end{array} \right\}$$

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 (3.3.3) can be written using the vector convention as

$$[\boldsymbol{\sigma}] = [\mathbf{D}][\boldsymbol{\varepsilon}], \tag{3.3.12}$$

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

$$\mathbf{D} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{3.3.13}$$

Since the matrix $[\mathbf{D}]$ is always symmetric, it follows that the integrand of the bilinear form in (3.3.8) can be written with the aid of (3.3.12) as

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

which shows that the bilinear form in (3.3.8) in indeed symmetric.

3.3.2 Semidiscrete Galerkin formulation of elastodynamics

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

$$\begin{aligned} (\mathbf{w}^h, \rho \ddot{\boldsymbol{v}}^h) + a(\mathbf{w}^h, \boldsymbol{v}^h) &= (\mathbf{w}^h, f) - (\mathbf{w}^h, \rho \ddot{\boldsymbol{g}}^h) - a(\mathbf{w}^h, \boldsymbol{g}^h) \\ (\mathbf{w}^h, \rho \boldsymbol{v}^h(0)) &= (\mathbf{w}^h, \rho \mathbf{u}_0) - (\mathbf{w}^h, \rho \boldsymbol{g}^h(0)) \\ (\mathbf{w}^h, \rho \dot{\boldsymbol{v}}^h(0)) &= (\mathbf{w}^h, \rho \dot{\mathbf{u}}_0) - (\mathbf{w}^h, \rho \dot{\boldsymbol{g}}^h(0)) \end{aligned} \tag{3.3.14}$$

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

$$\begin{aligned}
\mathbf{w}^h(\boldsymbol{x},t) = w_i^h(\boldsymbol{x},t)\boldsymbol{e}_i &= \sum_{A\in\eta-\eta_{q_i}} N_A(\boldsymbol{x})c_{iA}(t)\boldsymbol{e}_i \\
\boldsymbol{v}^h(\boldsymbol{x},t) = v_i^h(\boldsymbol{x},t)\boldsymbol{e}_i &= \sum_{A\in\eta-\eta_{g_i}} N_A(\boldsymbol{x})d_{iA}(t)\boldsymbol{e}_i \\
\boldsymbol{g}^h(\boldsymbol{x},t) = g_i^h(\boldsymbol{x},t)\boldsymbol{e}_i &= \sum_{A\in\eta_{g_i}} N_A(\boldsymbol{x})g_{iA}(t)\boldsymbol{e}_i
\end{aligned} \tag{3.3.15}$$

Substituting (14) into (13) we get, (ignoring $\boldsymbol{e}_i, \text{and } \boldsymbol{e}_j$ 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, $\boldsymbol{g} = 0, \text{and } \eta_{q_i} = 0$, therefore (10) reduces to a set of 3η equations in 3η unknowns,

$$\sum_{j=1}^{n_{\text{dof}}} \sum_{B\in\eta} (N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)\rho\ddot{d}_{jB}(t) + \sum_{j=1}^{n_{\text{dof}}} \sum_{B\in\eta} a(N_A\boldsymbol{e}_i, N_B\boldsymbol{e}_j)d_{jB}(t) = (N_A\boldsymbol{e}_i, \mathbf{f}) + (N_A\boldsymbol{e}_i, \mathfrak{h})_\Gamma \tag{3.3.16}$$

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,

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

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

$$\begin{aligned} \boldsymbol{m}^e &= \left[m^e_{pq}\right] \\ m^e_{pq} &= \delta_{ij} \int_{\Omega_e} N_a \rho N_b d\Omega \end{aligned} \tag{3.3.17}$$

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* $n_{\mathrm{ed}} n_{\mathrm{en}}$.

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

Similarly, the stiffness matrix can be written as,

$$\begin{aligned} \boldsymbol{K} &= \bigwedge_{e=1}^{n_{\text{el}}} (\boldsymbol{k}^e) \\ \boldsymbol{k}^e &= \left[k_{pq}^e\right] \\ k_{pq}^e &= \boldsymbol{e}_i^T \int_{\Omega_e} \boldsymbol{B}_a^T \boldsymbol{D} \boldsymbol{B}_a d\Omega \boldsymbol{e}_j \end{aligned} \tag{3.3.18}$$

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,

$$\begin{aligned} \mathbf{f}(t) &= \mathbf{f}_{\text{nodal}}(t) + \bigwedge_{e=1}^{n_{\text{el}}} (\mathbf{f}^e(t)) \\ \mathbf{f}^e &= \{f_p^e\} \\ f_p^e &= \int_{\Omega_e} N_a f_i d\Omega + \int_{\Gamma_{\mathfrak{h}_i}} N_a \mathfrak{h}_i d\Gamma \end{aligned} \tag{3.3.19}$$

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

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

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 $\boldsymbol{d}_0 = \dot{\boldsymbol{d}}_0 = \mathbf{0}$.

3.3.3 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 $\beta = 1/4 \,\text{and}\, \gamma = 1/2$. We first define the predictors,

$$\begin{aligned}
\tilde{\boldsymbol{d}}_{n+1} &= \boldsymbol{d}_n + \Delta t \boldsymbol{v}_n + \frac{\Delta t^2}{4} \boldsymbol{a}_n \\
\tilde{\boldsymbol{v}}_{n+1} &= \boldsymbol{v}_n + \frac{\Delta t}{2} \boldsymbol{a}_n
\end{aligned} \tag{3.3.20}$$

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

$$\left(\boldsymbol{M} + \frac{\Delta t^2}{2} \boldsymbol{K} \right) \boldsymbol{a}_{n+1} = \mathbf{f}_{n+1} - \boldsymbol{K} \tilde{\boldsymbol{d}}_{n+1} \tag{3.3.21}$$

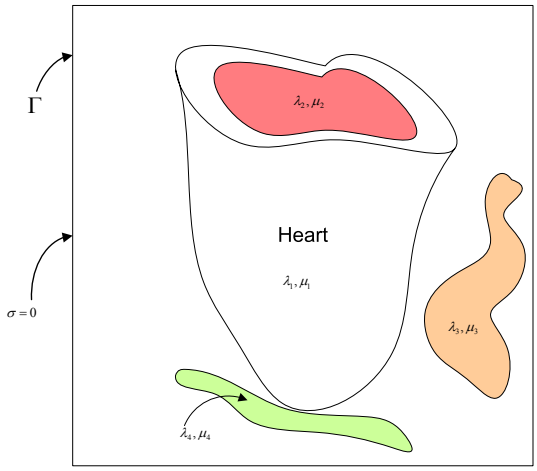


Figure 3.2: *The material properties and boundary conditions for the cardiac model. We set the stress, $\sigma = 0$, at the boundary Γ. Different Lamé parameters are selected for the myocardium (λ_1, μ_1), blood (λ_2, μ_2), the lungs (λ_3, μ_3) and for bone (λ_4, μ_4)*

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

$$\begin{aligned} \boldsymbol{d}_{n+1} &= \tilde{\boldsymbol{d}}_{n+1} + \frac{\Delta t^2}{4}\boldsymbol{a}_{n+1} \\ \boldsymbol{v}_{n+1} &= \tilde{\boldsymbol{v}}_{n+1} + \frac{\Delta t}{2}\boldsymbol{a}_{n+1} \end{aligned} \tag{3.3.22}$$

Imposing boundary conditions.

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 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 3.2.

Multigrid acceleration.

Multigrid methodologies have revolutionized scientific computation, especially for elliptic partial differential equations. Multigrid solvers consist of three main components: the smoother that reduces the algebraic residual at each level, and the restriction and prolongation operators for intergrid transfers [13]. Typical smoothers are stationary iterative solvers, e.g. Gauss-Seidel. The multigrid method works very well for constant coefficient PDEs, but slows down for strongly variable coefficient problems. In [2] a multigrid method for high-contrast materials is presented but is quite restrictive: material property jumps have to align with the grid. Algebraic multigrid is another alternative, but it requires an assembled matrix; this is costly and incompatible with our goals. Here we use a different approach. We are not using the multigrid algorithm to solve, but rather *to precondition* a Krylov method. Furthermore the smoothers within the multigrid iteration consist of a number of preconditioned Krylov iterations. We are using a Conjugate Gradient solver both to drive the overall residual, and as a smoother at each level. For high-contrast materials it is important to precondition the smoothers too; we have found that a simple damped Jacobi method suffices to obtain good algorithmic scalability. We use classical full-weighting and linear interpolation intergrid transfer operators. Extensive numerical tests have demonstrated robustness on high contrast inhomogeneities. Our code is developed on top of PETSc [7], a scientific computing library from Argonne National Laboratory.

3.4 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.

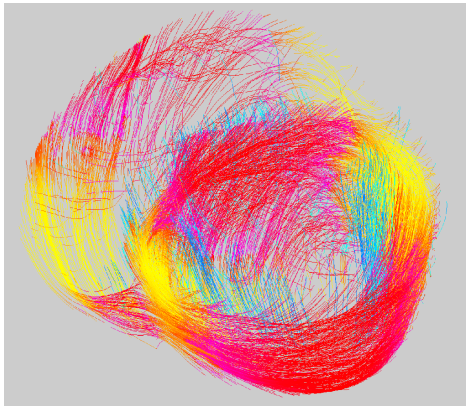


Figure 3.3: *Heart fiber orientation in the human heart, obtained from diffusion tensor imaging*

The principal eigenvector of the diffusion tensor is known to align with fiber tracts in the brain [56] and in the heart [64]. Heart fibers reconstructed from cardiac DTI are shown in Figure 3.3.

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:

$$\mathbf{D} = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix} \tag{3.4.1}$$

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 (3.4.1) 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, $\mathbf{e}_1$ [82, 64]. 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.

3.5 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 [66] 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. We test the effectiveness of the tensor remapping algorithm by comparing the mapped tensors with the ground truth diffusion tensors for 19 canine datasets[2]. The method of computing the transformation between the two geometries and the tensor reorientation algorithm are now described.

3.5.1 Deformable Image Registration

Image warping for deformable registration has received a great deal of attention during the past decade [96]. In the present work we used a very-high-dimensional elastic transformation procedure in 3D volume space, referred to as the hierarchical attribute matching 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 [66].

3.5.2 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

[2]CCBM, Johns Hopkins University

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 [91].

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 [91].

3.6 Results and Validation

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 3.4.

The error in the fiber orientations is further demonstrated on one slice in Figure 3.5, 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 3.6. We evaluate an error of less than 10° since it is close to the average error obtained from DTI imaging [64] and histological measurements[74]. 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 [59, 65].

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[86].

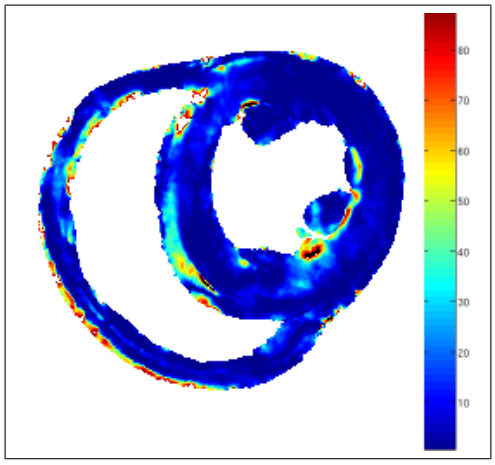


Figure 3.4: *The angle between the mapped principal direction with the actual principal direction, in degrees. The image is overlaid on a segmentation of the heart based on the fractional anisotropy*

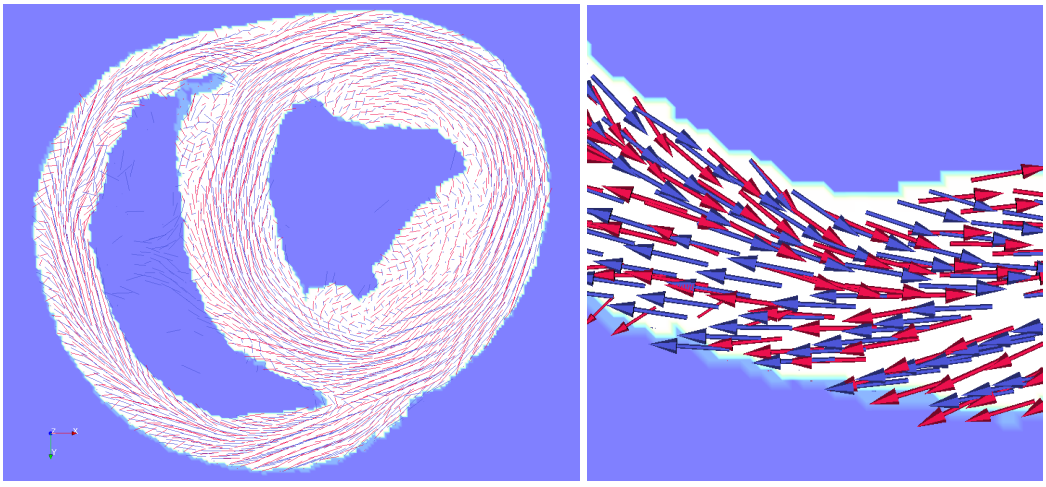


Figure 3.5: *The principal directions of the original DT are shown in blue. The mapped principal directions are shown in red. The glyphs are overlaid on a segmentation of the heart based on the fractional anisotropy of the mapped DT*

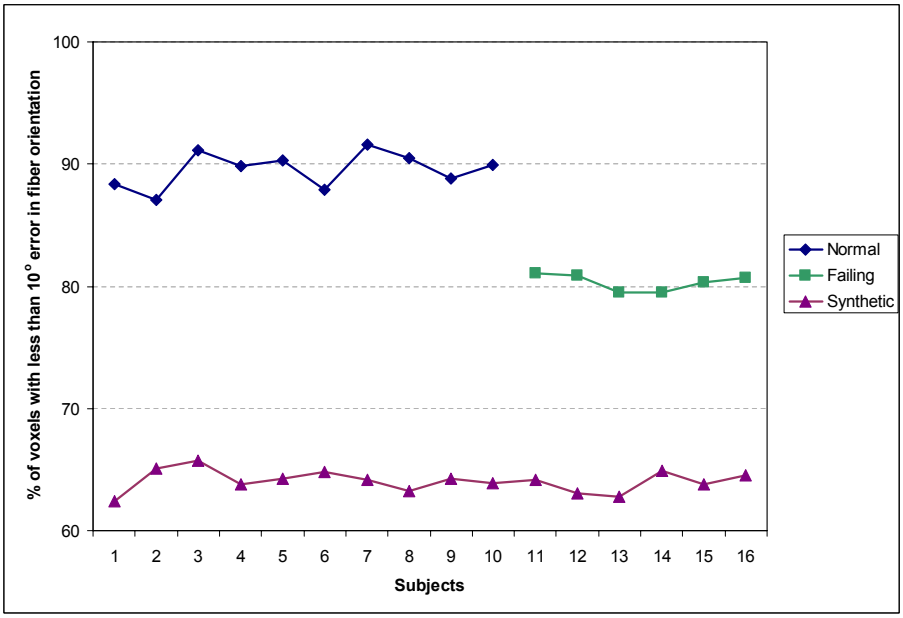


Figure 3.6: *The percentage of voxels having less than* 10° *error in the principal directions after mapping*

Chapter 4

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 [8, 39], adaptive mesh refinement methods [33, 57], and many-body algorithms [32, 35, 77, 88, 93, 94]. The earliest examples of tree-based spatial decompositions of $\mathcal{R}^n$ can be traced to the use of *binary space partitions* [25, 26, 63]. Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes. This subdivision gives rise to a representation of the space by means of a tree data structure known as a BSP tree. The BSP partitions its domain into two subregions and therefore the BSP tree is a binary tree. BSP trees can be used in spaces with any number of dimensions, but *quadtrees* [23] and *octrees* [49] are most useful in partitioning 2D and 3D domains, respectively. These use axis aligned lines and planes, respectively, instead of arbitrary hyperplanes and are efficient for the specific domains they are intended to work on.

Octrees and quadtrees are usually employed while solving two types of problems: searching and partitioning. 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 $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$. Similar benefits can be obtained by the use of *space-*

filling curves [81]. This equivalence has been used in linear quadtree and octree representations [10, 85], including the current work. Unstructured meshes are often preferred over uniform discretizations because they can be used with complicated domains and permit rapid grading from small to large elements. However, generating large unstructured meshes is a challenging task [68]. On the contrary, octree-based unstructured hexahedral meshes can be constructed efficiently [11, 31, 61, 62, 67, 84]. Although they are not suitable for highly complicated geometries, they provide a good compromise between adaptivity and simplicity for numerous applications like solid modeling [49], object representation [5, 14], visualization [24], image segmentation [73], adaptive mesh refinement [33, 57] and N-body simulations [32, 35, 77, 88, 93, 94].

Octree data structures used in discretizations of partial differential equations should satisfy certain spatial distribution of octant size [10, 85]. That is, adjacent octants should not differ greatly in size[1]. Furthermore, conforming discretizations require a so-called 'balance condition' that is necessary 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 *dangling* 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 an 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, 89, 94] and balance refinement [39, 85] of octrees, and the best known algorithms exhibit

[1]This is referred to as the 2:1 balance constraint. A formal definition of this constraint is given in section 4.1.2.

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, 85, 89, 94]. The proximity preserving property of space-filling curves makes them attractive for data partitioning. All 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 [85, 89, 94]. Section 4.2.1 describes the issues that arise in using a parallel top-down approach.

Bern et al. [10] 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. [85] 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 [45] 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.

Synopsis and Contributions. In this paper we present two parallel algorithms: one 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 built an additional auxiliary list of

[2]While we enforce the 0-*balance* constraint, [85] only enforce the 1-*balance* constraint. Note that it is harder to 0-*balance* a given octree. See section 4.1.2 for more details on the different balance constraints.

[3]There exists a unique least common balance refinement for a given octree [50].

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 [85]. The main parallel tools used are sample sorts (accelerated by biotonic sorts), and standard point-to-point/collective communication calls.[4]

In a nutshell, the major contributions of this work are:

- A parallel bottom-up algorithm for coarsening octrees, which is also used for partitioning the input in our other algorithms.
- A parallel bottom-up algorithm for constructing linear octrees. We avoid the synchronization issues that are usually associated with parallel top-down approaches to the problem.
- An algorithm for enforcing 2:1 balance refinement in parallel. 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 4.2.5 and 4.2.5, are the main hurdles in achieving good isogranular scalability.

Remark: The main parallel cost of the algorithm is that related to the parallel sorts that run in $\mathcal{O}(N \log N)$ work and $\mathcal{O}(\frac{N}{n_p} \log(\frac{N}{n_p}) + n_p \log(n_p))$ 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 on the ini-

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

tial distribution and the 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.

Organization of the paper. In section 4.1 we introduce some terminology that will be used in the rest of the paper. Section 4.2 describes the various components of our construction and balance refinement algorithms. In Section 4.3 we present numerical experiments, including fixed size and isogranular scalability tests on different data distributions. Finally, in Section 4.4 shortcomings of the proposed approach are discussed and some suggestions for future work are also offered. Table 4.1 summarizes the notation that is used in the subsequent sections.

4.1 Background

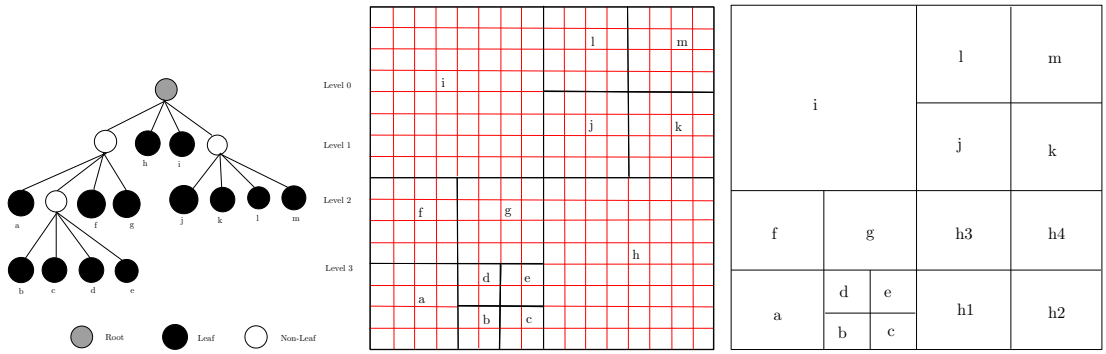
Octrees are trees in which every node has a maximum of eight children. They are analogous to binary trees (maximum of 2 children per node) in 1-D and quadtrees (maximum of 4 children per node) in 2-D. A node with no children is called a *leaf*. The only node with no parent is the *root*. Nodes that have 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 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. 4.1(a), the root of the tree is at level 0 and the children of any node are one level higher than the parent.

Octrees and quadtrees[5] can be used to partition cuboidal and rectangular regions, re-

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

Table 4.1: Notation

D_{max}	Maximum depth of the tree.
$\mathcal{L}(N)$	Level of octant N.
$\mathcal{P}(N)$	Parent 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.
$\{\ldots\}$	A set of elements.
$\emptyset$	The empty set.
$A \leftarrow B$	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.
A_{global}	Union of the list A from all the processors.
n_p	Total number of processors.
`Send(`A`,` r `)`	Send A to processor with rank $= r$.
`Receive()`	Receive from any processor.
N^p_{max}	Maximum number of points per octant.



(a) (b) (c)

Figure 4.1: (a) Tree representation of a quadtree and (b) decomposition of a square domain using the quadtree, superimposed over an uniform grid, and (c) a balanced linear quadtree: result of balancing the quadtree.

spectively (Fig. 4.1(b)). 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. To reduce storage costs, only the complete list of leaf nodes is stored, i.e., as a linear octree. 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*[6].

4.1.1 Morton encoding

In order to construct a Morton encoding, the maximum possible depth, D_{max}, of the tree is specified *a priori*. The domain is represented by an uniform grid of $2^{D_{max}}$ indivisible cells in each dimension (Fig. 4.1(b)). 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 corners, also known as its anchor, and its level in the tree (Fig. 4.2

[6]Morton encoding is one of many space-filling curves [17]. 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.

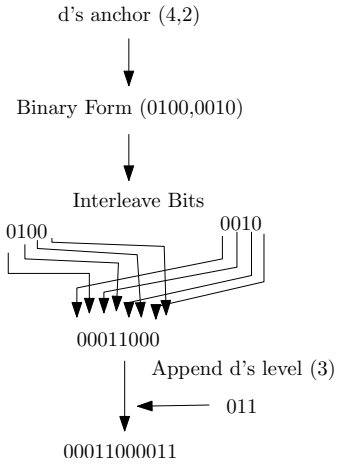


Figure 4.2: Computing the Morton id of quadrant 'd' in the quadtree shown in Fig. 4.1(b). The anchor for any quadrant is it's lower left corner.

).

The Morton encoding for any octant is derived by interleaving[7] 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. Interesting properties of the Morton encoding scheme are listed in Appendix 4.5. In the rest of the paper 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.

4.1.2 Balance Constraint

In many applications involving octrees, it is desirable that adjacent elements do not differ greatly in size [36, 39, 85]. Generalizing Moore's [50] categorization of the general balance conditions, we have the following definition:

Definition 1 *A linear d-tree is k-balanced if and only if, for any $l \in [1, D_{max})$, no leaf at*

[7]Instead of bit-interleaving as described here, we use a multicomponent version (Appendix 4.6) of the Morton encoding scheme.

level l shares an m-dimensional face[8] *($m \in [k, d)$) 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. An example of a 0-*balanced* quadtree is shown in Figure 4.1(c). The balance algorithm proposed in this 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.

4.2 Algorithms

4.2.1 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

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

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, 85, 89, 94] 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 4.2.4. 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[9]. This process is continued until no further splits are required. This procedure is summarized in Algorithm 1.

Next we describe algorithmic components that are fundamental to our construction and balance refinement algorithms: 1) the generation of a coarse linear octree between two

[9]Refer to the Appendix 4.8 on how to sample the points in order to construct the coarsest possible octree.

Algorithm 1 CONSTRUCTING A COMPLETE LINEAR OCTREE FROM A DISTRIBUTED LIST OF POINTS (PARALLEL) - `Points2Octree`

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: $\mathcal{O}(n \log n)$, where $n =$ len(L).
Storage: $\mathcal{O}(n)$, where $n =$ len(L).

1. $F \leftarrow$ [Octant$(p, D_{max}), \forall p \in L$]
2. Sort(F)
3. $B \leftarrow$ BlockPartition(F) (Algorithm 4)

4. **for each** $b \in B$
5. **if** NumberOfPoints(b) $> N^p_{max}$
6. $B \leftarrow B - b + \mathcal{C}(b)$
7. **end if**
8. **end for**

given octants; 2) generation of a complete linear octree from a partial set of octants; and 3) coarsening of octrees.

4.2.2 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 2) 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 between the two octants a and b.

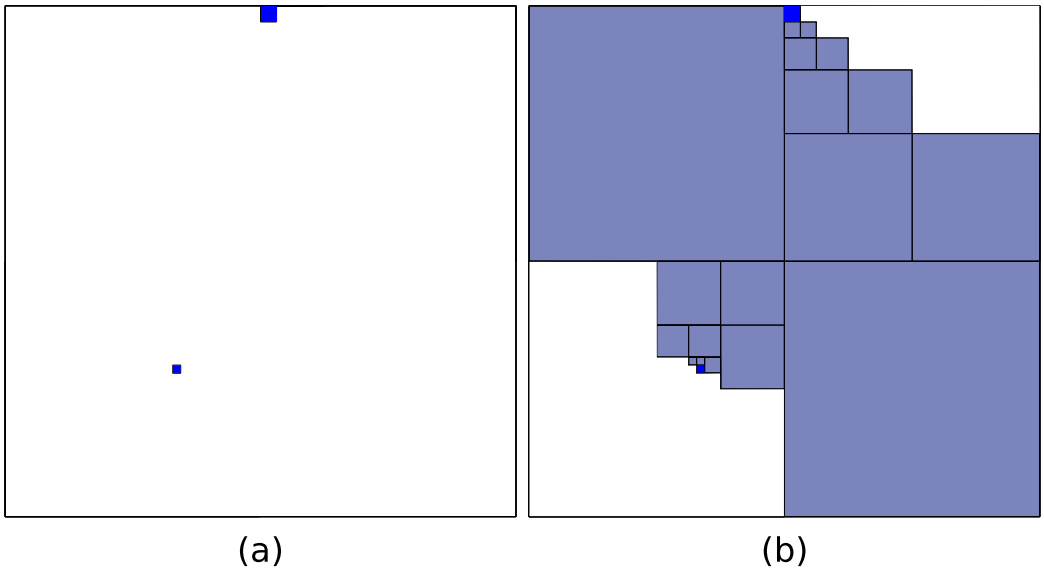


Figure 4.3: `(a)` Two cells: Input to `Algorithm 2`. `(b)` The minimal number of octants between the cells given in `(a)`. This is produced by using `(a)` as input to `Algorithm 2`.

This is illustrated in Figure 4.3. This algorithm is based on the Properties 3 and 4 of the Morton ordering, which are listed in Appendix 4.5.

4.2.3 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 4.4(a)), the octants are initially sorted based on the Morton ordering. Algorithm 8 is subsequently used to remove overlaps, if any. Two additional octants are 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, Property 7), which does not overlap the first given octant, and the second is the coarsest ancestor of the greatest possible octant (the deepest last descendant of the root octant, Property 9), which does not overlap the last given octant. The octants are distributed across the processors to get a weight-based uniform load distri-

Algorithm 2 CONSTRUCTING A MINIMAL LINEAR OCTREE BETWEEN TWO OCTANTS (SEQUENTIAL) - `CompleteRegion`

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

1. $W \leftarrow \mathcal{C}(\mathcal{A}_{finest}(a, b))$
2. **for each** $w \in W$
3. **if** $(a < w < b)$ `AND` $(w \notin \{\mathcal{A}(b)\})$
4. $R \leftarrow R + w$
5. **else if** $(w \in \{\{\mathcal{A}(a)\}, \{\mathcal{A}(b)\}\})$
6. $W \leftarrow W - w + \mathcal{C}(w)$
7. **end if**
8. **end for**
9. `Sort(R)`

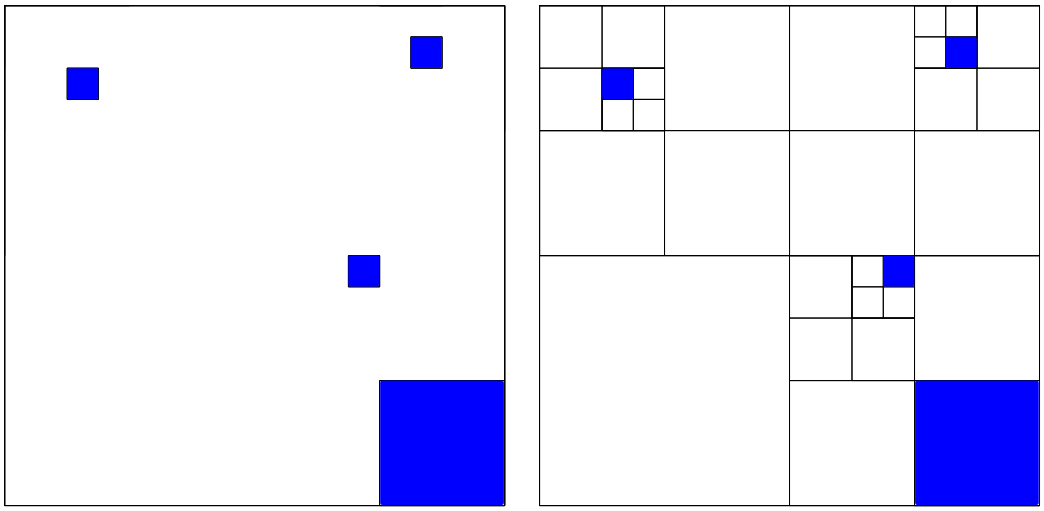


Figure 4.4: (a) A partial set of quadrants: Input to Algorithm 3. (b) A complete linear quadtree containing the cells in (a). This is produced by using (a) as input to Algorithm 3.

bution, and such that the last element on any processor is the same as the first element on the next processor. The local complete linear octree is subsequently generated by completing the region between every consecutive pair of octants as described in Section 4.2.2. The overlap guarantees that the union of these local complete octrees gives us a global complete and linear octree. We ignore the last octant on each processor, except the last, since these were replicated. This is illustrated in Figure 4.4.

4.2.4 Parallel bottom-up coarsening of octrees

Given a distributed list of leaves, we want to construct a complete linear coarse octree. We first sort the leaves according to their Morton ordering and then distribute them across the processors so that every processor has the same number of leaves. We select the least and the greatest octant at each processor (e.g., octants a and h from Figure 4.5(a)) and complete the region between them, as described in Section 4.2.2, to obtain a list of coarse

Algorithm 3 CONSTRUCTING A COMPLETE LINEAR OCTREE FROM A PARTIAL (INCOMPLETE) SET OF OCTANTS (PARALLEL) - `CompleteOctree`

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

```
1.  RemoveDuplicates(L)
2.  L ← Linearise(L) ( Algorithm 8 )
3.  Partition(L) ( Algorithm 5 )
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 2 )
18.     R ← R + L[i] + A
19. end for
20. if rank = (n_p − 1)
21.   R ← R + L[len(L)]
22. end if
```

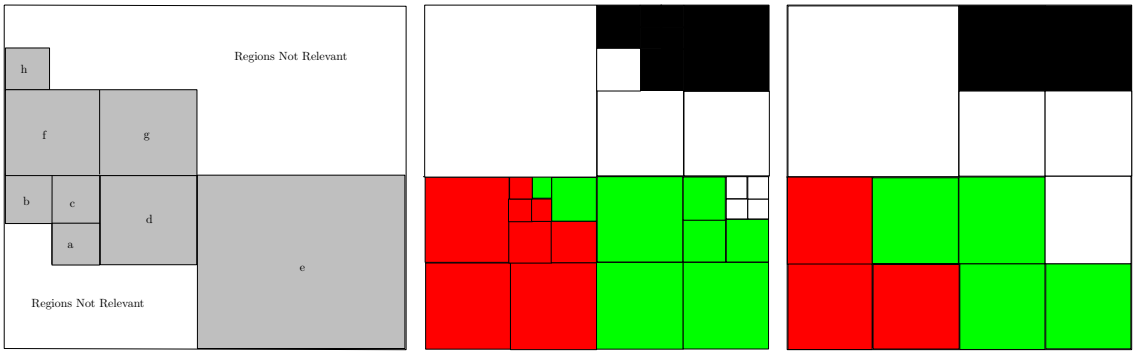


Figure 4.5: `(a)` A minimal list of quadrants covering the local domain on some processor, and `(b)` A Morton ordering based partition of a quadtree across `4` processors, and `(c)` Coarse quadrants and partition produced by using the quadtree shown in `(b)` as input to `Algorithm 4`.

octants. We then select the coarsest cell(s) out of this list of coarse octants (octant e in Figure 4.5(a)). We use the selected octants at each processor and construct a complete linear octree as described in Section 4.2.3. This gives us a global coarse complete linear octree that is based on the underlying data distribution[10].

Using the parallel coarsening algorithm for partitioning octants

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[11] 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. However, this approach does not avoid overlaps. Two desirable qualities of any partitioning strategy are load balancing, and minimization of

[10]Refer to the Appendix 4.7 for an estimate of the number of blocks produced.

[11]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 $n > p$.

overlap between the processor domains. We use our parallel bottom-up coarsening strategy to achieve these. First, we construct a global coarse linear octree based on the underlying distribution as described in section 4.2.4. In order to assign these coarse blocks to the processors, we first compute the load of each block by computing the number of original octants that lie within each of these blocks. The blocks are then distributed across the processors such that the total weight on each processor is roughly the same[12]. 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). The original octants are then partitioned to align with the coarse block boundaries. Algorithm 4 lists all the steps described above and Figures 4.5(b) and 4.5(c) illustrate a sample input to Algorithm 4 and the corresponding output, respectively.

4.2.5 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 4.1.2. 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

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

Algorithm 4 PARTITIONING OCTANTS INTO LARGE CONTIGUOUS BLOCKS (PARALLEL) - `BlockPartition`

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: $\mathcal{O}(n)$, where $n = \texttt{len}(F)$.
Storage: $\mathcal{O}(n)$, where $n = \texttt{len}(F)$.
Time: Refer to the Appendix 4.7.

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

Algorithm 5 PARTITIONING A DISTRIBUTED LIST OF OCTANTS (PARALLEL) – `Partition`

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

```
1.   S ← Scan( weight(W) )
2.   if rank = (n_p − 1)
3.     TotalWeight ← max(S)
4.     Broadcast(TotalWeight)
5.   end if
6.   w̄ ← TotalWeight / n_p
7.   k ← (TotalWeight) mod n_p
8.   for p ← 1 to n_p
9.       if p ≤ k
10.        Q ← {x ∈ W | (p − 1).(w̄ + 1) ≤ S(x) < p.(w̄ + 1)}
11.      else
12.        Q ← {x ∈ W | (p − 1).w̄ + k ≤ S(x) < p.w̄ + k}
13.      end if
14.      Q_tot ← Q_tot + Q
15.      Send(Q, (p − 1))
16.  end for
17.  R ← Receive()
18.  W ← W − Q_tot + R
```

so that the surface area and thereby the corresponding inter-processor boundaries are minimized. Determining whether a given partition provides the minimal surface area[13] is NP complete and determining the optimal partition is NP hard, since the problem is equivalent to the set-covering problem [19].

We use the parallel coarsening and partitioning algorithm (described in sections 4.2.4 and 4.2.4) 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 [10]. We describe this approach in Algorithm 6. In an alternative approach, the nodes search for neighbors and resolve any violations of the balance constraint [83, 85]. 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

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

reducing the cost of the searching operation. The complete linear octree is first partitioned into coarse blocks using the algorithm described in Section 4.2.4. 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 4.2.5; 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 [85] described in Section 4.2.5. The performance improvements from using the combined approach are presented in Section 4.3.2.

Balancing a local block

In principle, Algorithm 6 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 8 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 7.

We note that a list of octants form 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

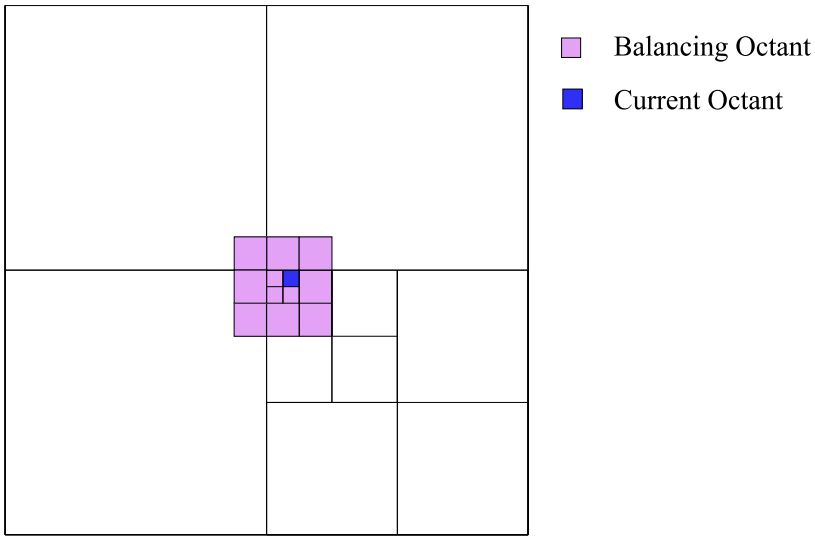


Figure 4.6: The minimal list of balancing quadrants for the current quadrant is shown. This list of quadrants is generated in one iteration of `Algorithm 6`.

4.6) 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 8 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 6). 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 7.

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 depends on the way the octree is stored; the pointer based approach being more popular [10, 83], 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

Algorithm 6 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: $\mathcal{O}(n \log n)$, where $n = \text{len}(R)$.
Storage: $\mathcal{O}(n)$, where $n = \text{len}(R)$.

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

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

Algorithm 7 BALANCING A LOCAL BLOCK (SEQUENTIAL) - `BalanceSubtree`

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 ← L,  P ← ∅,  R ← ∅

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 + N(P(t), l − 1)
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 8 )
```

Algorithm 8 REMOVING OVERLAPS FROM A SORTED LIST OF OCTANTS (SEQUENTIAL) - `Linearise`

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

the octants are sorted globally in the ascending Morton order, allowing us to search in $\mathcal{O}(\log n)$.

In order to find neighbors coarser than the current cell, we use the approach illustrated in Figure 4.7. First, the finest cell at the far corner (marked as 'Search Corner' in Figure 4.7) 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[14]. The neighbors (at the finest level) of this cell (N) are then selected and used as the search keys. These are denoted by $\mathcal{N}^s\left(N, D_{max}\right)$. The maximum lower bound[15] 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 no coarser neighbor can exist in that direction. This idea can be extended to incomplete

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

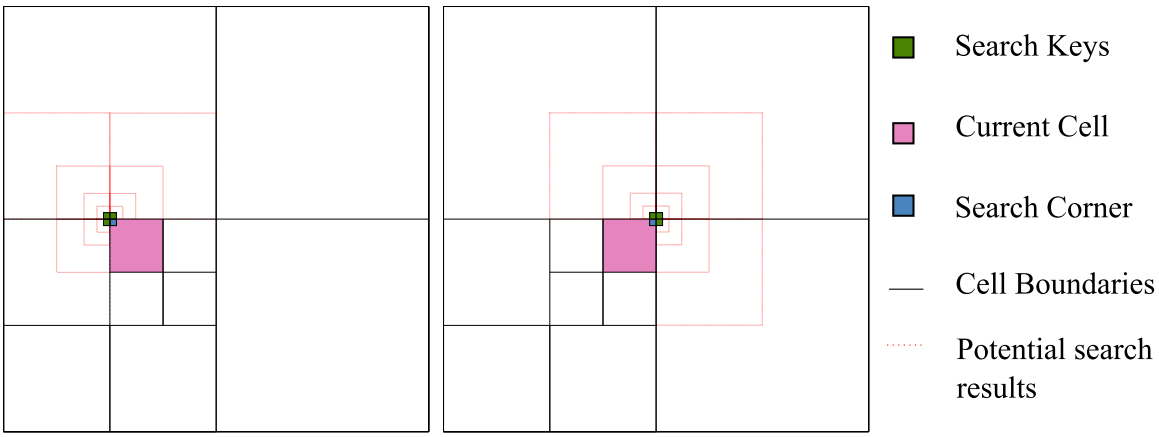


Figure 4.7: To find neighbors coarser than the current cell, we first select the finest cell at the far corner. The far corner is the one that is not shared with any of the current cell's siblings. The neighbors of this corner cell are determined and used as the search keys. The search returns the greatest 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.

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 9) of the *prioritized ripple propagation* algorithm first proposed by Tu et al. [83], 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[16] 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 3 or by using Algorithm 10, which is a variant of Algorithm 7. Both the algorithms perform equally

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

well.[17]

One difference with earlier versions of the ripple propagation algorithm is that our version works with incomplete domains. In addition, earlier approaches [10, 83, 85] 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[18] are required per octant. The work and storage costs incurred for balancing using the proposed algorithm to construct n balanced octants are $\mathcal{O}(n \log n)$ and $\mathcal{O}(n)$, respectively. This is true irrespective of the domain, including domains that are not simply connected.

Insulation against the ripple-effect

An interesting property of complete linear octrees is that a boundary octant cannot be finer than its internal neighbors[19] (Figure 4.8(a)) [83]. 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[20]. 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*

[17] 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.

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

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

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

Algorithm 9 RIPPLE PROPAGATION ON INCOMPLETE DOMAINS (SEQUENTIAL) – `Ripple`

Input: L, a sorted incomplete linear octree.
Output: W, a balanced incomplete linear octree.
Work: $\mathcal{O}(n \log n)$, where $n =$ len(L).
Storage: $\mathcal{O}(n)$, 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 4.2.5 )
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 10 )
16.          else
17.              R ← R + W[i]
18.          end if
19.      end for
20.      W ← R
21.  end for
```

Algorithm 10 COMPLETING A LOCAL BLOCK (SEQUENTIAL) - `CompleteSubtree`

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

```
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 8 )
```

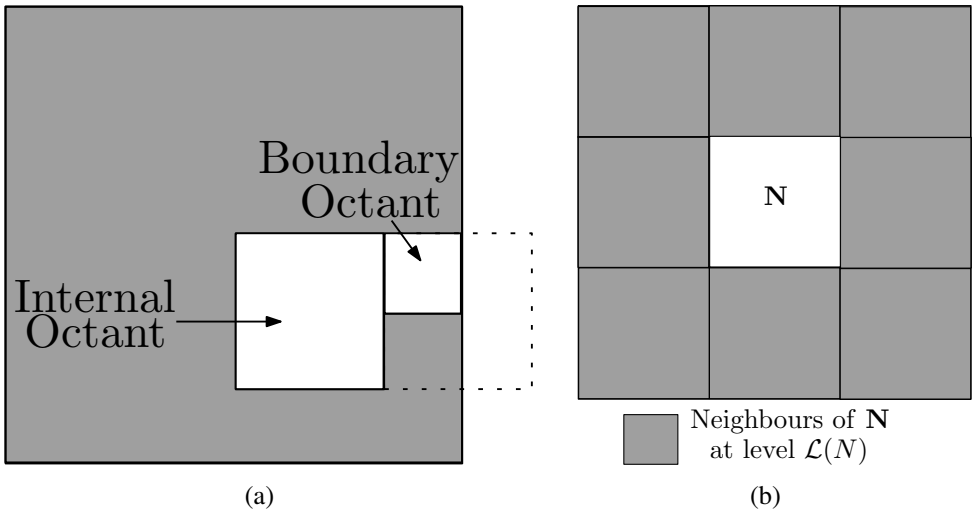


Figure 4.8: (a) A boundary octant cannot be finer than its internal neighbors, and (b) an illustration of an insulation layer around octant N. No octant outside this layer of insulation can force a split on N.

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 1 *No octant outside the insulation layer around octant* N *can force* N *to split (Figure 4.8(b)).*

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 4.2.5), 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

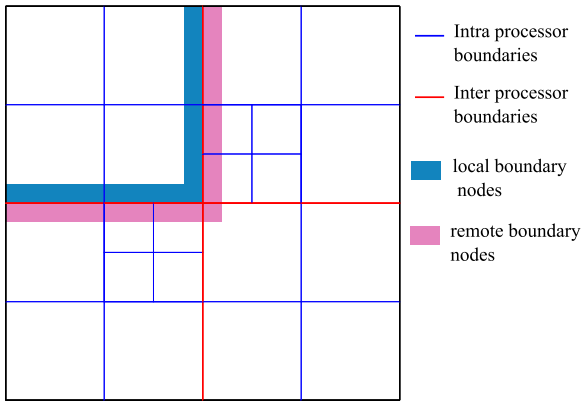


Figure 4.9: A coarse quadtree illustrating inter and intra processor boundaries. First, every processor balances each of its local blocks. Then, each processor balances the cells on its intra-processor boundaries. The octants that lie on inter-processor boundaries are then communicated to the respective processors and each processor balances the combined list of local and remote octants.

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 [85]. 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, 71]. 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 that lie on inter-processor boundaries and `(2)` an artificial insulation layer (Property 1) for the boundary octants can be constructed with little commu-

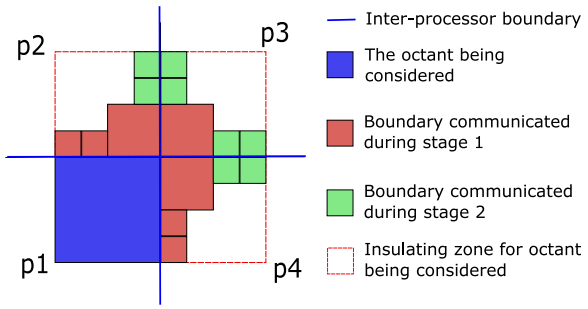


Figure 4.10: Communication for inter-processor balancing is done in two stages: First, every octant on the inter-processor boundary is communicated to processors that overlap with its insulation layer. Next, all the local inter-processor boundary octants that lie in the insulation layer of a remote octant received from another processor are communicated to that processor.

nication overhead (Section 4.2.5). The construction of this insulation layer is done in two stages (Figure 4.10): First, every local octant on the inter-processor boundary (Figure 4.9) 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 11 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 4.2.5). 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 4.2.5 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 4 and hence is impossible. Hence, assume that $c \in \{\mathcal{D}(a)\}$. By Property 9, $c \leq \mathcal{DLD}(a)$. Property 10 would then imply that $b \in \{\mathcal{D}(a)\}$. Property 5 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 and let it be sent to processor p during the first stage of communication. If all elements in $\mathcal{I}(N)$ overlap some octant in q or p, then the inter-processor boundary octants on p that overlap with some element in $\mathcal{I}(N)$ 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 an inter-processor boundary octant, a, on the 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 inter-processor boundary octant is sent to all its adjacent processors, a must have been sent to q during the first stage of communication.*

The keys steps involved in parallel balancing are summarized in Algorithm 11.

Communication costs for parallel balancing

Here, we compare the communication costs associated with the two approaches (upfront communication versus iterative communication). 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-

Algorithm 11 BALANCING COMPLETE LINEAR OCTREES (PARALLEL)

Input: A distributed sorted complete linear octree, L.
Output: A distributed complete balanced linear octree, R.
Work: $\mathcal{O}(n \log n)$, where $n =$ len(L).
Storage: $\mathcal{O}(n)$, where $n =$ len(L).
Time: Refer to Section 4.2.5.

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

processor boundary can be classified based on the *degree of the face*[21] 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.

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 insulation layer of some octant encloses the domains controlled by multiple processors, it is unlikely for dense octrees. Hence, it is reasonable to assume that in the first stage the octants are only communicated via point-to-point operations. Under this assumption, 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

$$N_u = \sum_{k=1}^{d} 2^{d-k} N_{k-1}. \tag{4.2.1}$$

Consider the example shown in Figure 4.11. 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 4.2.1. 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 algo-

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

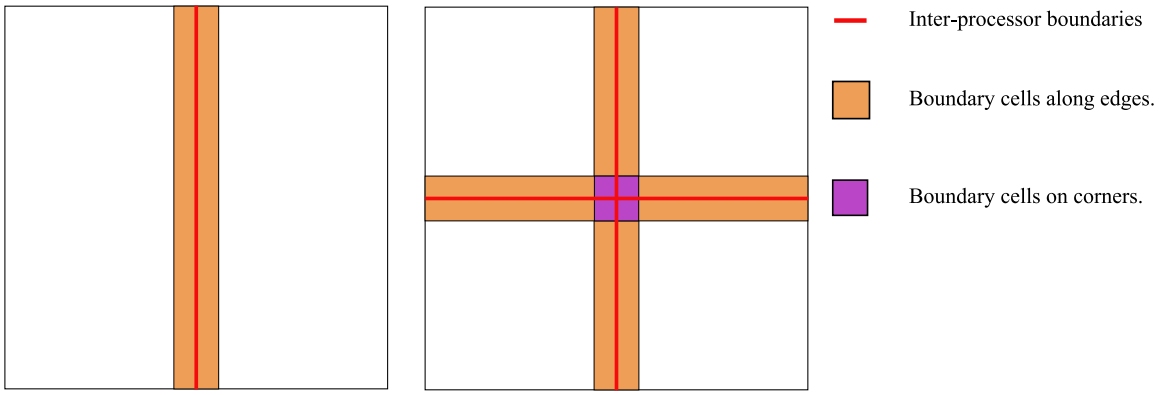


Figure 4.11: Cells that lie on the inter-processor boundaries. The figure on the left shows an inter-processor boundary involving 2 processors and the figure on the right shows an inter-processor boundary involving 4 processors.

rithm. Consider the example shown in Figure 4.10. 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 4.10, 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 edge or corner boundary octant. Therefore, we only communicate $\mathcal{O}(N_1 + N_0)$ 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

$$N_s = \sum_{k=1}^{d} 2^{k-1} N_{k-1}. \tag{4.2.2}$$

Again considering the example shown in Figure 4.11, 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 4.2.2. It is important to observe the difference between the communication estimates for upfront communication, 4.2.1, with that of the search based approach, 4.2.2. 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[22].

4.3 Results

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

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

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 the 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.

First, results from an experiment to compare different strategies for the local balancing stage is presented. This highlights the advantages of using the two-stage approach over existing approaches. Following this, results from the fixed-size and isogranular scalability analysis are presented.

4.3.1 Test Data

Point data of different sizes were generated for three different distributions; Gaussian, Log-normal and Regular. The regular distribution corresponds to a set of points uniformly distributed such that they are regularly spaced. Datasets were generated for all three distributions of increasing sizes that result in a balanced octree with octants ranging from 10^6(1M) to 10^9(1B). The fixed size scalability analysis was performed by selecting the 1M, 32M and 128M Gaussian point distributions to represent small, medium and large problems. Since the input sizes for different stages of the algorithm vary, we provide the input and output sizes for different stages in table 4.3.1. Since the regularly spaced points produce octrees that are inherently balanced, these datasets serve to estimate the communication costs associated with the parallel balancing algorithm.

Problem size	Gaussian			log-Normal			Regular	
	Points	Unbalanced Octants	Balanced Octants	Points	Unbalanced Octants	Balanced Octants	Points	Octants
1M	180K	607K	995.4K	180K	607K	987.5K	405.2K	994.9K
2M	361K	1211K	2000.5K	361K	1214K	2016.9K	2097.2K	2097.2K
4M	720K	2434K	3973.9K	720K	2434K	3958.5K	2.4M	4.06M
8M	1466K	4912K	8.0M	1466K	4920K	8.1M	3.24M	7.96M
16M	2886K	9686K	16M	2886K	9712K	16M	16.78M	16.78M
32M	5.8M	19.6M	31.9M	5.8M	19.6M	31.8M	19.25M	32.53M
64M	11.7M	39.3M	64.4M	11.7M	39.3M	64.7M	25.93M	63.67M
128M	23.5M	79.3M	130.6M	23.5M	79.4M	130.1M	134.22M	134.22M
256M	47M	158.6M	256.8M	47M	158.3M	256.4M	153.99M	260.24M
512M	94M	315.1M	516.9M	94M	315.5M	520.8M	168.20M	337.66M
1B	164.5M	553.6M	914.9M	164.5M	555.2M	912.2M	1.07B	1.07B

Table 4.2: Input and output sizes for the construction and balancing algorithms for the scalability experiments on Gaussian, Log-Normal, and Regular point distributions. Note that Regular point distributions are inherently balanced, and therefore the number of octants is only reported once.

4.3.2 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 strategies for local balancing, it does not involve any communication and hence was evaluated on a quad dual-core shared memory machine. We compared our two-stage approach, discussed in Section 4.2.5, with two other approaches; the first approach is the prioritized ripple propagation idea applied on the entire local domain [85], and the second approach is to use ripple propagation in 2 stages, where the local domain is first split into coarser blocks[23] 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 points. The results are shown in in Figure 4.12. All three approaches demonstrate good fixed size scalability, but the two-stage approach demonstrate better absolute runtime.

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

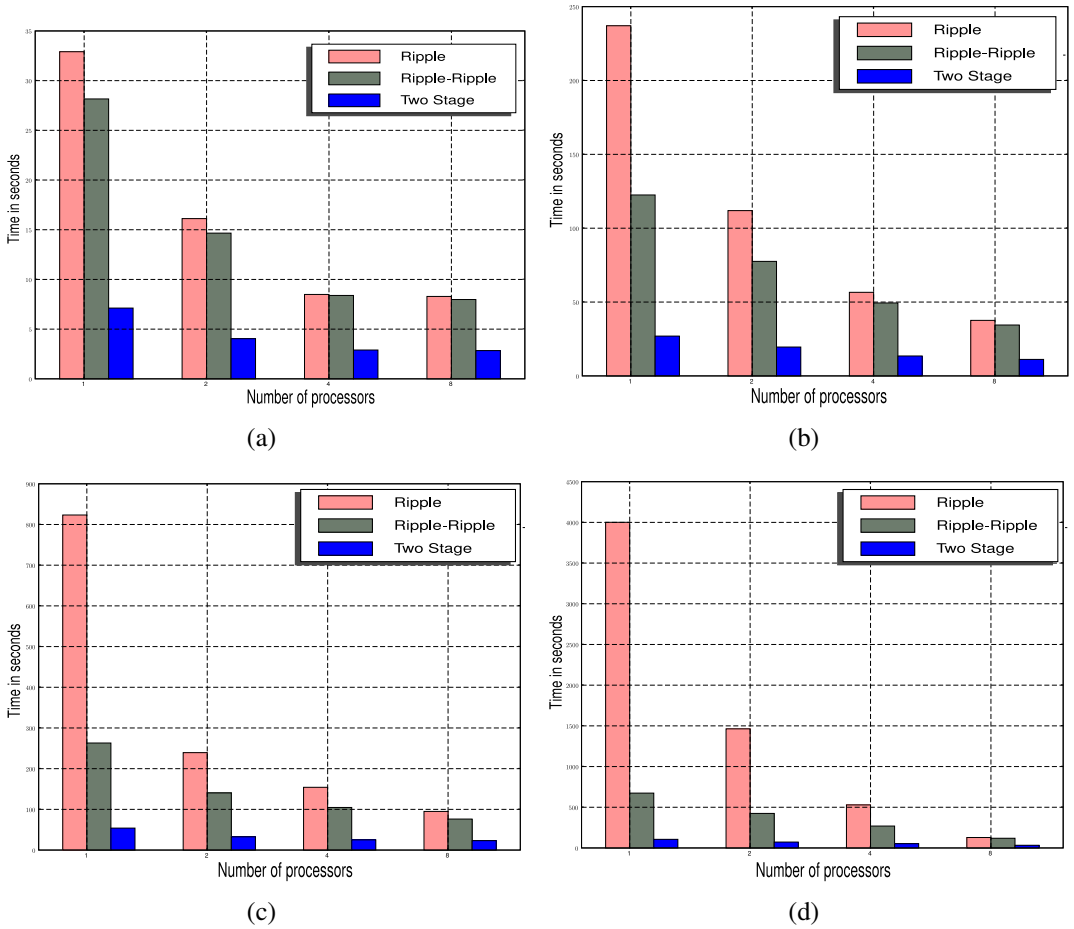


Figure 4.12: Comparison of three different approaches for balancing linear octrees `(a)` for a Gaussian distribution of `1M` octants, `(b)` for a Gaussian distribution of `4M` octants, `(c)` for a Gaussian distribution of `8M` octants, and `(d)` for a Gaussian distribution of `16M` octants.

4.3.3 Scalability analysis

In this section we provide experimental evidence on 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 analyze the improvement in performance when the problem size is kept constant and the number of processors are increased. Fixed size scalability allows us to determine the problem sizes for which speedup can be expected by increasing the processor count. 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 $\mathcal{O}(\frac{N}{n_p}\log\frac{N}{n_p} + n_p^2 \log n_p)$ if the samples are sorted using a serial sort. This causes problems when $\mathcal{O}(N) < \mathcal{O}(n_p^3)$ 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 given processor. Using the parallel bitonic sort to sort the samples [29] reduces the complexity to $\mathcal{O}(\frac{N}{n_p}\log\frac{N}{n_p} + n_p \log n_p)$.

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. In all cases the tests were performed with a maximum of one point per octant. This resulted in octrees with over 1 billion octants in some cases. 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 it can be seen in Table 4.3.1 that on an average the number of unbal-

anced octants is 3 times the number of input points, and that the number of octants doubles as a result of 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 regular grid .

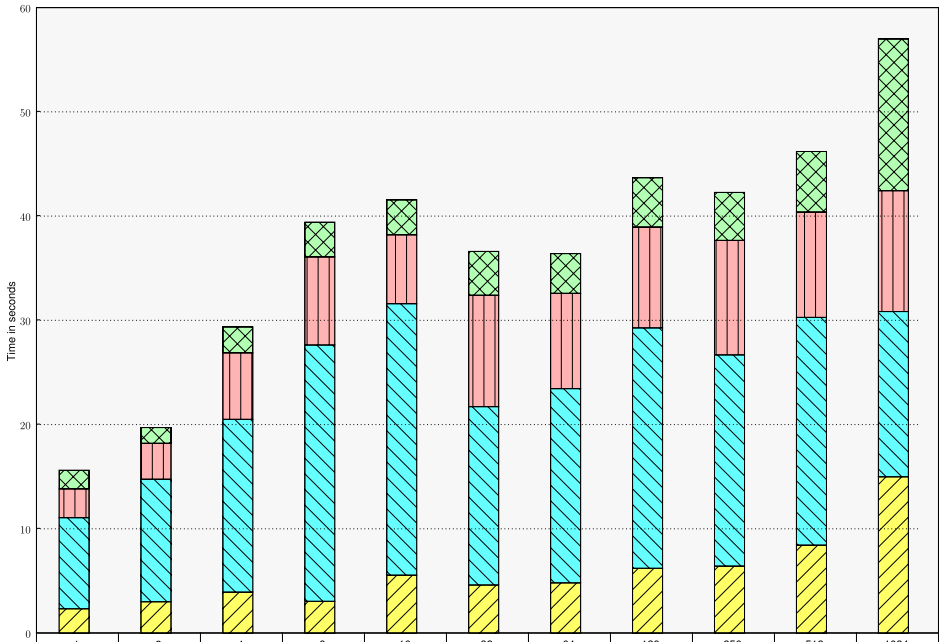
Wall-clock timings, speedup and efficiency for the isogranular analysis for the three distributions are shown in Figures 4.13, 4.14, and 4.15. The isogranular analysis for the regularly spaced distribution allows us to estimate the overhead involved in communication, partitioning and in balancing the local blocks using Algorithm 7. The ripple propagation algorithm does not do any work in this experiment. 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 4.16, 4.17 and 4.18.

4.4 Conclusions

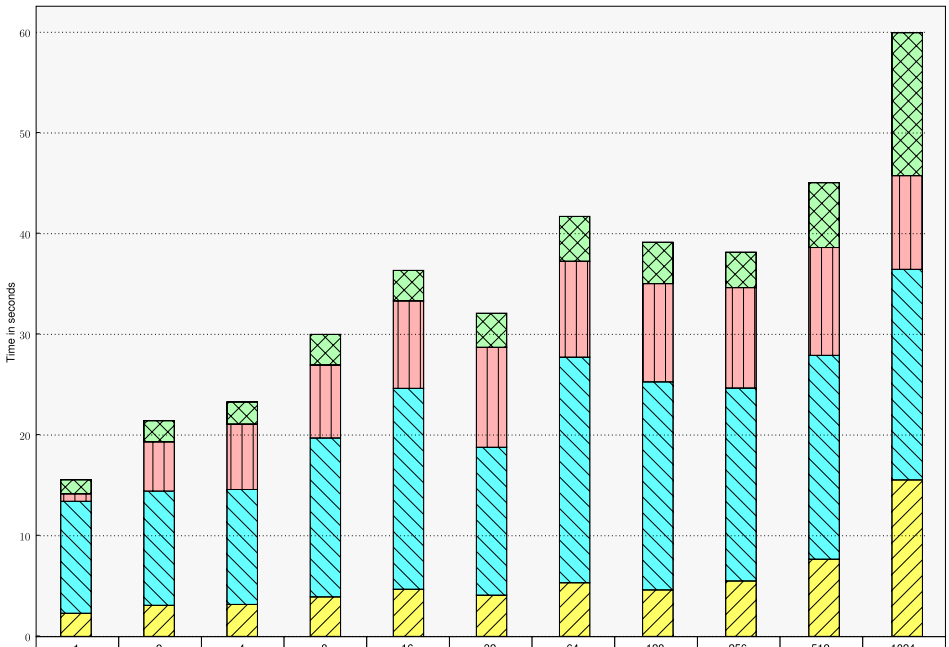
We have presented two new parallel algorithms for constructing and balancing large linear octrees on distributed memory machines. We have also tested MPI-based scalable parallel implementations for both the algorithms. Our algorithms have several important features:

- Experiments on three different types of input distributions demonstrate that the algorithms are insensitive to the underlying data distribution.
- Our algorithms avoid iterative communications and thus are able to achieve low absolute runtime and good scalability.



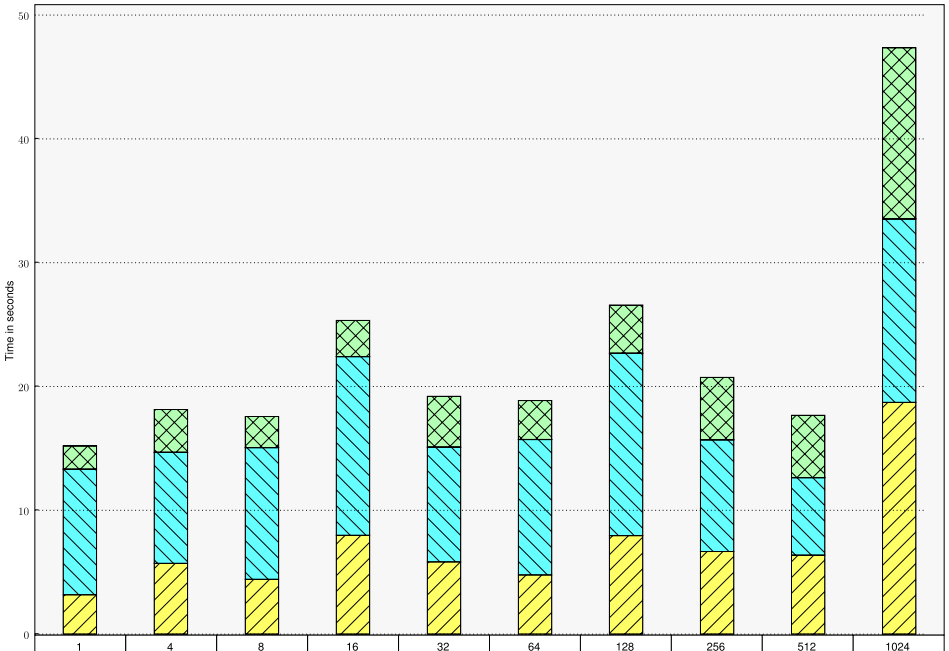
	1	2	4	8	16	32	64	128	256	512	1024
Construction	2.3	3.0	3.9	3.0	5.6	4.6	4.8	6.2	6.4	8.4	15
Internal Balance	8.7	11.8	16.6	24.6	26.0	17.1	18.6	23.1	20.3	21.8	15
Boundary Balance	2.8	3.4	6.4	8.4	6.6	10.7	9.1	9.7	11.0	10.1	11
Communication for Balance	1.8	1.5	2.5	3.3	3.4	4.2	3.8	4.7	4.6	5.8	14

Figure 4.13: 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)` communication (including related pre-processing and post-processing) during balance refinement `(Algorithm 11)`, `(2)` balancing across intra and inter processor boundaries `(Algorithm 9)`, `(3)` balancing the blocks `(Algorithm 7)` and `(4)` construction from points `(Algorithm 1)`.



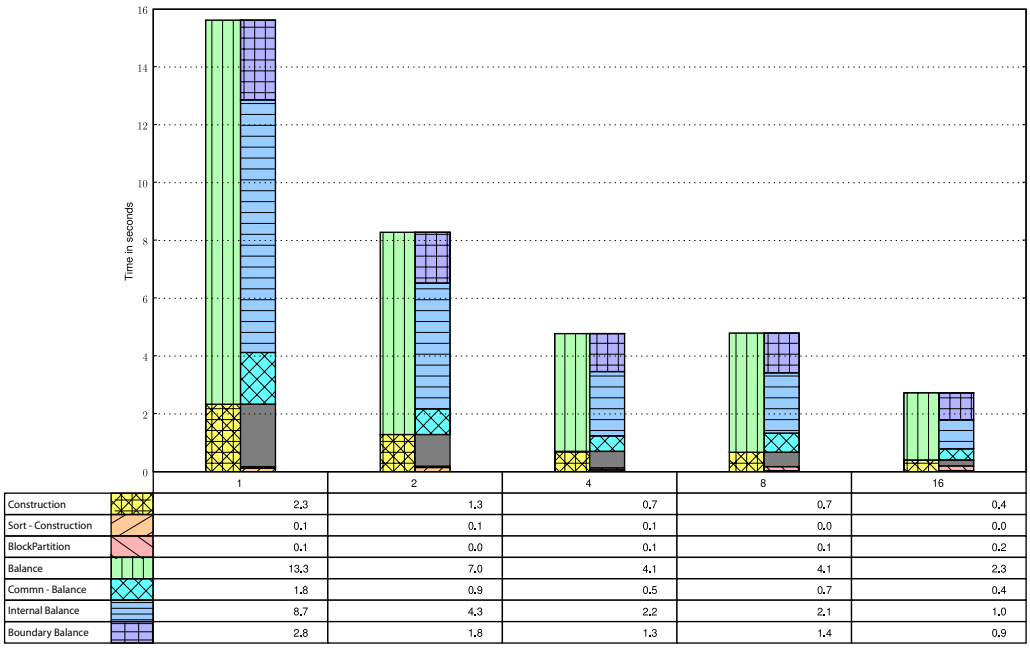
	1	2	4	8	16	32	64	128	256	512	1024
Construction	2.3	3.1	3.2	3.9	4.7	4.1	5.3	4.6	5.5	7.7	15
Internal Balance	11.1	11.3	11.4	15.8	20.0	14.7	22.4	20.7	19.2	20.2	20
Boundary Balance	0.7	4.9	6.5	7.2	8.7	9.9	9.5	9.8	10.0	10.7	9
Communication for Balance	1.4	2.1	2.2	3.1	3.0	3.4	4.5	4.1	3.5	6.4	14

Figure 4.14: Isogranular scalability for 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 11)`, `(2)` balancing across intra and inter processor boundaries `(Algorithm 9)`, `(3)` balancing the blocks `(Algorithm 7)` and `(4)` construction from points `(Algorithm 1)`.



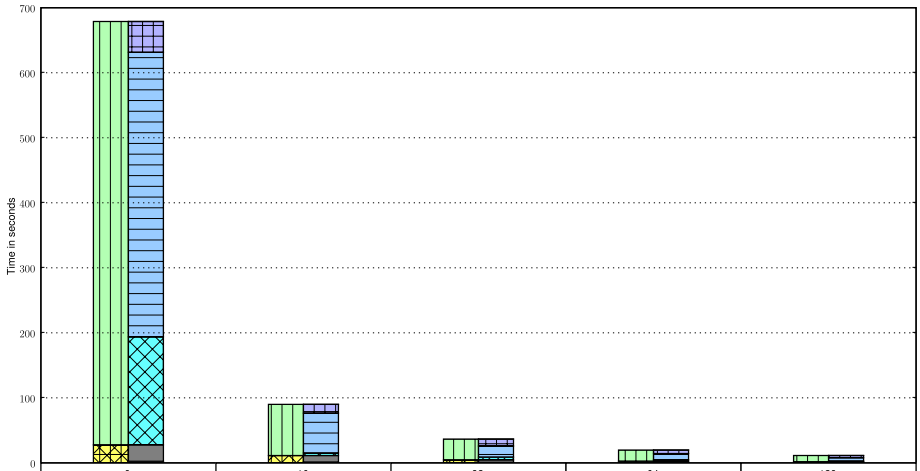
	1	4	8	16	32	64	128	256	512	1024
Construction	3.2	5.7	4.4	8.0	5.8	4.8	8.0	6.7	6.4	18
Internal Balance	10.1	9.0	10.6	14.4	9.3	10.9	14.7	9.0	6.3	14
Boundary Balance	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Communication for Balance	1.9	3.5	2.5	2.9	4.1	3.1	3.9	5.0	5.0	19

Figure 4.15: 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)` communication (including related pre-processing and post-processing) during balance refinement `(Algorithm 11)`, `(2)` balancing across intra and inter processor boundaries `(Algorithm 9)`, `(3)` balancing the blocks `(Algorithm 7)` and `(4)` construction from points `(Algorithm 1)`. 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.



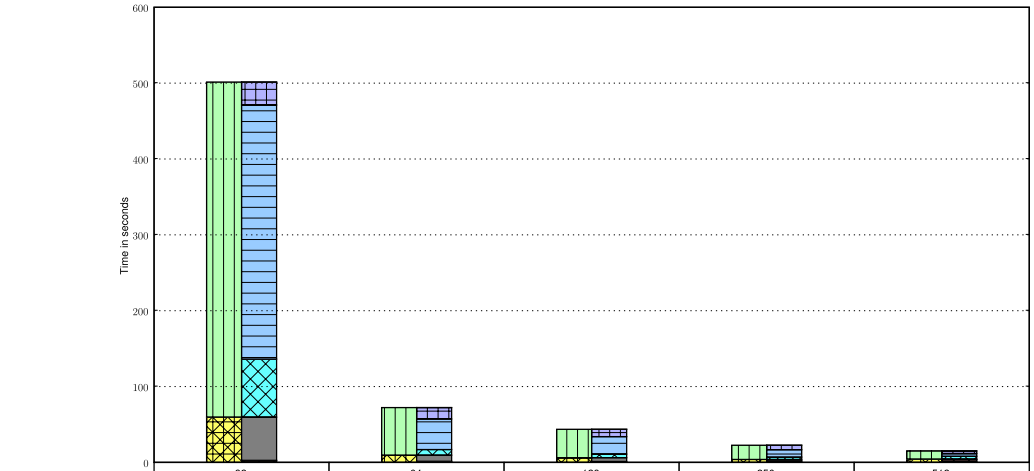
	1	2	4	8	16
Construction	2.3	1.3	0.7	0.7	0.4
Sort - Construction	0.1	0.1	0.1	0.0	0.0
BlockPartition	0.1	0.0	0.1	0.1	0.2
Balance	13.3	7.0	4.1	4.1	2.3
Commn - Balance	1.8	0.9	0.5	0.7	0.4
Internal Balance	8.7	4.3	2.2	2.1	1.0
Boundary Balance	2.8	1.8	1.3	1.4	0.9

Figure 4.16: Fixed size scalability for 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 11)` and `(2)` construction `(Algorithm 1)`, 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 9)`, `(2)` balancing the blocks `(Algorithm 7)`, `(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 4.17: Fixed size scalability for 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 11)` and `(2)` construction `(Algorithm 1)`, 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 9)`, `(2)` balancing the blocks `(Algorithm 7)`, `(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 4.18: Fixed size scalability for 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 11)` and `(2)` construction `(Algorithm 1)`, 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 9)`, `(2)` balancing the blocks `(Algorithm 7)`, `(3)` communication (including related pre-processing and post-processing) during balance refinement, `(4)` local processing during construction, `(5) BlockPartition` and `(6) Sample Sort`.

- The experiments for comparing the performance of different algorithms for the local balancing stage demonstrated that the one proposed in this paper has a significantly lower running time than the others.

- We demonstrated scalability up to 1024 processors: we were able to construct and balance octrees with over 1 billion octants in less than a minute.

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 4.2.5). 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.

4.5 Properties of Morton encoding

Property 2 *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 4.1(b) and 4.1(c) 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 4.19. Similarly, in 3-D six different types of Morton ordering are possible.*

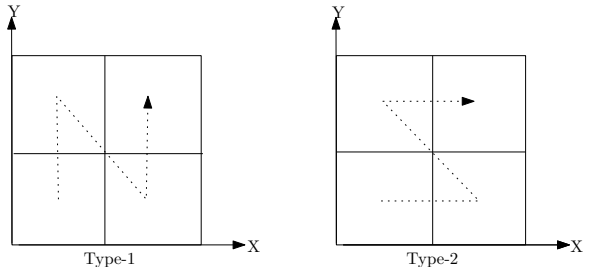


Figure 4.19: Two types of z-ordering in quadtrees.

Property 3 *Given three octants,* $a < b < c$ *and* $c \notin \{\mathcal{D}(b)\}$*:*

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

Property 4 *The Morton id of any node is less than those of its descendants.*

Property 5 *Two distinct octants overlap if and only if one is an ancestor of the other.*

Property 6 *The Morton id of any node and of its first child*[24] *are consecutive. It follows from Property 4 that the first child is also the child with the least Morton id.*

Property 7 *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 8 *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.*

[24] the child that has the same anchor as the parent

Property 9 *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[25] *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 10 *Every octant in the range $(N, \mathcal{DLD}(N)]$ is a descendant of N.*

4.6 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 12 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 4.1.1. The other logical operations can be readily derived from these two operations.

4.7 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 8 octants[26] per processor. The minimum number of octants on any processor, n_{min}, can be expressed in

[25] child with the greatest Morton id

[26] 2^d cells for a d-tree.

Algorithm 12 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. $X_i \leftarrow (A_i \oplus B_i), \quad i \in \{x, y, z\}$
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**

terms of N and the imbalance factor[27], 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 $\mathcal{O}(p)$. Specifically, the total number of blocks produced by the coarsening algorithm, N_{blocks}, satisfies:

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

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

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

4.8 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 2 *well-separated* [28] 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 4.3.1).

[28] 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.

Chapter 5

Inverse Problem

Chapter 6

Results

Bibliography

[1] K. Abd-Elmonieum, S. Sampath, N. Osman, and J.L. Prince. Tool for automatic real-time regional cardiac function analysis using harp. In *SPIE's Medical Imaging*, San Diego, CA, Feb 2003. 14

[2] R. E. Alcouffe, Achi Brandt, J. E. Dendy, Jr., and J. W. Painter. The multigrid method for the diffusion equation with strongly discontinuous coefficients. *SIAM J. Sci. Statist. Comput.*, 2(4):430–454, 1981. 45

[3] D.C. Alexander, C. Pierpaoli, P.J. Basser, and J.C. Gee. Spatial transformations of diffusion tensor magnetic resonance images. *Medical Imaging, IEEE Transactions on*, 20:1131–1139, 2001. 49

[4] American Heart Association. Heart disease and stroke statistics. *American Heart Association, Dallas, Texas*, 2004. 1, 2

[5] D. Ayala, P. Brunet, R. Juan, and I. Navazo. Object representation by means of non-minimal division quadtrees and octrees. *ACM Trans. Graph.*, 4(1):41–59, 1985. 55

[6] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc. 92

[7] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, and Barry F. Smith. PETSc home page. http://www.mcs.anl.gov/petsc, 2001. 45

[8] R. Becker and M. Braack. Multigrid techniques for finite elements on locally refined meshes. *Numerical Linear Algebra with applications*, 7:363–379, 2000. 54

[9] M. Beers and R. Berkow. *The Merck Manual of Diagnosis and Therapy, 17th ed.* Whitehouse Station, NJ, Merck Research Laboratories, 1999. 2

[10] Marshall W. Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry and Applications*, 9(6):517–532, 1999. 55, 56, 73, 76, 81

[11] 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. 55

[12] D.A. Bluemke, E.A. Krupinski, T. Ovitt, K. Gear, E. Unger, L. Axel, L.M. Boxt, G. Casolo, V.A. Ferrari, B. Funaki, et al. MR Imaging of Arrhythmogenic Right Ventricular Cardiomyopathy: Morphologic Findings and Interobserver Reliability. *Cardiology*, 99(3):153–162, 2003. iv, 3, 4, 5

[13] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977. 45

[14] Pere Brunet and Isabel Navazo. Solid representation and operation using extended octrees. *ACM Trans. Graph.*, 9(2):170–197, 1990. 55

[15] P. Cachier. How to trade off between regularization and image similarity in non-rigid registration? In W.J. Niessen and M.A. 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. 9

[16] 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. 9

[17] Paul M. Campbell, Karen D. Devine, Joseph E. Flaherty, Luis G. Gervasio, and James D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003. 60

[18] Yasheng Chen, Yu-Ping 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. 7, 14

[19] T. Corman, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990. 73

[20] F.H. Epstein and W.D. 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. 8

[21] Makela et al. A review of cardiac image registration methods. *IEEE Transactions on Medical Imaging*, 21(9):1011–1021, 2002. 10, 16

[22] V.A. Ferrari and C.H. Scott. Arrhythmogenic Right Ventricular Cardiomyopathy. *Journal of Cardiovascular Electrophysiology*, 14(5):483–484, 2003. 4

[23] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974. 54

[24] L.A. Freitag and R.M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *ACM/IEEE 1999 Conference on Supercomputing*, 13-18 Nov. 1999. 55

[25] Henry Fuchs, Gregory D. Abram, and Eric D. Grant. Near real-time shaded display of rigid objects. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 65–72, New York, NY, USA, 1983. ACM Press. 54

[26] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, New York, NY, USA, 1980. ACM Press. 54

[27] E. Gladilin. *Real-time Simulation of Physically Realistic Global Deformations*. PhD thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin, 2003. 35

[28] G.H. Golub and P.J. Basser. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1983. 49

[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. 57, 85, 92, 95

[30] Henry Gray. *Anatomy of the Human Body*. Lea & Febiger, Philadelphia, 1918. x, 34

[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. 55

[32] Leslie Greengard and William 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. 54, 55

[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. 54, 55

[34] Elizabeth Guest, Elizabeth Berry, Richard A. Baldock, Marta Fidrich, and Mike A. Smith. Robust point correspondence applied to two-and three-dimensional image registration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(2):165–179, 2001. 20, 23

[35] Bhanu Hariharan, Srinivas Aluru, and Balasubramaniam 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. 54, 55, 56, 63

[36] Brian Von Herzen and Alan 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. 61

[37] T.J.R Hughes. *The Finte Element Method. Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987. 35

[38] Garot J., Bluemke D.A., Osman 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. 7, 14

[39] 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*, pages B1221+, December 2002. 54, 55, 61

[40] M.J. Ledesma-Carbayo, J. Kybic, M. Desco, A. Santos, and M. Unser. Cardiac motion analysis from ultrasound sequences using non-rigid registration. In W.J. Niessen and M.A. Viergever, editors, *MICCAI*, volume 2208 of *Lecture Notes in Computer Science*, pages 889–896, Utrecht, The Netherlands, October 14-17, 2001. Springer. 8, 9, 15

[41] 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. 5, 12

[42] Guttman M.A., Prince J.L., and McVeigh E.R. Tag and contour detection in tagged mr images of the left-ventricle. *IEEE Transactions on Medical Imaging*, 13:74–88, 1994. 7, 14

[43] J. Mackerle. Finite element modelling and simulations in cardiovascular mechanics and cardiology: A bibliography 1993–2004. *Computer Methods in Biomechanics and Biomedical Engineering*, 8(2):59–81, 2005. 33

[44] Anant Madabhushi, Michael D. Feldman, Dimitris N. Metaxas, John Tomaszeweski, and Deborah Chute. Novel stochastic combination of 3d texture features for automated segmentation of prostatic adenocarcinoma from high resolution mri. In *MICCAI*, 2003. 12

[45] 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. 56

[46] 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. 5, 12

[47] 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. 9

[48] J.C. McEachen, II, A. Nehorai, and J.S. Duncan. Multiframe temporal estimation of cardiac nonrigid motion. *Image Processing, IEEE Transactions on*, 9(4):651–665, Apr. 2000. 8, 9, 15

[49] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19:129–147, 1982. 54, 55

[50] Doug Moore. The cost of balancing generalized quadtrees. In *Symposium on Solid Modeling and Applications*, pages 305–312, 1995. 56, 61

[51] N.F. Osman, W.S. Kerwin, E.R. McVeigh, and J.L. Prince. Cardiac motion tracking using CINE harmonic phase(HARP) magnetic resonance imaging. *Magnetic Resonance in Medicine*, 42(6):1048–1060, 1999. 7, 8

[52] X. Papademetris, A. J. Sinusas, D. P. Dione, and J. S. Duncan. Estimation of 3D left ventricular deformation from echocardiography. *Medical Image Analysis*, 5:17–28, 2001. 8, 15

[53] PM Pattynama, HJ Lamb, EA van der Velde, EE van der Wall, and A. de Roos. Left ventricular measurements with cine and spin-echo MR imaging: a study of repro-

ducibility with variance component analysis. *Radiology*, 187(1):261–8, 1993. iv, 3

[54] M. Paul, E. Schulze-Bahr, G. Breithardt, and T. Wichter. Genetics of arrhythmogenic right ventricular cardiomyopathy-status quo and future perspectives. *Zeitschrift für Kardiologie*, 92(2):128–136, 2003. 4

[55] D. Perperidis, R. Mohiaddin, and D. Rueckert. Spatio-temporal free-form registration of cardiac mr image sequences. In *MICCAI*, pages 911–919, 2004. 8, 9, 15

[56] C. Pierpaoli, P. Jezzard, P.J. Basser, A. Barnett, and G.D. Chiro. Diffusion tensor mr imaging of the human brain. *Radiology*, 201(3):637–648, 1996. 46

[57] 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. 54, 55

[58] 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. 5, 12

[59] Frank B. Sachse. *Computational Cardiology: Modeling of Anatomy, Electrophysiology, and Mechanics*, volume 2966 of *Lecture Notes in Computer Science*. Springer, 2004. 32, 33, 51

[60] S.G. Sawada, Segar D.S., and Ryan T. Echocardiographic detection of coronary artery disease during dobutamine infusion. *Circulation*, 83:1605–1614, 1991. 12

[61] R. Schneiders. An algorithm for the generation of hexahedral element meshes based on an octree technique, 1997. 55

[62] R. Schneiders, R. Schindler, and F. Weiler. Octree-based generation of hexahedral element meshes, 1996. 55

[63] R.A. Schumaker, P. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. *U.S. Air Force Human Resources Laboratory, Technical Report*, 69(14), 1969. 54

[64] D.F Scollan, A. Holmes, R.L. 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. 46, 47, 51

[65] M. Sermesant, K. Rhode, G.I. Sanchez-Ortiz, O. Camara, R. Andriantsimiavona, S. Hegde, D. Rueckert, P. Lambiase, C. Bucknall, E. Rosenthal, H. Delingette, D.L.G. 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. 51

[66] Dinggang Shen and C. Davatzikos. Hammer: hierarchical attribute matching mechanism for elastic registration. *Medical Imaging, IEEE Transactions on*, 21(11):1421–1439, Nov. 2002. 5, 12, 16, 47, 48

[67] M.S. Shephard and M.K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 26:709–749, 1991. 55

[68] Jonathan Richard 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. 55

[69] P. Shi, A.J. Sinusas, R. T. Constable, and J.S. 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. 7, 8, 9, 14, 15

[70] S.C. Smart, T. Knickelbine, F. Malik, and K.B. 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. 6

[71] Arun K. Somani and Allen M. Sansano. Minimizing overhead in parallel algorithms through overlapping communication/computation. Technical report, Institute for Computer Applications in Science and Engineering (ICASE), 1997. 85

[72] 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. 8, 9, 15

[73] K.C. Strasters and J.J. Gerbrands. 3-dimensional image segmentation using a split, merge and group-approach. *Pattern Recognition Letters*, 12(5):307–325, May 1991. 55

[74] Daniel D. Streeter Jr. and William 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. 51

[75] H. TANDRI, H. CALKINS, K. NASIR, C. BOMMA, E. CASTILLO, J. RUTBERG, C. TICHNELL, J. LIMA, and D.A. BLUEMKE. Magnetic Resonance Imaging Findings in Patients Meeting Task Force Criteria for Arrhythmogenic Right Ventricular Dysplasia. *magnetic resonance imaging*, 2003. 5

[76] C.H. Teh and R.T. Chin. On image analysis by the methods of moments. *IEEE Trans. PAMI*, 10:496–513, 1988. 5, 12

[77] Shang-Hua Teng. Provably good partitioning and load balancing algorithms for parallel adaptive n-body simulation. *SIAM Journal on Scientific Computing*, 19, 1998. 54, 55

[78] G. Thiene, C. Basso, F. Calabrese, A. Angelini, and M. Valente. Pathology and Pathogenesis of Arrhythmogenic Right Ventricular Cardiomyopathy. *Herz*, 25(3):210–215, 2000. 4

[79] G. Thiene, A. Nava, D. Corrado, L. Rossi, and N. Pennelli. Right ventricular cardiomyopathy and sudden death in young people. *New Engl J Med*, 318:129–133, 1988. 4

[80] Francois Meyer 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. 7, 14

[81] H. Tropf and H. Herzog. Multidimensional range search in dynamically balanced trees. *Angewandte Informatik*, 2:71–77, 1981. 55

[82] W.-Y.I. Tseng, T.G. Reese, R.M Weisskoff, and V.J. Wedeen. Cardiac diffusion tensor mri in vivo without strain correction. *Magnetic Resonance in Medicine*, 42(2):393–403, 1999. 47

[83] Tiankai Tu and David R. O'Hallaron. Balance refinement of massive linear octree datasets. *CMU Technical Report*, CMU-CS-04(129), 2004. 73, 76, 80, 81

[84] Tiankai Tu and David 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. 55

[85] Tiankai Tu, David R. O'Hallaron, and Omar 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. 55, 56, 57, 61, 63, 73, 74, 81, 85, 93

[86] Joseph C. Walker, Julius M. Guccione, Yi Jiang, Peng Zhang, Arthur W. Wallace, Edward W. Hsu, and Mark B. Ratcliffe. Helical myofiber orientation after myocardial infarction and left ventricular surgical restoration in sheep. *J Thorac Cardiovasc Surg*, 129(2):382–390, 2005. 51

[87] Yu-Ping Wang, Y. Chen, and A. A. Amini. Efficient lv motion estimation using subspace approximation techniques. *Medical Image Analysis*, 20(6):499–513, Jun. 2001. 8, 15

[88] 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. 54, 55

[89] Michael S. Warren and John K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of Supercomputing '93*, 31 March 1993. 55, 56, 63

[90] World Heart Foundation website. http://www.worldheart.org/call-to-action.php. 2

[91] Dongrong Xu, Susumu Mori, Dinggang Shen, Peter C.M. van Zijl, and Christos Davatzikos. Spatial normalization of diffusion tensor fields. *Magnetic Resonance in Medicine*, 50:175–182, 2003. 49, 50

[92] Zhong Xue, Dinggang Shen, and C. Davatzikos. Determining correspondence in 3-d mr brain images using attribute vectors as morphological signatures of voxels. *Medical Imaging, IEEE Transactions on*, 23(10):1276–1291, Oct. 2004. 16, 20

[93] Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J. Comput. Phys.*, 196(2):591–626, 2004. 54, 55

[94] Lexing Ying, George Biros, Denis Zorin, and Harper 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. 54, 55, 56, 63

[95] EA Zerhouni, DM Parish, WJ Rogers, A Yang, and EP Shapiro. Human heart: tagging with mr imaging–a method for noninvasive assessment of myocardial motion. *Radiology*, 169(1):59–63, 1988. 6, 7, 14

[96] Barbara Zitová and Jan Flusser. Image registration methods: a survey. *Image Vision Comput.*, 21(11):977–1000, 2003. 10, 16, 48