2011

On the Real-Time Performance, Robustness and Accuracy of Medical Image Non-Rigid Registration

Yixun Liu

College of William & Mary - Arts & Sciences

Follow this and additional works at: https://scholarworks.wm.edu/etd

Part of the Biomedical Engineering and Bioengineering Commons, Computer Sciences Commons, and the Investigative Techniques Commons

Recommended Citation


https://dx.doi.org/doi:10.21220/s2-06t5-q122

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.
On the Real-Time Performance, Robustness and Accuracy of Medical Image Non-Rigid Registration

Yixun Liu
Kaifeng, Henan, China

Master of Biomedical Engineering, Fudan University, 2003
Bachelor of Computer Science, Jiangsu University, 1996

A Dissertation presented to the Graduate Faculty of the College of William and Mary in Candidacy for the Degree of Doctor of Philosophy

Department of Computer Science

The College of William and Mary
August, 2011
Three critical issues about medical image non-rigid registration are performance, robustness and accuracy. A registration method, which is capable of responding timely with an accurate alignment, robust against the variation of the image intensity and the missing data, is desirable for its clinical use. This work addresses all three of these issues.

Unacceptable execution time of Non-rigid registration (NRR) often presents a major obstacle to its routine clinical use. We present a hybrid data partitioning method to parallelize a NRR method on a cooperative architecture, which enables us to get closer to the goal: accelerating using architecture rather than designing a parallel algorithm from scratch. To further accelerate the performance for the GPU part, a GPU optimization tool is provided to automatically optimize GPU execution configuration.

Missing data and variation of the intensity are two severe challenges for the robustness of the registration method. A novel point-based NRR method is presented to resolve mapping function (deformation field) with the point correspondence missing. The novelty of this method lies in incorporating a finite element biomechanical model into an Expectation and Maximization (EM) framework to resolve the correspondence and mapping function simultaneously. This method is extended to deal with the deformation induced by tumor resection, which imposes another challenge, i.e. incomplete intra-operative MRI. The registration is formulated as a three variable (Correspondence, Deformation Field, and Resection Region) functional minimization problem and resolved by a Nested Expectation and Maximization framework. The experimental results show the effectiveness of this method in correcting the deformation in the vicinity of the tumor. To deal with the variation of the intensity, two different methods are developed depending on the specific application.

For the mono-modality registration on delayed enhanced cardiac MRI and cine MRI, a hybrid registration method is designed by unifying both intensity- and feature point-based metrics into one cost function. The experiment on the moving propagation of suspicious myocardial infarction shows effectiveness of this hybrid method. For the multi-modality registration on MRI and CT, a Mutual Information (MI)-based NRR is developed by modeling the underlying deformation as a Free-Form Deformation (FFD). MI is sensitive to the variation of the intensity due to equidistant bins. We overcome this disadvantage by designing a Top-to-Down K-means clustering method to naturally group similar intensities into one bin. The experiment shows this method can increase the accuracy of the MI-based registration.

In image registration, a finite element biomechanical model is usually employed to simulate the underlying movement of the soft tissue. We develop a multi-tissue mesh generation method to build a heterogeneous biomechanical model to realistically simulate the underlying movement of the brain. We focus on the following four critical mesh properties: tissue-dependent resolution, fidelity to tissue boundaries, smoothness of mesh surfaces, and element quality. Each mesh property can be controlled on a tissue level. The experiments on comparing the homogeneous model with the heterogeneous model demonstrate the effectiveness of the heterogeneous model in improving the registration accuracy.
# Table of Contents

Acknowledgments

List of Tables

List of Figures

1 Real-time Non-rigid Registration Using GPU and Multicore 2
  1.1 Introduction ..................................................... 2
  1.2 Related Work .................................................. 6
    1.2.1 CPU Related Work ........................................ 6
    1.2.2 GPU Related Work ........................................ 9
  1.3 Non-Rigid Registration Approach ................................ 11
  1.4 Parallel Implementation ...................................... 13
    1.4.1 GPU Implementation of 3D Block Matching ................ 14
      1.4.1.1 CUDA Programming Model .............................. 15
      1.4.1.2 Mapping to the CUDA Programming Model ............ 17
      1.4.1.3 Memory Issue .......................................... 18
      1.4.1.4 Optimization of GPU Program ......................... 20
    1.4.2 Multicore Implementation of Incremental Finite Element Solver 22
      1.4.2.1 Parallel Partitioning ................................. 23
      Element partitioning .......................................... 24
      Vertex partitioning .......................................... 26
      1.4.2.2 Renumbering .......................................... 27
      1.4.2.3 Assembling ........................................... 28
  1.5 Results .......................................................... 29
    1.5.1 GPU VS. Cluster on Block Matching ........................ 29
    1.5.2 GPU Optimization of Block Matching ...................... 30
    1.5.3 Incremental Solver ........................................ 32
  1.6 Discussion ........................................................ 34
  1.7 Conclusion ...................................................... 35

2 GPU Program Optimizations 37
  2.1 Introduction ..................................................... 37
  2.2 Background on GPU Architecture and CUDA ..................... 40
  2.3 Challenges in the Optimization of GPU Programs ................ 41
  2.4 Adaptive Optimization Framework ............................... 44
3 A Robust Point-based Registration Method for Brain Shift

3.1 Introduction ........................................... 54
3.2 Method .................................................. 58
3.2.1 Energy Function ....................................... 58
3.2.2 Expectation and Maximization ......................... 61
3.2.3 Outlier Rejection ..................................... 62
3.3 Experiments on Brain Shift ............................. 63
3.3.1 Register the LRS Space with the Image Space .... 65
3.3.2 Results ............................................... 67
3.4 Conclusion .............................................. 69

4 A Robust NEM Registration Method for Tumor Resection

4.1 Introduction ............................................. 71
4.2 Method .................................................. 73
4.2.1 3D Multi-tissue Tetrahedral Mesh Generation .... 76
4.2.1.1 Coarse Multi-tissue Mesh ......................... 76
4.2.1.2 Deform the Mesh Surface To the Tissue Boundary .. 77
4.2.2 Nested Expectation and Maximization Non-rigid Registration (NEMNRR) .......................... 78
4.2.2.1 Cost Function ..................................... 78
4.2.2.2 Nested Expectation and Maximization Solver .... 81
Inner EM .................................................. 82
Outer EM .................................................. 83
4.3 Results .................................................. 85
4.3.1 Experiments on Synthetic Data ....................... 86
4.3.2 Experiments on Clinical MRI ......................... 88
4.3.2.1 Low-Field Intra-operative MRI ................... 88
4.3.2.2 High-Filed Intra-operative MRI .................. 90
4.3.2.3 Heterogeneous Model VS. Homogeneous Model .. 95
4.4 Conclusion .............................................. 98

5 A Robust Hybrid Registration Method for Cardiac Motion Tracking

5.1 Introduction ............................................. 100
5.2 Method .................................................. 103
5.2.1 Select the Reference Frame in the Cine ............ 104
5.2.2 Compute Deformation Fields within Cine Series ..... 104
5.2.3 Register DE-MRI to the Reference Cine ............ 106
I am grateful to my adviser Nikos Chrisochoides for everything he has done. I also thank Andriy Kot for his providing the dissertation template. Miss Sydney Folsom and Caitlin Reed spent quite a lot of time on proofreading. I really appreciate them.

This work is supported in part by NSF grants: CCF-1139864, CCF-1136538, and CSI-1136536 by the John Simon Guggenheim Foundation and the Richard T. Cheng Endowment.
List of Tables

1.1 Optimal GPU execution configuration for <ImageBlock, SearchWindow>. Row: image block dimensions. Column: search window dimensions. .......................................................... 32
1.2 Performance comparison between sequential and parallel solver. .......... 32
1.3 Performance evaluation for parallel solver on large mesh. Mesh loading and element graphic creation are performed using one core. ............ 32

2.1 Benchmarks. ........................................................................ 52
3.1 LRS Specification. ............................................................... 65

4.1 Relative accuracy improvement [BMNRR, NEMNRR] (%) for superficial points. Negative value means the error is not reduced but increased. 95
4.2 Relative accuracy improvement [BMNRR, NEMNRR] (%) for deep and rigid points. Negative value means the error is not reduced but increased. 96
4.3 Quantitative comparison between homogeneous model and heterogeneous model for five slices. [homogeneous, heterogeneous]........ 98

5.1 Quantitative measures of DE-MRI to cine registration. .................. 109
5.2 Area/Thickness change %. ACI: Area Change of Infarction zone. ACN: Area Change of Normal myocardium. TCI: Thickness Change of Infarction zone. TCN: Thickness Change of Normal myocardium. .......... 112

6.1 Accuracy evaluation (mm) on 7 detectable feature points of CT: 1) anterior horn of right lateral ventricle (AHRLV), 2) pons (PONS), 3) anterior horn of left lateral ventricle (AHLLV), 4) posterior horn of right lateral ventricle (PHRLV), 5) posterior horn of left lateral ventricle (PHLLV), 6) septum pellucidum (SP), and 7) splenium of corpus callosum (SCC). .... 124

7.1 Operation case table for tetrahedron T with label L. ..................... 132
7.2 Quantitative evaluation for the multi-tissue mesh on the brain atlas. The atlas is regularized as spacing: 1mm x 1mm x 1mm, size: 240 x 240 x 259. The parameters are: subdivision threshold=0.85, $\lambda = 1.0$. .......... 147
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Block Matching. a. Floating Image b. Reference Image.</td>
<td>12</td>
</tr>
<tr>
<td>1.2</td>
<td>Block Matching result. The arrow points to the direction of the displacement and the color scale encodes the norm of the displacement, in millimeters. The metric is NCC and for clarity, only 1% of the displacement field is shown.</td>
<td>13</td>
</tr>
<tr>
<td>1.3</td>
<td>Parallel NRR framework.</td>
<td>14</td>
</tr>
<tr>
<td>1.4</td>
<td>Data parallel framework. Left: Regular data partitioning of 3D image on GPU. Right: Irregular data partitioning of 3D tetrahedron mesh on multicore.</td>
<td>15</td>
</tr>
<tr>
<td>1.5</td>
<td>CUDA programming model.</td>
<td>17</td>
</tr>
<tr>
<td>1.6</td>
<td>Mapping from sequential BM to GPU programming model. Left: Sequential BM. Right: GPU BM.</td>
<td>19</td>
</tr>
<tr>
<td>1.7</td>
<td>Desirable Matrix.</td>
<td>24</td>
</tr>
<tr>
<td>1.8</td>
<td>Element partitioning. ParMETIS (middle) takes initial partitioning (left top) and element graph (left bottom) as inputs and outputs the mapping from elements to cores (right top). Migrate elements to appropriate cores according to this mapping to yield final partitioning (right bottom).</td>
<td>25</td>
</tr>
<tr>
<td>1.9</td>
<td>Vertices partitioning. A load imbalance matrix before vertex partitioning (a) and a load balance matrix after vertex partitioning (b).</td>
<td>26</td>
</tr>
<tr>
<td>1.10</td>
<td>4-way partitioned mesh and registration points. Total Tetras=7272, Vertices=1607. Tetras in red sub-mesh=1818, Vertices in red sub-mesh=517. Tetras in blue sub-mesh=1818, Vertices in blue sub-mesh=526. Tetras in green sub-mesh=1818, Vertices in green sub-mesh=497. Tetras in wireframe sub-mesh=1818, Vertices in wireframe sub-mesh=496.</td>
<td>27</td>
</tr>
<tr>
<td>1.11</td>
<td>Renumbering and mapping table.</td>
<td>28</td>
</tr>
<tr>
<td>1.12</td>
<td>Performance evaluation using six existing retrospective data in BWH. Image block: 4 x 4 x 4, Search window: 5 x 5 x 5, Optimal Thread block (obtained from Table 1.1): 4 x 4 x 4. The size of the image block and the search window are represented by the half of the length.</td>
<td>30</td>
</tr>
<tr>
<td>1.13</td>
<td>Thread block size vs. Kernel time, MRI: 316 x 316 x 188, spacing: 0.94 x 0.94 x 1.50, intraMRI: 316 x 316 x 188, spacing: 0.94 x 0.94 x 1.50, Selected points: 5000, Search Window size: 10 x 10 x 10, Image block size: 4 x 4 x 4.</td>
<td>31</td>
</tr>
<tr>
<td>1.14</td>
<td>Optimal block size vs. worst block size. Light blue bar is the kernel runtime with optimal block size and dark blue bar is the kernel runtime with worst block size. The speedup ranges from 4.0 (8 x 8 x 8) to 7.2 (7 x 7 x 7).</td>
<td>31</td>
</tr>
</tbody>
</table>
1.15 Registration results. Top left: preoperative MRI. Top middle: intraoperative MRI. Top right: deformed preoperative MRI. Bottom left: deformed preoperative MRI (red) superimposed on intra-operative MRI. Bottom middle: deformed preoperative MRI (1st iteration) superimposed on intra-operative MRI. Bottom right: deformed preoperative MRI (10th iteration) superimposed on intra-operative MRI.

1.16 Precision evaluation for different mesh. Middle figure is the closeup of iterations from 15 to 20.

2.1 G-ADAPT: An adaptive optimization framework for GPU programs.

3.1 TPS and FFD.
3.2 LRS.
3.3 Space transformation.
3.4 Point based NRR.
3.5 Qualitative evaluation.
3.6 Quantitative evaluation.

4.1 The context of the Nested Expectation and Maximization Non-rigid Registration (NEMNRR). The red boxes represent the new technologies we present in this chapter and the gray boxes represent the existing technologies.

4.2 Coarse multi-tissue mesh generation. (a) L1 and L2 are tissue labels, the dash line is the real boundary and the blue line is the submesh interface. (b) Distribute labels. (c) Subdivide if not satisfy the resolution criterion. (d) Distribute labels again.

4.3 Element outliers and point outliers.

4.4 Nested Expectation and Maximization framework.

4.5 Illustration of Nested Expectation and Maximization. Row: inner EM, Column: outer EM.

4.6 Results from the synthetic data.

4.7 The 3.0 T magnet system (Signa SP, Siemens Medical Systems) of the Neurological Department of Huashan Hospital, FDU, Shanghai, China.

4.8 Results of NEMNRR and the comparison between NEMNRR and BMNRR.

4.9 BOLD and high-field intra-operative MRI.

4.10 Resected elements and mesh quality after deformation.

4.11 Deformation field. The color denotes deformation magnitude and the arrow points to deformation direction. Part of the brain, not including ventricles, is removed to display the deformation field of ventricles.

4.12 NEMNRR results using high-field intra-operatively MRI.

4.13 BOLD deformation. The first two rows correspond to two different slices. Left image in the first row: before correction. Right image in the first row: corrected BOLD. The last row: superimpose uncorrected BOLD on the corrected BOLD to show the deformation.
4.14 Seven anatomical feature points. Superficial points: Three right lateral fissure points (RLF1-3), Deep points: left and right anterior horns of the lateral ventricle (LAH, RAH), and two Rigid points: basilar sulcus (BS) and anterior commissure (AC). ........................................... 95

4.15 Quantitative evaluation of the accuracy using four cases including two low-field MRI (case1 and case2) and two high-field MRI (case3 and case4). 96

4.16 Single tissue and multi-tissue meshes. ........................................ 97

4.17 Model comparison using ventricle. (a) superimpose the ventricle extracted from the pre-operative MRI, corrected by the homogeneous model, on the intra-operative MRI. (b) superimpose the ventricle extracted from the pre-operative MRI, corrected by the heterogeneous model, on the intra-operative MRI. ........................................... 97

5.1 \(c_k\): the reference frame. \(d_k\): deformation from \(c_k\) to the DE-MRI. \(d_l, l = 1 \ldots n, l \neq k\): deformation from \(c_l\) to \(c_k\). Solid line arrow points to the deformation direction and dash line arrow points to the propagation direction. \(p_i\) is the position of the pixel with index \(i\). ......................... 103

5.2 A DE-MRI image is registered to the selected cine frame using the variational registration. The warped DE-MRI shows unrealistic deformation due to enhanced infarction bearing no contrast in cine. ......................... 106

5.3 Three propagation schemes. ................................................. 108

5.4 Comparison between intensity-based method and the hybrid method. (a) cine image. (b) DE-MRI. (c) aligned DE-MRI by CCPD. (d) before registration. (e) CC registration. (f) CCPD registration. ......................... 109

5.5 Area and thickness of both infarction and healthy myocardium over one cardiac cycle. ................................................. 110

5.6 The first row is the cine images; the second row is propagated DE-MRI, and the third row is propagated contour superimposed on the cine. The propagated contours are zoomed in at last row. The image with a green box in the first row corresponds with \(C_k\) in Fig.5.1, and the image with a green box in the second row corresponds with \(E^*_k\). Due to space limit, we only present the results at three different phases. ......................... 111

6.1 K-means clustering. (a) is original MRI and (b) is K-means clustered (labeled) image, in which midbrain and white matter with label 180, gray matter and skin with label 162 fall into the same cluster. (c) is CT, whose window position and width are carefully adjusted, and (d) is clustered CT.117

6.2 The influence of the bin size to the registration. (a) shows the misalignment of blue region with green region leads to additional (blue, background) combination, and therefore a higher joint entropy. (b) shows some details of blue region can be distinguished using a small bin size. The misalignment leads to additional (yellow, background), (red, background), and (green, background) combinations. However, a small bin size does not change the registration result. (c) shows a large blue region is produced by a large bin size. The registration result is not unique. . . 118
6.3 The rigidly registered MRI (a) is non-rigidly registered with CT (b). The resulting MRI (c) is merged with CT and two merged slices are shown as (d) and (e). ................................................................. 122

6.4 The comparison of the results. The row is the index of the slice and the column is the registration method. The bin number we use in the Image-to-Image method is 256, which yields the best result among 32, 64, 128, 256. The bin number in the Cluster-to-Image method is 31 (the number of clusters) for clustered CT, and \(2K = 62\) for MRI. For the Cluster-to-Cluster method, the bin numbers are 31 for clustered CT, and 13 for clustered MRI respectively. Some detectable boundaries of soft tissues of CT, such as the cerebellar hemisphere, midbrain, and ventricles, are extracted, highlighted, and overlapped on registered MRI. The green arrows point to the boundaries exhibiting significant improvement of the accuracy using the Cluster-to-Image method. ............................ 123

7.1 Multi-tissue mesher framework ............................................. 128

7.2 BCC lattice and red-green subdivision (These two figures come from [68]). 129

7.3 Coarse multi-tissue mesh generation. (a) L1 and L2 are tissue labels, the dash line is the real boundary and the blue line is the submesh interface. (b) Redistribute labels according to operation table 7.1. (c) Subdivide if not satisfy the resolution criterion defined in (e). (d) Redistribute labels again. (e) Resolution criterion: 0.85 is the subdivision threshold, an experiment value evaluated on MRI, visible human and brain atlas. Points represent voxels and colors represent different tissues. \(S_1\) is the voxel set within the blue submesh (blue dash lines) and \(S_2\) is the voxel set within the blue tissue (blue curves). ............................. 131

7.4 Three special cases. The circle represents the tissue region and the polygon represents the submesh. For simplicity, the voxels are not shown. All three cases show a big discrepancy between the tissue boundary and the submesh boundary. However, for case (a), because the tissue is totally covered by the submesh, \(\frac{|S_1 \cap S_2|}{|S_1|}\) has the highest value 1.0. For case (b), because the submesh is totally covered by the tissue region, \(\frac{|S_1 \cap S_2|}{|S_1|}\) has the highest value 1.0. For case (c), \(\frac{|S_1|}{|S_2|}\) can be equal to be 1.0, if the submesh and tissue region have the same number of voxels. ............................ 133

7.5 Point sets. The source point set (b) includes all the surface nodes of the coarse mesh (a), and the target point set (d) is the edge points in the multi-label image (c). ........................... 135

7.6 The calculation of \(d_s\) of node \(s\). \(p_1\) and \(p_2\) are two neighboring nodes of \(s\). \(t_1\), \(t_2\) and \(q\) are the closets points corresponding to \(p_1\), \(p_2\), and \(s\), respectively. Their average position is \(s'\). Project \(s' - s\) on unit normal \(n\) of node \(s\) to produce \(d_s\). ............................ 138

7.7 Multi-tissue mesh generation for MRI data. (a) is the multi-label image. The coarse multi-tissue mesh (b) is generated with subdivision threshold 0.85. (c) is the deformed multi-tissue mesh. The numbers of source points and target points are 4497 and 31241, respectively. ......................... 140
7.8 (a) is the closeup of the inner ventricle. (b) is the wireframe view of the two submeshes and (c) is the extracted ventricle.  
7.9 (a) is the brain with a ventricle hole. (b) is the extracted ventricle surface. (c) is the wireframe view of the hole. The front surfaces of the brain are culled to show the hole.  
7.10 (a) shows the conformity of the interface. The part in the rectangle is enlarged in (b).  
7.11 Multi-tissue mesh generation for visible human data. (a) is the multi-label image. The coarse multi-tissue mesh (b) is generated with subdivision threshold 0.85. (c) is the deformed multi-tissue mesh. The numbers of source points and target points are 5828 and 26060, respectively.  
7.12 (a) is the closeup of the inner. (b) is the wireframe view of the three submeshes, and (c) is the extracted two nerves.  
7.13 Tissue-aware quality control. The two values in the bracket are minimum and maximum dihedral angles.  
7.14 Multi-tissue mesh for brain atlas. Five tissues along with the rest of the brain (a) are discretized. 43: right caudata nucleus (RCN), 53: left caudata nucleus (LCN), 98: right anterior horn of lateral ventricle (RAHLV), 99: left anterior horn of lateral ventricle (LAHLV), 140: corpus callosum (CC). The numbers of source points and target points are 6225 and 39136, respectively.  
7.15 Conformity of interfaces.  
7.16 The evaluation of fidelity, tissue dependent resolution, and quality on the brain atlas.
On the Real-Time Performance, Robustness and Accuracy of Medical Image Non-Rigid Registration
Chapter 1

Real-time Non-rigid Registration

Using GPU and Multicore

1.1 Introduction

Image registration is the process of aligning images so that corresponding features can be easily related [34]. Given two images, a floating image $F$ and a reference image $R$, image registration methods aim to find a deformation field $\Phi$, which best aligns the deformed floating image with the reference image according to similarity measures.

Image registration has a wide variety of applications in medical fields [36, 59]:

- combine images of the same/different subjects from same/different modalities to obtain rich diagnostic information
- align temporal sequences of images to monitor tumor growth
- align images from multiple subjects in cohort studies
- compensate for brain deformation in Image-guided neurosurgery (IGNS)
Medical image registration falls into two categories: Rigid registration and Non-Rigid Registration (NRR). Rigid registration assumes the anatomical structures of interest do not deform or distort, which simplifies the registration process, but limits its application. NRR techniques, which are characterized by a capacity to estimate transformations that model not only affine parameters (global translation, rotation, scale, and shear) but also local deformations, have broad applications, but are usually more computationally expensive [36]. A host of NRR techniques abound in literatures [5, 6, 82, 89, 91, 108]. Readers are referred to [59] and [36] for an excellent survey.

The long CPU runtime of the existing NRR techniques is a major barrier to their routine clinical use in all time-critical intra-operative applications, which typically allow only few minutes as a maximum response time.

A host of studies employ parallel computing to accelerate NRR on multicore or clusters [17, 37, 86, 96, 107]. Recently, some groups implemented NRR on Graphics Processing Units (GPUs) [50, 69, 90, 104, 111]. However, to date there have no been attempts to accelerate NRR using the cooperative architecture: multicore and GPU, which is widely available in commodity PCs. This cooperative architecture can be easily deployed in the Operating Room (OR) without hindering routine surgery procedures. In addition, GPU’s SIMD programming model and multicore’s SPMD programming model complement each other and provide a powerful and flexible hybrid programming environment. GPU has a massive number of cores, which is effective in performing regular computations, but provides limited support for communication and synchronization [73]. Multicore architectures have limited cores, but are more universal in allowing the implementation of irregular algorithms and providing flexible support for communication
and synchronization. Intel research [81] has recently reported an interesting study about strong and weak sides of multicore vs. GPU. Algorithms with the following characteristics are defined as irregular algorithms:

- Require dynamic data types, such as sparse matrices, linked lists, or trees
- Have high likelihood of contended synchronization
- Have moderate control flow, such as well-structured conditional nests, nested loops, and recursive functions

Multicore architecture (including tera-scale architecture) addresses irregular algorithms efficiently, whereas GPU does not.

GPU programming models are constrained in such a way that the compiler and runtime can reason about the application and extract the parallelism automatically. Examples of this include DirectX, CUDA, and Cg. The programmers need to reformulate their application to fit these constraints if the application involves irregular algorithms. However, this reformulation often requires considerable programmer effort, and can result in significantly less efficient software algorithms. Unlike GPUs, Multicore architectures have the following features, which make CPU more suitable for performing irregular algorithms than GPU [81]:

- Inter-core communication through substantial, coherent cache hierarchies
- Efficient, low latency thread synchronizations across the entire core array
- Narrower effective SIMD width

The GPU programming model is very suitable for performing regular algorithms. A regular algorithm can be easily mapped, without reformulation, to GPU program-
ming and reach desirable speedup as we will show in this chapter. If we separate NRR into a regular part and an irregular part and implement them on GPU and multicore, respectively, we can gain desirable speedup for each part. Moreover, with minor modification to the original algorithm, each part can be easily mapped to its corresponding programming model. Minimizing changes to the original algorithm is very important. Usually, before resorting to parallel computing to accelerate our algorithms, we have a well-designed sequential code with established accuracy, robustness, and performance. As we parallelize it, we attempt to map this sequential code to a parallel architecture to gain speedup instead of developing a parallel algorithm from scratch.

Cooperative architecture provides us with a powerful parallel computing environment. To take full advantage of it, in addition to GPU implementation of regular part, an extended GPU optimization tool based on our previous work is developed to optimize the partitioning. For multicore implementation of incremental Finite Element solver, a parallel data partitioning algorithm on unstructured mesh is developed, which is capable of alleviating the impact of load imbalance under the constraint of minimal communication.

The contributions of this chapter are:

- A GPU based 3D Block Matching algorithm
- A GPU optimization tool
- A parallel incremental Finite Element solver

Based on these parts, a data parallel implementation of NRR on the cooperative architecture is presented, which is characterized by:
• Desirable speedup

• Minimal changes to the existing sequential algorithm

• Higher price/performance ratio

The remainder of this chapter is organized as follows: in Section 1.2, we review related work on parallel NRR. Since there are no reports about parallel implementation of NRR on the cooperative architecture, we will review NRR according to GPU and CPU implementation separately. In Section 1.3, we will briefly describe the principle of the NRR technique used in this chapter. The details on how to parallelize this NRR method on the cooperative architecture are presented in Section 1.4. Experimental results about performance and accuracy are shown in Section 1.5. After discussion about the applicability of the techniques provided in Section 1.6 of this chapter, we conclude this chapter with Section 1.7.

1.2 Related Work

There is a significant body of literature addressing parallel NRR. We separate such related work into two categories: CPU- and GPU-related studies.

1.2.1 CPU Related Work

Warfield et al. [107] parallelized a feature point-based non-rigid registration method by distributing two operations: resampling and comparison across a cluster of symmetric multicore. This method has been successfully performed on intrapatient and interpatient
registrations. The execution time for the cluster implementation is compatible with routine clinical use, usually 5-10 minutes.

Unlike operation distribution, Ino et al. [37] provided a data distributed parallel algorithm for free-form based non-rigid registration [89]. The floating image and reference image are distributed using a block distribution algorithm to achieve higher speedup for the hotspot of the sequential algorithm. The block size for \( N \) cores is determined by selecting the balancing point between memory usage and execution time. This data distribution method can effectively reduce the memory usage per node in a data parallel programming model and is particularly suitable for registering large images.

This free-form based NRR has also been parallelized by Rohlfing et al. [86]. Unlike the work presented by Ino et al. [37], Rohlfing et al. only simply broke the data into equally sized partitions. Our NRR method includes two parts: Block Matching and incremental Finite Element (FE) Solver. For FE solver, we also employ a data partitioning technique. The data we partition are irregular mesh and registration points, which is different from the regular image data in [37] and regular B-spline control points in [86]. In fact, for the regular image data used in Block Matching, we use GPU instead of cluster to perform the partitioning.

Christensen et al. [18] implemented a 3D medical registration algorithm on a 16 core SGI Challenge computer (SIMD architecture) and a 128 \( \times \) 128 core MasPar computer (MIMD architecture), respectively. Their experiments showed that for larger dataset, MIMD implementation had nearly linear performance improvement as the number of cores was increased.

Another non-rigid registration implementation on MIMD parallel processing comput-
ers comes from the work of Salomon et al. [95]. The registration problem is expressed as a minimization of a global, highly non-linear energy function depending on a large number of parameters. The parameter vector is distributed in order to map each variable on one core, then a parallel updating scheme is achieved, which results in a massively parallel implementation. The experiment based on 3D brain MRI from interpatients showed that this method yielded accurate registration and excellent relative speedup. One contribution of this paper is that they present a reasonable assessment for the performance by comparing the relative speedup to its upper bound given by Amdahl's law [4]. The relative speedup is calculated as \( \frac{T_1}{T_N} \), where \( T_1 \) is the execution time using one core and \( T_N \) is the execution time using \( N \) cores. The upper bound is calculated as \( \frac{1}{S+P} \), where \( S \) is the fraction that must be executed in sequential and \( P \) is the fraction that must be executed in parallel. The closer the relative speedup approaches the upper bound, the better the performance is. We will employ this method to evaluate the performance of our parallel incremental FE solver.

To compensate for brain shift, Chrisochoides et al. [17] parallelized a feature point-based non-rigid registration by using distributed and grid computing. This registration method formulates brain shift as a functional minimization problem using Partial Differential Equations (PDEs) whose solution is approximated by Finite Element Method (FEM). This functional can be decomposed into a similarity energy and a regularization energy. The similarity energy is computed by Block Matching, a computationally intensive operation.

In this work we parallelize Block Matching using GPU. The results demonstrate that it is relatively easy to map a regular part to a GPU programming model. Furthermore,
higher speedup can be obtained compared to its cluster implementation.

1.2.2 GPU Related Work

Graphics Processing Units (GPUs) have drawn great interest from researchers and industry practitioners in extending GPU computations beyond their traditional uses in graphics rendering. Readers are referred to [74] for an excellent review of general purpose computation using GPU (GPGPU).

Levin et al. [50] implemented a high-performance Thin Plate Spline (TPS) volume warping algorithm that accelerated the application of the TPS nonlinear transformation by combining hardware-accelerated 3D textures, vertex shaders, and trilinear interpolation. Antonio et al. [90] used polynomial mapping as non-rigid transformation and achieved a factor of 4.11 speedup with a single GPU and 6.68 with a GPU pair over CPU-based NRR. Vetter et al. [104] implemented non-rigid registration on a GPU using mutual information and the Kullback-Leibler divergence, and reported that GPU performed up to 5 times faster per iteration than the CPU implementation.

Kubias et al. [47] presented a 2D/3D registration method on GPU. Both DRR generation and the computation of the similarity measure were executed on the GPU. The experimental results showed 3 to 6 speedup. However, their work only focused on the rigid registration. Yoo et al. [97] reported nearly 50% speedup compared to CPU by implementing a clustering-based image registration on GPU. Unlike other GPU implementation, they only used transformation of texture coordinations in vertex program and resampling in fragment program.

Li et al. [51] provided a GPU accelerated Levenberg-Marquardt non-linear opti-
mization method for tracking cardiac motion in CMR images. Their method was only applicable to 2D frame. Unlike the study in [69], they implemented the non-rigid registration using OpenGL environment instead of CUDA, therefore resulting in additional overheads.

Yousfi et al. [111] described a new strategy to implement, on GPU, the computation of image similarity metrics for intensity-based registration. Their experimental results confirmed significant speedup of calculations.

Muyan-Ozcelik et al. [69] implemented Demons algorithm, a non-rigid registration method, on NVIDIA's Quadro FX 5600 GPU with CUDA. Reportedly, they achieved the fastest runtime among the available GPU-based Demons implementations based on 3D CT lung images. As they pointed out, current GPU-based registration needed to be implemented vertically from the ground up. On the contrary, CPU code can be shared by most programmers, and allows for horizontal program development.

In contrast to the previous GPU implementations, we parallelize the irregular part on multicore with a SPMD programming model by means of data partitioning, and therefore avoid redesigning a parallel algorithm.

In our NRR, compared to the Block Matching, the incremental FE solver is only a small bottleneck. We parallelize it due to the consideration of scalability. In the NRR, the Block Matching accounts for about 10% computation, and the FE solver accounts for about 90% computation. According to Amdahl's law [4], without the parallelization of the FE solver, the speedup for $N$ cores is bounded by a factor of $100/(10 + 90/N) \rightarrow 10$ as $N \rightarrow \infty$. That is, although many cores are available, the speedup never reaches a factor of 10 if the FE solver is processed in sequential.
1.3 Non-Rigid Registration Approach

The NRR method we are targeting is based on the concept of energy minimization [20]. A sparse set of registration points within the preoperative brain MRI are identified. The displacement, denoted as vector $D$, between the pre- and intra-operative images is estimated using Block Matching (BM) at each such point. Based on these displacements $D$, the deformation field defined at mesh nodes, denoted as vector $U$, is estimated under the constraint of a biomechanical model.

Registration is formulated as an energy minimization problem:

$$U = \arg \min_U \{(HU - D)^T S (HU - D) + U^T K U\}, \quad (1.1)$$

where $K$ is the stiffness matrix; $H$ is the linear interpolation matrix from the displacements recovered by Block Matching; $S$ is the BM weight matrix.

Regularization of the solution using the mechanical energy is susceptible to outliers and unavoidably contains approximation error [20]. We address this problem by iterative estimation of the displacement:

$$F_0 = 0, \quad F_{i-1} = K U_{i-1},$$

$$[K + H^T S H] U_i = H^T S D + F_{i-1} \quad (1.2)$$

This iterative method reduces the approximation error at each iteration, while rejecting outliers. In the remainder of the chapter we refer to equation 1.2 as incremental Finite Element Solver due to its incremental improvement of the accuracy. Such formulation is robust against outliers and minimizes the approximation error at the expense of longer execution time.
To solve $U$ from equation 1.2, we need to know $D$, which can be obtained using Block Matching. Block Matching is a well-known technique widely used in motion coding, image processing and compression [2, 10, 19, 62, 70, 112]. Block Matching is based on the assumption that a complex non-rigid transformation can be approximated by point-wise translations of small image regions. Considering a block $B(O_k)$ in the floating image centered in $O_k$ and a predefined search window $W_k$ in the reference image, the Block Matching algorithm consists in finding the position $O_m$ in $W_k$ that maximizes the similarity $M$, which can be: mean square difference of intensity (MSD), mutual information (MI), or normalized cross correlation (NCC).

$$O_m = \arg\max_{O_t \in W_k} [M(B(O_k), B(O_t))]$$ (1.3)

**Figure 1.1**: Block Matching. a. Floating Image b. Reference Image.

The graphical description for the matching of one block (2D) is shown as Fig. 1.1. The highlighted regions are a block in the floating image and its corresponding search window in the reference image.

Performing this operation defined by equation 1.3 on every selected block in the floating image produces a sparse estimation of the displacement between the two images,
whose graphical description is shown as Fig. 1.2.

Figure 1.2: Block Matching result. The arrow points to the direction of the displacement and the color scale encodes the norm of the displacement, in millimeters. The metric is NCC and for clarity, only 1% of the displacement field is shown.

1.4 Parallel Implementation

Our NRR method consists of two computationally intensive components: Block Matching and the incremental Finite Element solver. Block Matching is characterized by regular data and regular operations, therefore, we implement it on GPU, designed to perform bulk computations of a kernel code on different input data. The incremental Finite Element solver operates on irregular data structures requiring synchronization and communication. Such computations cannot fully benefit from GPU architecture [81], so we implement it on multicore. A nonlinear FE solver has been implemented by [102] on GPU. Compared to the GPU implementation, a multicore SPMD model allows higher level parallelism. We can take full advantage of the existing sequential code to parallelize it horizontally instead of vertically by some preprocessing: data partitioning and renumbering.

Our parallel NRR framework is shown in Fig. 1.3, in which Block Matching and
incremental Finite Element solver are parallelized on GPU and multicore, respectively. Fig. 1.4 illustrates the data parallel partitioning method on the cooperative architecture. The regular image data is distributed across a CUDA programming model by dividing the image into image blocks. The irregular mesh is distributed across multicore by decomposing mesh into sub-meshes. Each GPU thread, denoted by T, is in charge of a small image block. Note that the notation Block in Fig. 1.4 is GPU thread block of the grid, which will be discussed in the GPU programming model. This data parallel NRR does not require designing a new parallel algorithm. We need only to partition the data on its suitable architecture, which enables us to reach the goal: accelerating using architecture.

In the following sections, we will present GPU implementation of 3D Block Matching and multicore implementation of the Finite Element solver.

1.4.1 GPU Implementation of 3D Block Matching

In this section, we first briefly introduce the GPU programming model, and then describe how to map an existing regular algorithm (Block Matching) to this model. Finally, two important performance related issues about GPU implementation: memory selection
and GPU optimization, are discussed in detail.

1.4.1.1 CUDA Programming Model

The NVIDIA Compute Unified Device Architecture (CUDA) [73] abstracts GPU as a general-purpose multithreaded SIMD (single instruction, multiple data) architectural model, and offers a C like interface supported by a compiler and a runtime system for GPU programming, which simplifies the writing of a GPU program. The CUDA programming model, as shown in Fig 1.5, organizes threads by grid, which is an array of blocks containing an array of threads [73]. All blocks in a grid have the same number of threads. Each thread block has a unique two dimensional coordinate given by the CUDA specific keywords: blockIDx.x, blockIDy.y. Each thread block is, in turn, organized as a three dimensional array of threads. The coordinates of threads in a block are uniquely
defined by three thread indices: $threadIdx.x$, $threadIdx.y$, $threadIdx.z$. The threads in a thread block can cooperate with each other by synchronizing their execution or using shared memory. Two threads from two different blocks cannot cooperate. A kernel is the core code to be executed as a grid of parallel threads. It performs on different sets of data using its ID specified by its block indices and thread indices in a SIMD fashion.

The execution resource of GPU are organized into Streaming Multicore (SMs). For example, GeForce 8800GTX has 16 SMs and each SM has 8 Streaming Processors (SPs). Once a block is assigned to a SM, all the threads in the block will be divided into 32-thread units termed as warps. In fact, warps are not part of the CUDA language definition. We mention it here because knowledge of warps can be helpful in understanding and optimizing the performance of CUDA application, such as the optimization of BM, which will be discussed later.

The warp is the unit of thread scheduling in SMs. At any point in time, only one of them can actually be executed by the hardware. However, to hide long latency operations, we want many warps residing in SM at any point in time. If one warp executes long latency operation, such as accessing the global memory, we can put it into a waiting queue and schedule other ready warps to execute. In the sense of hiding latency, it is better to have as many warps as possible residing in SMs. However, the maximum number is limited by the hardware resources. SM occupancy, the ratio of active warps to the maximum number of warps supported on a SM, can be used to measure the degree to which the latency can be hidden. Generally, higher occupancy means higher possibility to hide latency and vice versa. However, the occupancy is only suitable for the measurement of the performance for bandwidth bound applications. For
Block Matching, a computation bound application, occupancy is meaningless. Therefore, we need to replace the search algorithm from occupancy-based heuristic algorithm with the exhaustive algorithm in the extended GPU optimization tool. We will discuss this tool in the GPU optimization section.

From Fig. 1.5, we can see that the CUDA programming model is characterized by two-level hierarchy and regularity, which means it is very suitable for the regular algorithm. Therefore, we migrate BM from cluster implementation to GPU implementation.

1.4.1.2 Mapping to the CUDA Programming Model

The key to efficient GPU implementation is in the mapping of a sequential program to the CUDA programming model, which is demonstrated in Fig. 1.6. Due to the regularity
of both BM and the CUDA programming model, the mapping is straightforward: the outer loop is mapped to a GPU Grid, the inner loop is mapped to a GPU thread block, and the NCC computation is mapped to the GPU kernel function. More specifically, for each registration point and the corresponding block in the floating image, we assign a separate CUDA thread to calculate its similarity within a different portion of the search window. The size of the portion depends on the sizes of the thread block and the search window. For instance, if the thread block is $4 \times 4 \times 4$ and the search window is $8 \times 8 \times 8$, each thread will be responsible for the calculation of the similarity within a $2 \times 2 \times 2$ portion. It is obvious that there should exist an optimal thread block for a specific search window. We will present a GPU optimization tool to automatically find this optimal thread block in the next section. The maximum similarity can be evaluated by parallel reduction of the computed similarity values. As we map NCC calculation to kernel function, the only change we make is to access data from GPU texture memory instead of CPU memory, more specifically, $CPU Array[i][j][k] \Rightarrow tex3D(GPUTexture, k, j, i)$.

GPU memory selection heavily impacts the performance. We implement BM on GPU global memory with a coalesced access pattern and texture memory, respectively, and find the latter is at least twice as faster the former.

1.4.1.3 Memory Issue

The data needed for BM computation includes:

- 3D floating images and reference images
- 1D array to store the selected block in floating image
- Calculated similarities
for each selected block floBlkOᵢ in Floating image do
  Define search window Wₒᵢ in Target image
  for each block tarBlkOᵢᵢ in Wₒᵢ do
    Calculate similarity s using NCC between floBlkOᵢᵢ and tarBlkOᵢᵢᵢ Kernel function
  end for
end for
Find the maximum s and corresponding Distance Dₒᵢ

Figure 1.6: Mapping from sequential BM to GPU programming model. Left: Sequential BM. Right: GPU BM.

The image data is stored in the texture memory instead of global memory because the texture memory supports cache, broadcast, and hardware addressing. Furthermore, the texture access does not require a coalesced access pattern (it is required for global memory access if we expect to reach desirable performance). To make full use of the 3D locality of texture memory, we use 3D texture and bind it to a 3D CUDA array instead of linear memory. However, to store selected blocks in floating image we use 1D texture and bind it to linear memory instead of CUDA array. This is due to the following considerations:

- We do not need filtering or normalized coordinates, which are supported in texture combined with CUDA array
- We can bind the texture directly to device memory without copies to intermediate CUDA arrays
- 3D locality is not an issue for selected blocks
- We can address larger textures (up to $2^{27}$ elements). For a texture reference bound to a one dimensional CUDA array, the maximum width is only $2^{13}$, which is too small to store 50K to 100K selected blocks.
The calculated similarity is stored in the shared memory to facilitate the reduction operation.

### 1.4.1.4 Optimization of GPU Program

As mentioned above, the mapping of the regular algorithm to the CUDA programming model is straightforward. The challenge is what is the optimal mapping? In other words, how many data should be assigned to a GPU thread to maximize performance? In addition to the search window, the image block is another important input parameter, which affects the precision of NRR [53]. This optimal partitioning problem can be formalized as:

```
Problem: Given input < ImageBlock, SearchWindow >, find optimal thread block, also known as GPU execution configuration: < blockSizeX, blockSizeY, blockSizeZ >.
```

GPU execution configuration significantly impacts the performance of GPU programs. As an example, 6 out of 7 benchmarks from NVIDIA SDK gain speedup ranging from 1.5 to 6.6 times when execution configuration is optimized [57].

It is difficult to determine the optimal configuration because there are many constraints imposed on the execution configuration including:

- Warp size (thread block size should be a multiplier of 32 because the warp is a GPU scheduling unit)
- The number of shared memory and register
- The maximum number of active blocks per SM
- The maximum number of active warps per SM
• The maximum number of active threads per SM

• The maximum number of threads per block

To find the optimal GPU configuration, we developed a GPU optimization tool: G-ADAPT in [57]. In this chapter, we extend this tool and apply it on the optimization of a real world application: Block Matching. The basic idea and implementary details of G-ADAPT are presented in the next chapter. The first thing in order to locate the optimal GPU configuration is to define the search space. In our previous work [57], we used a \texttt{pragma} to define the space. The \texttt{pragma} will be inserted into a CUDA program file, usually at the beginning of the file. For BM, we define a specific \texttt{pragma} as shown below. The first sentence defines the range of the search space, and the next three sentences identify the variables, which constitute the search space.

```c
#pragma mrange3 low high step
#define Block_Size.X 16
#define Block_Size.Y 4
#define Block_Size.Z 2
```

For 3D BM, the \texttt{pragma} we use is: low = 32, high = 128, step = 2. The space defined by this \texttt{pragma} is equivalent to the space defined by the following,

\begin{equation}
32 \leq \text{Block\_Size\_X} \times \text{Block\_Size\_Y} \times \text{Block\_Size\_Z} \leq 128 \tag{1.4}
\end{equation}

where \text{Block\_Size\_X}, \text{Block\_Size\_Y} and \text{Block\_Size\_Z} are power of step = 2.

There is a total of 84 combinations satisfying the above \texttt{pragma} and Specifications for Compute Capability 1.2 of CUDA (\text{Block\_Size\_Z} \leq 64) [73]. We set the low bound to 32 because we do not want the thread size to be smaller than the warp size. The number
of consumed registers in the thread dictates the upper bound. The global register pool is $8K$, and each thread consumes 33 registers, so the maximum number of threads closer to power of 2 is 128. Shared memory is not a limitation for BM, because each block consumes only 2180 bytes shared memory (total $16KB$). In order to guarantee the thread size to be a multiple of the warp, we set $\text{step}$ to 2.

For a specific input $<$ ImageBlock, SearchWindow $>$, the size of the search space can reach 84, so the size of the search space for all inputs will be $84 \times \#\text{ImageBlock} \times \#\text{SearchWindow}$. The search space is so large that it is not practical to find the optimal execution configuration by trial.

Once the search space is obtained, the tool exhaustively searches for the optimal execution configuration in the search space. In our previous work [57], to explore the search space we used a greedy algorithm, which steps along the direction maximizing the occupancy. Maximizing the occupancy can help cover latency during global memory loads that are followed by $\_\text{syncthreads()}$. However, as we mentioned in Section 1.4.1.1, occupancy is only a useful sign for bandwidth bound applications. BM is bottlenecked by computation, so we replaced the previous greedy algorithm with an exhaustive method.

The pseudo code of this GPU optimization tool is shown in Algorithm 1 and the results of this algorithm will be presented in Section 1.5.

1.4.2 Multicore Implementation of Incremental Finite Element Solver

Compared to Block Matching, the incremental Finite Element solver is only a small bottleneck, but for the consideration of scalability, we still need to parallelize it. The Finite Element solver includes two steps: assembling and solver of a linear system of
Input: <ImageBlock, SearchWindow>, originalBM.cu with pragma
Output: <Block_Size_X, Block_Size_Y, Block_Size_Z>
1. generate search space based on equation 1.4
2. insert CUDA event before and after kernel function // record kernel execution time
3. for each tuple <X, Y, Z> in search space do
   4. transform originalBM.cu into newBM.cu using variable replacement:
      Block_Size_X = X, Block_Size_Y = Y, Block_Size_Z = Z
   5. compile transformed code using NVCC
   6. execute and record the kernel running time output by GPU event
   7. output <running time, execution configuration> into database
   8. end for
9. find <Block_Size_X, Block_Size_Y, Block_Size_Z> corresponding to minimum running time in the database.

Algorithm 1: Pseudo code of GPU optimization tool.

equations. The assembling is performed on unstructured mesh and the solver is involved in lots of communications and synchronizations, which are not good aspects of GPU. To parallelize the Finite Element solver, we need to distribute the mesh and registration points among the cores, which is different from the distribution of the image and selected blocks in Block Matching. We present a parallel partitioning method, which consists of element partitioning and vertex partitioning. Furthermore, to reduce the impact on the existing sequential algorithm, a local renumbering is used and its communication to the global renumbering can be facilitated by a mapping table. After the partitioning and renumbering, the system of equations can be assembled and solved. In the following sections, we will discuss these issues in detail.

1.4.2.1 Parallel Partitioning

The partitioning of the unstructured mesh includes two steps: element partitioning and vertex partitioning. The purpose of partitioning is to obtain a matrix, which is efficient in evaluating matrix-vector multiplication—a major computational component of the CG
The desirable matrix is shown in Fig. 1.7, where different colors distinguish different cores. Each core holds two kinds of entries: interface entries and non-interface entries. The non-interface entries do not involve communication as performing matrix vector multiplication, but the interface entries do. Our strategy consists of minimizing the number of interface entries in order to reduce the communication, and distributing the non-interface entries by favoring the cores with a minimum number of entries to reach load balancing.

Figure 1.7: Desirable Matrix.

Initially, elements and vertices can be distributed across cores without any specific requirements. Then, element partitioning is used to minimize interface and vertex partitioning is used to reach load balancing.

**Element partitioning** We employ ParMETIS, a leading partition tool [63], to perform the element partitioning. ParMETIS can dramatically reduce the time spent in communication by computing mesh decompositions such that the numbers of interface elements are minimized [44].

We use Fig. 1.8 to illustrate the parallel element partitioning. ParMETIS takes an initial partitioning and an element graph as inputs. The element graph describes
the relation of these elements, which can be generated by locating the pair of elements sharing the same edge for 2D mesh or by locating the pair of elements sharing the same face for 3D mesh. We use red and green colors to denote different cores. If a vertex/element has red color, this vertex/element is owned by red core. As shown in Fig. 1.8, initially the green core holds elements 0,1,2 and 6 and the red core holds elements 3, 4 and 5. ParMETIS will partition the graph along the solid line instead of the dash line in order to minimize the cutting edge (an approximation to communication). ParMETIS tells us which core should hold which elements as shown in the top right figure, where 3(g) denotes element 3 should be assigned to green core. According to the partitioning result, we need to move the element to the correct core. This can be done by using gather and scatter operations. Firstly, we gather elements 0 to 6 into a distribute array and then scatter them into their corresponding cores. As a result, the green core holds element 0, 1, 3 and 6, and the red core holds elements 2, 4 and 5. In addition to moving elements, the registration points also need to be moved along with
the element. We let each element keep a list of registration points and move them along with the element. Next we need to partition the vertices and then renumber them.

**Vertex partitioning** Element partitioning minimizes the number of interface elements and, in turn, minimizes the number of interface entries shown in Fig. 1.7. However, it does not take load balancing into account. Because the load is proportional to the number of the vertices held in the core, we need to decide how to distribute interface vertices across cores. For example, if we let a green core hold interface vertices 3 and 4, it will lead to a load imbalance matrix as shown in Fig. 1.9 a. We use a simple greedy algorithm for vertex partitioning by always assigning the interface vertices to the core with the minimum number of vertices. Fig. 1.9 b shows the result of this algorithm. Note that this simple greedy algorithm cannot guarantee absolute load balancing, but it can alleviate the impact of load imbalance while maintaining minimum communication.

![Figure 1.9: Vertices partitioning. A load imbalance matrix before vertex partitioning (a) and a load balance matrix after vertex partitioning (b).](image)

An example of the 4-way (the number of cores on a typical workstation) partitioned mesh is given in Fig. 1.10. Different colors denote different sub-meshes. One sub-mesh is visualized with wireframe to show its inner registration points.
1.4.2.2 Renumbering

Once each core has its own vertices, global renumbering can be performed straightforwardly just by renumbering the vertices core by core. To minimize the change of the sequential code, we use a local numbering strategy and let each core keep a mapping table to relate the local numbering with the global numbering. In the right figure of Fig. 1.11, the numbers in the blue circles are global numbering, and the numbers in the white circles are local numbering. The advantage of this approach is that the sub-mesh can be considered as a standalone mesh on each of the cores, while the communication with the non-local sub-meshes can be facilitated by the mapping table. With this mapping table, replacing the access to Node\[i\] with Node[MappingTable[i]], the original sequential code is easily parallelized on multicore.

After partitioning and renumbering, each core holds a sub-mesh with contiguous local numbering, vertex list, and local to global mapping table. Then we can assemble our linear system of equations.
1.4.2.3 Assembling

Based on equation 1.2, we need to assemble the matrices $K$ and $H^TSH$. Construction of the stiffness matrix $K$ has been well documented elsewhere [9]. In order to improve the performance of assembling matrix $H^TSH$, we directly set the values at its corresponding entries instead of assembling $H$, and then multiplying its transpose with $S$ and $H$.

Each registration point $k$ (i.e. the block center $O_k$) contained in a tetrahedron with vertex $(v_0, v_1, v_2, v_3)$ will contribute to $H^TSH$ at position $(v_i, v_j), i, j \in [0:3]$ with the submatrices $H_{v_i}S_kH_{v_j} = h_i \times S_k \times h_j$, in which $S_k$ are $3 \times 3$ confidential submatrix and $h_j, j \in [0:3]$ are linear interpolation factors [20]. The linear interpolation factors are computed for the block center $O_k$ inside the tetrahedron with

$$
\begin{bmatrix}
  h_1 \\
  h_2 \\
  h_3 \\
  h_4
\end{bmatrix} =
\begin{bmatrix}
  v_1^x & v_1^y & v_1^z & v_1^x & v_1^y & v_1^z \\
  v_2^x & v_2^y & v_2^z & v_2^x & v_2^y & v_2^z \\
  v_3^x & v_3^y & v_3^z & v_3^x & v_3^y & v_3^z \\
  1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}^{-1}
\begin{bmatrix}
  o_1^x \\
  o_1^y \\
  o_1^z \\
  o_1^t
\end{bmatrix}
$$

Assembling yields a linear system $AU = b$, where $A$ and $b$ are distributed across cores as shown in Fig. 1.9 b. Because $A$ is a semi-positive sparse definite matrix, we use $CG$ to solve it. This component is computed in parallel and facilitated by PETSc implementation [78].
The following algorithm sketches the above discussion. Note that in Algorithm 2, steps 1 and 2 are performed by the core with rank 0, and step 3 to step 10 are performed in parallel.

<table>
<thead>
<tr>
<th><strong>Input:</strong> Mesh: element list and vertex list</th>
<th><strong>Output:</strong> Node displacement vector $U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initiate partitioning</td>
<td>1. Distribute element and vertex lists</td>
</tr>
<tr>
<td>2. Create element graph</td>
<td>2. Create element graph</td>
</tr>
<tr>
<td>3. Elements partitioning</td>
<td>3. Minimize communication</td>
</tr>
<tr>
<td>4. Elements movement</td>
<td>4. Elements movement</td>
</tr>
<tr>
<td>5. Vertices partitioning</td>
<td>5. Reach load balancing</td>
</tr>
<tr>
<td>7. Global and Local renumbering</td>
<td>7. Minimize the change to the sequential code</td>
</tr>
<tr>
<td>8. Create mapping table</td>
<td>8. Facilitate mapping</td>
</tr>
<tr>
<td>10. Solve $U$</td>
<td>10. Solve $U$</td>
</tr>
</tbody>
</table>

**Algorithm 2:** Data parallel incremental Finite Element solver.

### 1.5 Results

In this section, we will show our experiment results concerning the GPU implementation of Block Matching, the GPU optimization for Block Matching, and the multicore implementation of the incremental Finite Element solver.

#### 1.5.1 GPU VS. Cluster on Block Matching

We compared the performance of the Block Matching on a typical modern workstation (Dell Precision T3400 equipped with NVIDIA GeForce 8800 GT GPU) with its MPI implementation running on a 8-node cluster (each node is Dell PowerEdge SC1435, 2×dual-core Opteron 2218, 2.6 GHz CPU). The results were collected for computations on 6 retrospective brain tumor resection cases, with the imaging parameters similar to the ones used for acquisition of brain imaging in the SPL Brain Tumor Resection dataset.
Fig. 1.12 shows the comparison of the performance for the considered implementations. Compared to the 4-node cluster (16 CPUs), the minimum speedup is 3.9 (case 1) and the maximum speedup is 7.7 (case 5). Compared to the 8-node cluster (32 CPUs), the minimum speedup is 1.9 (case 1) and the maximum speedup is 3.8 (case 5).

1.5.2 GPU Optimization of Block Matching

The trace results for \( \text{ImageBlock} = 4 \times 4 \times 4, \text{SearchWindow} = 10 \times 10 \times 10 \) are shown in Fig. 1.13. \( 8 \times 8 \times 2 \) is the optimal GPU thread block size found by the GPU optimization tool. \( 16 \times 4 \times 2 \) is our initial choice for thread block size. Compared to our initial choice, this tool can reduce the runtime by \( (15.8 - 12.2) \div 15.8 = 23\% \).

In fact, to avoid selecting a worse block size, our initial selection has taken into full account the factors, such as 3D locality of texture memory. A worse block size heavily impacts the performance, as shown in Fig. 1.14. For search window \( 7 \times 7 \times 7 \), the speedup
can reach 7.2 with optimal block size, which is so large that it is necessary to use the GPU optimization tool to find the optimal block size.

Because Block Matching is widely used in other image processing fields, we summarize its optimal configuration in Table 1.1 by executing Algorithm 1 for different $\langle \text{ImageBlock}, \text{SearchWindow} \rangle$. Thus, even without this tool, users can select the
optimal configuration according to their specific \(<ImageBlock, SearchWindow>\). we hope our effort can facilitate the use of BM in medical image processing field.

**Table 1.1:** Optimal GPU execution configuration for \(<ImageBlock, SearchWindow>\). Row: image block dimensions. Column: search window dimensions.

<table>
<thead>
<tr>
<th>Image Block Dimensions</th>
<th>Search Window Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 x 5 x 5</td>
<td>&lt;4 x 4 x 4&gt;</td>
</tr>
<tr>
<td>6 x 6 x 6</td>
<td>&lt;8 x 8 x 2&gt;</td>
</tr>
<tr>
<td>7 x 7 x 7</td>
<td>&lt;4 x 4 x 4&gt;</td>
</tr>
<tr>
<td>8 x 8 x 8</td>
<td>&lt;8 x 8 x 2&gt;</td>
</tr>
<tr>
<td>9 x 9 x 9</td>
<td>&lt;4 x 4 x 4&gt;</td>
</tr>
<tr>
<td>10 x 10 x 10</td>
<td>&lt;8 x 8 x 2&gt;</td>
</tr>
</tbody>
</table>

1.5.3 Incremental Solver

The CPU used in the experiment is 2 x Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz. The runtime of the solver depends on the size of the mesh. Three different sizes of the mesh ranging from small, medium, to large were generated based on case 6 using the sequential mesh generator developed in our group. A thoroughly evaluated biconjugate gradient solver implemented within Gmm++ library [31] is used for comparison with our parallel incremental solver. The runtime required for partitioning, matrix and vector assembling, and incremental solver are listed in Table 1.2.

**Table 1.2:** Performance comparison between sequential and parallel solver.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Vertices</th>
<th>Tetras</th>
<th>Sequential(time:second)</th>
<th>Parallel(time:second)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Assemblage</td>
<td>Solver</td>
<td>Partition</td>
</tr>
<tr>
<td>1607</td>
<td>7272</td>
<td>6.780</td>
<td>23.560</td>
<td>0.040</td>
<td>0.450</td>
</tr>
<tr>
<td>3526</td>
<td>17137</td>
<td>7.500</td>
<td>35.80</td>
<td>0.10</td>
<td>0.43</td>
</tr>
<tr>
<td>6737</td>
<td>33931</td>
<td>8.040</td>
<td>43.320</td>
<td>0.12</td>
<td>0.49</td>
</tr>
</tbody>
</table>

**Table 1.3:** Performance evaluation for parallel solver on large mesh. Mesh loading and element graphic creation are performed using one core.

<table>
<thead>
<tr>
<th>#Processors</th>
<th>MeshLoading</th>
<th>InitParAndGraph</th>
<th>ParallelPartitioning</th>
<th>Assemblage</th>
<th>Solver</th>
<th>TotalTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Processor</td>
<td>1.02</td>
<td>0.04</td>
<td>0.14</td>
<td>1.05</td>
<td>11.25</td>
<td>13.5</td>
</tr>
<tr>
<td>4 Processors</td>
<td>1.00</td>
<td>0.03</td>
<td>0.09</td>
<td>0.49</td>
<td>4.66</td>
<td>6.27</td>
</tr>
</tbody>
</table>

32
Compared with the sequential solver, our parallel solver needs additional partitioning time, but the overall gain in performance is significant.

The above results demonstrate the desirable speedup brought by this parallel solver, but they cannot truly reveal the performance of the parallel solver. Simply comparing the number of cores is not adequate. As we mentioned in Section 1.2, Salomon et al. [95] proposed an adequate method to evaluate the performance by comparing the relative speedup to its upper bound given by Amdahl’s law. The sequential parts in the FE solver are mesh loading, partitioning initiation, and element graph creation, which account for about 9% of the total computation and can be calculated from Table 1.3. The upper bound for 4 cores can be calculated as: \( \frac{100}{9+9/4} = 3.2 \). The relative speedup is about 2.2 for large mesh, which is obtained by calculating the ratio between the execution time on one core and four cores from the data in Table 1.3. As we can see, the relative speedup is close to its upper bound.

We evaluated our parallel implementation on tumor resection cases. The registration results are shown in Fig. 1.15, in which we highlighted the discrepancy along the boundary. It is clear that the discrepancy reduces as the number of iteration increases.

Each part gains desirable speedup, reducing the total runtime to less than 1 minute (calculated using large mesh in Table 1.2 for case 1 in Fig. 1.12: \( 45 + 0.12 + 0.49 + 4.66 = 50.27s \)). The algorithms in both regular and irregular parts are unchanged, therefore maintaining the accuracy of the sequential code. Our experiments (see Fig. 1.16) show that the difference in accuracy between the parallel and the sequential implementations is below 0.006mm (large mesh), which is normal for parallel implementation due to concurrency.
1.6 Discussion

The techniques provided in this chapter can be used separately on different fields. For instance, we can use GPU BM for motion tracking, use the GPU optimization tool to find optimal GPU configuration for GPU programs, and use the multicore FE solver for surgery simulation. Combining these techniques, we can address not only image registration, but also other Partial Differential Equation (PDE) model-based image processing and analysis. The technique of utilizing the PDE model to perform image processing and analysis arose as early as 1985. Readers are referred to [32] for a review of this kind of technique from 1985 to 2000. Recently, this technique has been widely used for image denoising, deblurring, invariant smoothing and restoration [3,103,106]. This PDE model-based image processing and analysis is characterized by 1) an irregular solver to find the numerical solution of PDE and 2) a regular algorithm to find the boundary con-
Our NRR also falls into this category. Block Matching is used to find a sparse displacement field (boundary condition), and then this displacement field is applied on the FE solver, which is derived from linear elastic PDE model, to get the unique solution. We hope the techniques developed in this chapter can facilitate the use of this kind of PDE model-based image processing and analysis on the widely available cooperative architecture in commodity PCs.

1.7 Conclusion

As we parallelize the existing sequential algorithm, we expect to accelerate it using available parallel architecture instead of designing it from scratch in order to maintain the established accuracy and robustness of the sequential code evaluated in the clinic. We present a parallel implementation based on the cooperative architecture and show how to best utilize both GPU and multicore for the parallelization of an existing sequential NRR algorithm. Our approach separates the sequential NRR into a regular part and an irregular part and implements them on GPU and multicore, respectively. Both the regular part and the irregular part do not require designing a new parallel algorithm,
but by means of data partitioning to facilitate the mapping of the sequential algorithm to parallel architecture to gain speedup.

The advantages of this method are 1) established accuracy, robustness, and performance remain, 2) desirable speedup gains with only minor changes to the sequential code, and 3) desirable price performance ratio. Compared to a 8-node cluster, our parallel NRR can reduce the execution time to less than 1 minute on a 11 times cheaper workstation. Note that "11 times" is estimated very roughly by assuming a typical modern workstation (Dell Precision T3400 equipped with NVIDIA GeForce 8800 GT GPU card) is less than $1,000 and each node (Dell PowerEdge SC1435, 2 x dual-core Opteron 2218, 2.6 GHz CPU) is about $1,400 (8 nodes). The network and management fees are not taken into account.
Chapter 2

GPU Program Optimizations

2.1 Introduction

As a specialized single-chip massively parallel architecture, Graphics Processing Units (GPU) have shown orders of magnitude higher throughput and performance per dollar than traditional CPUs. The properties have recently drawn great interest from researchers and industry practitioners in extending GPU computation beyond the traditional uses in graphics rendering [8]. Besides hardware innovations, progresses in programming models have significantly improved the accessibility of GPU for general-purpose computing. In particular, the NVIDIA Compute Unified Device Architecture (CUDA) [73] abstracts GPU as a general-purpose multithreaded SIMD (single instruction, multiple data) architectural model, and offers a C-like interface supported by a compiler and a runtime system for GPU programming. CUDA simplifies the development of GPU programs. However, developing an efficient GPU program remains as challenging as before, if not more. Four aspects account for these challenges. The first is the complexity in GPU architecture. On an NVIDIA GeForce 8800 GT, for example,
there are over one hundred cores, four types of off-chip memory, hundreds of thousands of registers, and many parameters (e.g. maximum number of threads per block, thread block dimensions) that constrain the programming. The second difficulty is that the multi-layered software execution stack makes it difficult to predict the effects of a code optimization. A special difficulty with CUDA is that, currently, a GPU program has to be compiled by the NVIDIA CUDA compiler (NVCC) and run on the NVIDIA CUDA runtime system, of which some details of both are not yet disclosed. Third, an optimization often has multiple effects, and the optimizations on different parameters often strongly affect each other. Finally, some GPU applications are input-sensitive. The best optimizations of an application may be different when different inputs are given to the application. Together, these factors make manual optimizations time consuming and difficult to attain the optimal, and, at the same time, form great hurdles to automatic optimizations as well.

On the other hand, optimizations are particularly important for GPU programming. Because of the tremendous computing power of GPU, there can be orders of magnitude performance difference between well optimized and poorly optimized versions of an application [8]. Several recent studies have tried to tackle the problem through empirical search-based approaches. Ryoo et al. [92] have defined efficiency and utilization models for GPU programs to help prune the optimization space. Baskaran et al. [8] have developed a polyhedral compiler model to optimize global memory accesses in affine loop nests, and used model-driven empirical search to determine the levels of loop unrolling and tiling. Although both studies have shown promising results, neither of them have explored the influence of program inputs on the optimization. Program inputs refer
to both the values and other related properties (e.g. dimensions of an input matrix) of the inputs given to a program. In this work, we initiate an exploration in this new dimension, showing that program inputs may influence the effectiveness of an optimization by up to a factor of 6. Based on the exploration, we develop a tool, G-ADAPT (GPU adaptive optimization framework), to efficiently discover near-optimal decisions for GPU program optimizations, and then tailor the decisions for each program input. More specifically, this work makes three major contributions. First, we develop a source-to-source compiler-based framework, G-ADAPT, for empirically searching for the best optimizations for GPU applications. The framework is distinctive in that it conducts program transformations and optimization-space search in a fully automatic fashion, and, meanwhile, offers a set of pragmas for programmers to easily incorporate their knowledge into the empirical search process. Second, this work examines the influence of program inputs on GPU program optimizations. We are not aware of any previous studies in this direction. The lack of such explorations may be due to a common intuition that as most GPU applications divide a task into small sub-tasks, the changes in their inputs do not affect the optimizations as long as the sub-tasks remain similar. Our experiments show that, although many GPU kernels conform to that intuition, some GPU programs exhibit strong input-sensitivity due to their computation patterns and the interplay with optimization parameters. Finally, based on the exposed input sensitivity, we construct a crossinput predictor by employing statistical learning (Regression Trees in particular) to make G-ADAPT automatically tailor optimizations to program inputs. As far as we know, this is the first framework that allows cross-input adaptive optimizations for GPU applications.
Experiments on NVIDIA GeForce 8800 GT GPU show that the adaptive optimization framework can predict the best optimizations for 7 GPU applications with over 93% accuracy. The adaptive optimization improves the program performance by as much as several times in comparison with manually optimized versions.

We organize the chapter as follows. Section 2.2 provides some background on GPU and its programming model. Section 2.3 discusses the challenges in GPU program optimizations. Section 2.4 describes G-ADAPT as our solution to those challenges. Section 2.5 reports evaluation of the framework. Section 2.6 discusses the training overhead and some other complexities of G-ADAPT. We conclude this chapter with a brief summary in Section 2.7.

## 2.2 Background on GPU Architecture and CUDA

This work uses the NVIDIA GeForce 8800 GT GPU as the architecture. It is a single-chip, massively parallel architecture with 112 cores and 512 MB off-chip memory. The GPU contains 14 streaming multiprocessors (SMs). Each SM contains 8 streaming processors (SPs) or cores, with the clock rate set at 1.51 GHz. Each SM also includes 2 special function units (SFUs) for the fast execution of complex floating point operations, such as sine and cosine. Besides the computing units, on each SM, there are 8192 32-bit registers and 16 KB shared memory. Unlike cache, the shared memory has to be managed explicitly in each GPU application.

The off-chip memory includes a 512 MB global memory, which is both readable and writable by every SP, and some constant memory and texture memory, which can only be read by the SPs. The constant memory and texture memory are cachable thanks to
some on-chip cache, but the global memory is not.

Directly programming such a massively parallel architecture is difficult; CUDA, a programming model developed by NVIDIA, simplifies GPU programming by a set of abstractions. The programming interface of CUDA is ANSI C with certain extensions. A GPU application written in CUDA is composed of CPU code and GPU kernels. CUDA abstracts the execution of a GPU kernel as multithreaded SIMD computation. The threads are grouped into many warps with 32 threads in each. Those warps are organized into a number of thread blocks. Each time, the runtime system maps one or more thread blocks to an SM. The warps in those blocks are dynamically scheduled to run on the SM. In GeForce 8800 GT, half of a warp is an SIMD execution unit. If one warp is stalled (e.g. due to memory accesses), the other warps can be switched in with nearly zero overhead. Therefore, the number of warps or thread blocks that are mapped to an SM determines the effectiveness of the pipelining execution in hiding latency. As the thread-block size determines the mapping of blocks on SMs, it is an important parameter in GPU program optimizations.

Threads may communicate in the following ways. Threads in a block may communicate through shared memory and be synchronized by a syncthreads primitive. But communications between threads that belong to different thread blocks have to use off-chip global memory; the communications are hence slow and inflexible.

2.3 Challenges in the Optimization of GPU Programs

Although CUDA simplifies GPU programming, it reduces little if any difficulty in optimizing GPU applications; to some degree, the added abstractions even complicate the
optimization as they make performance prediction more difficult.

a) Optimizations: There are two main ways to improve the performance of a GPU program: maximization of the usage of computing units, and reduction of the number of dynamic instructions. Optimizations to reach the first goal fall into two categories. The first includes those techniques that attempt to increase the occupancy of the computing units. One typical example is to reduce resource consumption of a single thread so that multiple thread blocks can be assigned to one SM at the same time. Multiple blocks may help keep the SM busy when the threads in one block are stalled for synchronization. Example transformations for that purpose include the adjustment of the number of threads per block, and loop tiling. The second category contains the techniques that try to reduce latencies caused by memory references (or branches). Examples include the use of cachable memory (e.g. texture memory), the reduction of bank conflicts in shared memory, and coalesced memory references (i.e. when threads in a warp reference a sequence of contiguous memory addresses at the same time).

Optimizations to reduce the number of dynamic instructions include many traditional compiler transformations, such as loop unrolling and common subexpression elimination. Although the CUDA compiler, NVCC, has implemented many of these techniques, researchers have seen great potential to adjust some of those optimizations, such as the levels of loop unrolling [8,92].

b) Challenges: It is difficult to analytically determine the best optimizations for a GPU application for three reasons. First, it is often difficult to accurately predict the effects of an optimization on the performance of the GPU application. The effects are often non-linear as Ryoo et al. have shown [92]. The undisclosed details of the CUDA
compiler and other abstractions add further unpredictability. Second, different optimizations often affect each other. Loop unrolling, for example, removes some dynamic instructions and exposes certain opportunities for the instruction scheduler to exploit; but it also increases register pressure for each thread. Given that the number of registers in an SM is limited, it may result in fewer threads an SM can hold, and thus affect the selection of thread-block size. Third, the many limits in GPU hardware add further complexity. In GeForce 8800 GT, for instance, the maximum number of threads per block is 512, the maximum number of threads per SM is 768, the maximum number of blocks per SM is 8, and at each time, all the threads assigned to an SM must use no more than 16 KB shared memory and 8192 registers in total. These constraints plus the unpredictable effects of optimizations make it extremely difficult to build an accurate analytical model for GPU optimization.

Empirical search serves as an alternative strategy for determining the best optimizations, whereby the optimizer searches for the best optimization parameters by running the GPU application many times, each time with different optimizations applied. Three obstacles must be removed before this solution becomes practical. First, a compiler is needed for abstracting out the optimization space and transforming the program accordingly. Second, effective space prunes are necessary for the search efficiency, especially when the optimization space is large. Third, the optimizer must be able to handle the influence of program inputs. Our study shows that the best values of optimization parameters of some GPU programs are different for different inputs. For example, an optimization suitable for one input to a reduction program degrades the performance of the program on another input by as much as 640%. For such programs, it is desirable
to detect the input-sensitivity and make the optimization cross-input adaptive.

2.4 Adaptive Optimization Framework

G-ADAPT is our solution to the challenges in GPU program optimization. It is a cross-input adaptive framework, unifying source-to-source compilation, performance modeling, and pattern recognition. This section first gives an overview of the framework, and then elaborates on every component in the framework.

2.4.1 Overview

Fig. 2.1 shows the structure of G-ADAPT. Its two parts, separated by the dotted vertical line, correspond to two stages of the optimization. The task of the first stage, shown as the left part in Fig. 2.1, is to conduct a series of empirical searches in the optimization space of the given GPU program. During the search, a set of performance data, along with the program input features, are stored into a database. After the first stage finishes, the second stage, shown as the the right part of Fig. 2.1, uses the performance database to recognize the relation between program inputs and the corresponding suitable optimization decisions. G-ADAPT then transforms the original GPU code into a program that is able to automatically adapt to an arbitrary input.

The first part uses empirical search to overcome the difficulty in modeling GPU program performance; the second part addresses the input-sensitivity issue by recognizing the influence of inputs and making GPU program adaptive.
Stage 1: Heuristic-Based Empirical Search and Data Collection

The first stage is an iterative process. The inputs to the process include a given GPU application (with some pragmas inserted) with a set of typical inputs. In the iterative process, the adaptive framework for each of the given inputs to the GPU application automatically searches for the best values of optimization parameters that can maximize the performance of the application. The process results in a performance database, consisting of a set of (input, best parameter values) tuples. Three components are involved in this iterative process. In each iteration of a given input to the GPU program, a compiler produces a new version of the application, a calibrator then measures the performance of the program on the given input, and the measured result is used by an optimization agent to determine what version of the program should be tried in the next iteration. When the system finds the best optimization values for that input, it stores the values into the performance database, and starts the iterations for another input. Several issues need to be addressed to make the empirical search efficient and widely applicable. The issues include how to derive optimization space from the application, how to characterize program inputs, and how to prune the search space to accelerate the search. In the following, we describe how the three components in the first stage of G-ADAPT work together to address these issues.

2.4.2.1 Optimization Pragmas and G-ADAPT Compiler:

We classify the optimization parameters in GPU applications into three categories, corresponding to three different optimization levels. In the first category are execution configurations of the program, that is the number of threads per block and the number
empirical search & data collection

pattern recognition & code generation

empirical search & data collection

pattern recognition & code generation

optimization parameters

optimized GPU code

performance calibrator

performance

optimization agent

perf. DB

G-ADAPT: An adaptive optimization framework for GPU programs.

of thread blocks for the execution of each GPU kernel. The second category includes
the parameters that determine how the compiler transforms the program code, such
as loop unrolling levels and the size of loop tiles. The third category includes other
implementation-level or algorithmic decisions, such as the selection of different algo-

rithms for implementing a function. These parameters together constitute the space for
the empirical search.

Different applications have different parameters to optimize; some parameters may
be implicit in a program, and the ranges of some parameters may be difficult to be
automatically determined because of aliases, pointers, and the entanglement among
program data.

Even though compilers may automatically recognize some parameters in the first two
categories, for automatic search to work, generally, it is necessary to have a mechanism
to easily expose all those kinds of parameters and their possible values for an arbitrary
In this work, we employ a set of pragmas, named G-ADAPT pragmas, to support the synergy between programs and compilers in revealing the optimization space. There are three types of pragmas. The first type is dedicated for the adjustment of scalar variable (or constant) values that control the execution configurations of the GPU application. The second type is for compiler optimizations. The third type is for implementation selection. The pragmas allow the inclusion of search hints, such as the important value ranges of a parameter and the suitable step size. For example, a pragma, "#pragma erange 64,512,2" above the statement "#define BLKSZ 256" means that the search range for the value of BLKSZ is from 64 to 512 with exponential (the first “e” in “erange”) increase with base 2.

We develop a source-to-source compiler, named the G-ADAPT compiler, to construct and explore the optimization space. The G-ADAPT compiler is based on Cetus [49], a C compiler infrastructure developed by the group led by Eigenmann and Midkiff. With some extensions added to Cetus, the G-ADAPT compiler is able to support CUDA programs, the G-ADAPT pragmas, and a set of program transformations (e.g. redundant elimination and various loop transformations).

The G-ADAPT compiler has twofold responsibilities. At the beginning of the empirical search, the compiler recognizes the optimization space through data flow analysis, loop analysis, and analysis on the pragmas in the GPU application. In each iteration of the empirical search, the compiler uses one set of parameter values in the search space to transform the application and produces one version of the application.
2.4.2.2 Performance Calibrator and Optimization Agent:

The performance calibrator invokes the CUDA compiler, NVCC, to produce an executable from the GPU program generated by the G-ADAPT compiler. It then runs the executable (on the current input) to measure the running time. After the run, it computes the occupancy of the executable on the GPU. The occupancy reflects the degree to which the executable exerts the computing power of the GPU. A higher occupancy is often desirable, but does not necessarily suggest higher performance. The occupancy calculation is based on the occupancy calculating spreadsheet [72] provided by NVIDIA. Besides hardware information, the calculation requires the information on the size of shared memory allocated in each thread, the number of registers used by each thread, and the thread block size. The calibrator obtains the information from the "cubin" files of the GPU program and the execution of the executable.

The calibrator then stores the parameter values, along with the running time and occupancy, into the performance database. It determines whether the termination conditions (explained next) for the search on the current input have been reached; if so, it stores the input, along with the best parameter values that have been found, into the performance database.

The responsibility of the optimization agent is to determine which point in the optimization space should be explored in the next iteration of the search process. The size of the optimization space can be very large. For $K$ independent parameters, with $D_i$ denoting the number of possible values of the $i$th parameter, the optimization space is as large as $\prod_{i=1}^{K} D_i$. It implies that for an application with many loops and implementation options, the space may become too large for the framework to enumerate all the
points. The optimization agent uses hill climbing to accelerate the search. Let K be the number of parameters. The search starts with all the parameters having their minimum values. In each of the next K iterations, the search increases one parameter by a step and keeps the others unchanged. After iteration (K + 1), it finds the best of the K parameter vectors that have been tried, and uses it as the base for the next K iterations. This process continues. When one parameter reaches the maximum, it stops increasing. When all parameters reach their maximum values, the search stops. This hill climbing search differs from the model-based prune proposed by Ryoo et al. [92]. Their approach is applicable when the program performance is not bounded by memory bandwidth; the method has shown a more significant prune rate than our approach does. On the other hand, the hill climbing search is generally more applicable, making no assumptions on the GPU application.

2.4.3 Stage 2: Pattern Recognition and Cross-Input Adaptation

After the first stage, the performance database contains a number of (input, best parameter values) tuples, from which the pattern recognizer learns the relation between program inputs and the optimization parameters. A number of statistical learning techniques can be used in the learning process. In this work, we select Regression Trees [35] for its simplicity and good interpretability. Regression Trees is a divide-and-conquer learning approach. It divides the input space into local regions with each region having a regular pattern. In the resulting tree, every non-leaf node contains a question on the input features, and every leaf node corresponds to a region in the input space. The question contained in a non-leaf node is automatically selected in light of entropy reduc-
tion, defined as the increase of the purity of the data set after the data are split by that question. We then apply Least Mean Squares (LMS) to the data that fall into each leaf node to produce the final predictive models.

To capitalize on the learned patterns, we need to integrate them into the GPU application. If there were just-in-time compiler (JIT) support, the integration could happen implicitly during runtime. The JIT compiles the program functions using the parameters predicted as the best for the program input. Without JIT, the integration can occur either through a linker, which links the appropriate versions of object files into an executable before every execution of the application, or an execution wrapper, which selects the appropriate version of executables to run every time. In our experiments, we use the wrapper solution because it has no linking overhead, and the programs in our experiments need only few versions of executables. The G-ADAPT compiler, along with the CUDA compiler, produces one executable for each parameter vector that is considered as the best for some training inputs in the performance database. When the application is launched with an arbitrary input, the version selector in the wrapper uses the constructed regression trees to quickly determine the right executable based on the input and then runs the program.

2.5 Evaluation

We use seven benchmarks to test the effectiveness of the optimization framework, as listed in Table 2.1. Most of the programs are from NVIDIA SDK [72]. The program, mvMul, is a matrix vector multiplication program from Fujimoto [28]. It is an efficient implementation, outperforming the NVIDIA CUBLAS [72] version significantly, thanks
to its adoption of a new algorithm along with an effective use of texture memory [28].

We emphasize that the programs we use have all been manually tuned by the developers. The reduction program, for instance, has gone through seven optimizations, respectively, on the algorithm, locality, branch divergence, loop unrolling, and so on. NVIDIA has used it as a typical example to demonstrate manual optimizations on GPU programs. The sequence of optimizations has accelerated the program by as much as a factor of 30.

The third column of the table shows the number of different inputs we have used for each benchmark. We create these inputs based on our understanding of the applications, with an attempt to cover a wide range of the input space.

The type of GPU we use is NVIDIA GeForce 8800 GT. It contains 512 MB global memory, 14 multiprocessors, 112 cores, with clock rates set at 1.51 GHz. Each multiprocessor has 16 KB shared memory and 8192 registers. Every GPU co-runs with 2 Intel Xeon processors (3.6 GHz) on a machine with SUSE Linux 2.6.22 installed.

The best configurations of three out of the seven programs change with their inputs. For all the programs, G-ADAPT is able to learn the relation between inputs and optimization parameters, producing over 93% prediction accuracy for the best optimization decisions. The prediction yields several times the speedup compared to the running times of the original programs.

2.6 Discussion

In this section, we first present the training overhead of G-ADAPT and then discuss some complexities in applying G-ADAPT for large applications. The two right-most columns
Table 2.1: Benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Num of Inputs</th>
<th>Prediction acc</th>
<th>Training iterations</th>
<th>Training time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>convolution filter of a 2D signal</td>
<td>10</td>
<td>100%</td>
<td>200</td>
<td>2625</td>
</tr>
<tr>
<td>matrixMul</td>
<td>dense matrix multiplication</td>
<td>9</td>
<td>100%</td>
<td>196</td>
<td>2539</td>
</tr>
<tr>
<td>reduction</td>
<td>sum of array</td>
<td>15</td>
<td>93.3%</td>
<td>124</td>
<td>124</td>
</tr>
<tr>
<td>scalarProd</td>
<td>scalar products of vector pairs</td>
<td>15</td>
<td>93.3%</td>
<td>75</td>
<td>29</td>
</tr>
<tr>
<td>transpose</td>
<td>matrix transpose</td>
<td>15</td>
<td>93.3%</td>
<td>124</td>
<td>124</td>
</tr>
<tr>
<td>transpose-co</td>
<td>matrix transpose with coalescing memory references</td>
<td>18</td>
<td>100%</td>
<td>54</td>
<td>1639</td>
</tr>
</tbody>
</table>

in Table 2.1 reveal the training overhead of G-ADAPT on the seven benchmarks. The total number of iterations range from 54 to 200, and the total training time spans from 29 seconds to 47 minutes. The time is determined by the number of training inputs, the dimensions of the search space, and the size of the inputs. The program, convolution, happens to run for a long time on some of its training inputs, resulting in the longest training time. It is worth noting that one complexity, input characterization, happens to be simple in our experiments. Input characterization is to determine the important features of program inputs. In our experiments, the inputs to the programs are just several numbers, indicating the sizes of the input signal, matrix, array, or vector, which naturally capture the important characteristics of the input data sets. However, for large complex GPU applications, the input characterization may need special treatment. One option is to develop some input characterization procedures and link them with G-ADAPT. A recent study [60] proposes an extensible input characterization language, XICL, to ease the efforts. Detailed studies remain to be our future work.

2.7 Conclusion

This chapter reports our exploration of the influence of program inputs on GPU program optimizations. It shows that for some GPU applications, the best optimizations are dif-
different for different inputs. It presents a compiler-based adaptive framework, G-ADAPT, which is able to extract optimization space from program code, and automatically search for the best optimizations for an GPU application on different inputs. With the use of Regression Trees, G-ADAPT produces crossinput predictive models from the search results. The models can predict the best optimizations from the input given to the GPU application, and thus enable cross-input adaptive optimizations. Experiments show significant performance improvement generated by the optimizations, demonstrating the promise of the framework as an automatic tool for resolving the productivity bottleneck in the development of efficient GPU programs.
Chapter 3

A Robust Point-based Registration Method for Brain Shift

3.1 Introduction

The point-based registration (PBR) has a wide range of applications. PBR can be classified into rigid and non-rigid methods. The readers are referred to [59] for an excellent review of this kind of method.

PBR can be defined as finding the mapping function $F$ from a given source point set $S$ and target point set $T$ with/without correspondence $C$. By carefully examining PBR, we classify them into two categories according to the correspondence $C$:

1. Relying on some specific algorithms to find the correspondence $C$ and then find the mapping function $F$ [20, 21, 27]
2. Find the correspondence and mapping function simultaneously [33, 75, 110]

Type 1 heavily relies on the specific algorithm. For instance, to compensate for brain shift, Clatz et al. [20] used block matching to find the correspondence. DeLorenzo et al. [21] used game theory to perform cortical surface tracking to find the correspondence. Ferrant et al. [27] simulated the extracted surface of the ventricle as a membrane to perform the tracking. The dependence on the specific algorithm will constrain the applicability of these methods. Compared to type 1, the methods in type 2 have no requirement for the correspondence and do not rely on specific algorithm to find it, therefore rendering these methods more flexible. However, the solutions of this kind of methods are more difficult to be obtained because two variables: $F$ and $C$, need to be solved.

The Iterative Closest Point (ICP) [38] is a well-known algorithm in type 2. This method utilizes the nearest-neighbor relationship to estimate the correspondence and then refines the transformation based on the correspondence iteratively. ICP is fast and can be guaranteed to converge at the local minimum. However, it uses binary correspondence and only supports rigid mapping. To deal with these drawbacks, Chui et al. presented a Robust Point Matching (RPM) [33] algorithm, which is characterized by the use of softassign for the correspondence and Thin-Plate Splines (TPS) as the non-rigid mapping. This method is presented in the computation visualization field, but now more and more groups employ it for the NRR of medical image.

Miga. et al. employed this method to deal with a challenging problem: NRR for tumor resection [23]. Vessels are identified in both pre-operative MRI and laser range image, and then the RPM is used to force the corresponding vessels to exactly match
each other under the constraint of the bending energy of the whole image. Li et al. [52] employed this method for NRR of longitudinal breast MR image. Papademetris et al. [75] extended RPM to improve its ability to deal with larger point sets and partially correlated point sets. They applied this method to the problem of forming composite activation maps from functional magnetic resonance images, and demonstrated that the superior performance of this method to pure intensity based registration in the specific area of the fusiform gyrus.

RPM uses TPS as the mapping function. The basis function of TPS is a solution of the biharmonic [48], which is not compact support, and therefore it will lead to, in real applications, unrealistic deformation far from the given point set. To clearly illustrate this point, we developed a landmark-based NRR using ITK ThinPlateSplineKernelTransform [45] as the transform. We used this NRR to register iMRI with Blood-Oxygen-Level Dependent (BOLD) image based on the feature points selected by surgeons. The results are shown in Fig. 3.1. It clearly shows that if we want to match the region near the craniotomy, only using the points near the craniotomy (red points in Fig.3.1(a) and Fig. 3.1(b)) will make the region far from these points to be deformed unrealistically, as shown in Fig. 3.1(c). If we use additional feature points (green color points), the deformation will become more reasonable, as shown in Fig. 3.1(d). Note that there are actually more red and green color points distributed in other slices. Only some of them are shown in Fig 3.1.

Other groups also realized this problem for TPS. Yang et al. [110] and Wachowiak et al. [105] provided Compact Support Radial Basis Function (CSRBF) to overcome this difficulty, and Papademetris et al. [75] replaced TPS with Free-Form Deformation
Figure 3.1: TPS and FFD.

(FFD) [93] as the mapping function. In comparison to TPS, FFD and CSRBF affect local deformation, but at the same time they will limit the estimated deformation to be valid only near the point sets. Fig. 3.1(e) shows that the control point $a(4,4)$ of FFD only influences its surrounding $4 \times 4$ region (denoted as a blue box). In other words, if we want to use FFD or CSRBF to estimate the deformation of the entire brain, we need the point set either to be dense or to cover the whole brain. However, in some real applications, only sparse information is available. For instance, to compensate for brain shift in Image-guided Neurosurgery (IGNS), we need to estimate the deformation of the entire brain based on scanned cortical surface [67, 99].

To overcome this difficulty, we need to rely more on a priori knowledge to estimate the deformation given the sparse point sets. We combine a biomechanical model with the RPM framework and take the stress energy as the regularization term. This method is capable of estimating the deformation far from the point sets because the biomechanical model agrees well with the behavior of the brain only with sparse information (boundary condition) available. Furthermore, we extend RPM from dealing with the outliers in one point set to the outliers in both source and target point sets, which means that this method still works despite partially correlated point sets. These extensions will result in difficulty in finding the solution. Unlike a TPS based method, whose analytical solution
is available, we have to use the finite element method to find the numerical solution.

The correspondence between the source point set and the target point set will be viewed as a variable and resolved in an Expectation and Maximization (EM) framework. In the next chapter, this EM framework will be extended to accommodate the missing image data.

This chapter contributes a novel point based NRR characterized by only two given point sets, which can be sparse and even partially correlated, that realistically estimate the mapping function for the whole domain.

3.2 Method

To solve the mapping function and correspondence, the NRR problem is formulated as a functional minimization decomposed into a regularization energy and a similarity energy.

3.2.1 Energy Function

Suppose there are two point sets \( S \) (Source point set) and \( T \) (Target point set) in \( \mathbb{R}^3 \) consisting of points \( s_i, i = 1, 2, .., p \) and \( t_i, i = 1, 2, .., l \), respectively. The functional is constructed as follows:

\[
W(u, C) = \int_\Omega \sigma(u)^t \varepsilon(u) + \lambda \sum_{i=1}^p \|s_i + u(s_i) - \sum_{t_j \in \Omega_R} c_{ij} t_j\|^2
\] (3.1)

The first term is the regularization energy defined by the stress energy of a linear elastic model, and the second term is the similarity energy. \( \lambda \) is used to control the trade-off between these two energies. Using the stress energy as the regularization term will
make the estimation of the mapping function more realistic than other work [33, 75, 110], which use the smoothing measure of TPS or CSRBF as the regularization term.

In the similarity energy, \( \Omega_R \) defines the search range, which is a sphere centered at the source point with radius \( R \). \( c_{ij} \) is the probability with which the point \( s_i \) corresponds with \( t_j \) located in \( \Omega_R \). \( u \) is the unknown displacement field and \( C \) is unknown correspondence matrix with entry \( c_{ij} \). Correspondence matrix \( C \) is similar with that in [33], but we define a range \( \Omega_R \) and only take into account the target points located in \( \Omega_R \). The search range basically makes our NRR act as a multi-resolution registration. As the range reduces, the registration will go from the coarse level to the fine level. \( c_{ij} \) is calculated as equation 3.2. For each source point \( s_i \), assume its potential correspondences are subject to the Gaussian distribution,

\[
c_{ij} = \frac{c'_{ij}}{\sum_{k=1}^{m} c'_{ik}}, \quad c'_{ij} = \frac{1}{R \sqrt{2\pi}} e^{-\frac{(t_j - s_i)^2}{2R^2}}, \forall t_j \in \Omega_R, j = 1 \ldots m
\]

Combining the search range with the Least Trimmed Squares [88] robust regression technique, we can effectively detect the outliers existing in both point sets.

It is difficult to find the analytical solution for equation 3.1. We use the finite element method to discretize the problem by approximating:

\[
u = \sum_{i=0}^{n} N_i U_i
\]

where \( n \) is the number of the vertices of the finite element mesh, \( N \) is the shape function, and \( U \) is the node displacement vector. For simplicity, we define vector \( D \) with entry
\( D_t(c_{ij}) = s_i - \sum_{t_j \in \Omega_R} c_{ij} t_j \) and equation 3.1 can be discretized as:

\[
W(U, \mathcal{C}) = U^T K U + \lambda (H U - D(C))^T (H U - D(C))
\]  \hspace{1cm} \text{(3.4)}

\( K \) is the stiffness matrix of size \( 3n \times 3n \). The building of \( K \) has been well documented in [9]. \( H \) is the linear interpolation matrix of size \( 3p \times 3n \).

Each registration point \( o_k \) with number \( k \) contained in tetrahedron with vertex number \( c_i, i \in [0:3] \) contributes to four \( 3 \times 3 \) submatrices: \( [H]_{kc_0}, [H]_{kc_1}, [H]_{kc_2}, \) and \( [H]_{kc_3} \). \( [H]_{kc_i} \) is defined as: \( [H]_{kc_i} = diag(h_i, h_i, h_i) \). The linear interpolation factor \( h_i \) is calculated as:

\[
\begin{bmatrix}
    h_0 \\
    h_1 \\
    h_2 \\
    h_3
\end{bmatrix} =
\begin{bmatrix}
    v_{c_0}^x & v_{c_1}^x & v_{c_2}^x & v_{c_3}^x \\
    v_{c_0}^y & v_{c_1}^y & v_{c_2}^y & v_{c_3}^y \\
    v_{c_0}^z & v_{c_1}^z & v_{c_2}^z & v_{c_3}^z \\
    1 & 1 & 1 & 1
\end{bmatrix}^{-1}
\begin{bmatrix}
    c_{k_0}^x \\
    c_{k_1}^y \\
    c_{k_2}^y \\
    c_{k_3}^z
\end{bmatrix}
\]  \hspace{1cm} \text{(3.5)}

where \( v_{c_i} \) is the vertex with number \( c_i \).

Similar to [20], the equation 3.4 can be solved by:

\[
\frac{\partial W}{\partial U} = [K + H^T H] U - H^T D(C) = 0 \Rightarrow [K + H^T H] U = H^T D(C)
\]  \hspace{1cm} \text{(3.6)}

Regularization of the solution using the mechanical energy inevitably makes the solution contain an approximation error [20]. We address this problem by iteratively estimating the displacement vector:

\[
F_0 = 0, \quad F_{i-1} = KU_{i-1},
\]

\[
[K + H^T H] U_i = H^T D(C) + F_{i-1}
\]  \hspace{1cm} \text{(3.7)}
$K + H^T H$ is a semi-positive definite matrix, and so we use Conjugate Gradient (CG) \cite{94} to resolve the linear system of equations. This component is computed in parallel, facilitated by PETSc implementation \cite{78}.

This energy function has two unknowns: $U$ and $C$. If one is closer to the real solution, the other is, too. So, Expectation and Maximization is employed to resolve them.

### 3.2.2 Expectation and Maximization

The Expectation and Maximization (EM) algorithm \cite{22} is a general algorithm for maximum-likelihood \cite{41} estimation of the model parameter (unknowns) in the presence of missing or hidden data. To estimate the model parameters, EM proceeds iteratively, and each iteration of the EM algorithm consists of two steps: The E step and the M step. In the E step, the missing data are estimated given the observed data and current estimate of the model parameters. In the M step, the likelihood function is maximized under the assumption that the missing data are known. The estimate of the missing data from the E step are used in lieu of the actual missing data. Convergence is assured since the algorithm is guaranteed to increase the likelihood at each iteration.

The intuition behind EM is the alternation between estimating the unknowns and the missing data. The point based non-rigid registration can be stated as finding the mapping function (unknown) between the source point set and target point set in the absence of the correspondence (missing data). The EM proceeds as follows:

- E step: estimate the correspondence given current estimate of the mapping function according to equation 3.2
• M Step: calculate the mapping function given the correspondence according to equation 3.6

### 3.2.3 Outlier Rejection

We present an outlier detection technique by combining the search range with the Least Trimmed Square (LTS) estimator [88]. The LTS estimator is a robust regression technique tolerant to outliers. Considering a linear regression model for sample \((x_i, y_i)\) with a response variable \(y_i\) and a vector of \(p\) explanatory variables \(x_i\):

\[
y_i = \beta x_i + \epsilon_i, i = 1, \ldots, n.
\]

(3.8)

where \(\beta\) is the coefficient vector and \(\epsilon\) is a random error term.

The LTS estimator is defined as:

\[
\bar{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{h} r_i^2(\beta)
\]

(3.9)

where \(r_1^2 \leq \ldots \leq r_n^2\) are the ordered squared residuals. Equation 3.9 is very similar to the traditional least square equation. The difference is that only \(h\) observations with the smallest squared residuals are used in the summation, thereby allowing the fit to stay away from the outliers. The best robustness properties are achieved when \(h\), termed as trimming constant, is approximately \(n/2\), in which case the breakdown point attains 50\% [88]. To determine the LTS estimator, we need to examine the total of \(\binom{n}{h}\) subsamples. Thus, the computation is very slow if \(n\) is large [87].

In this work, we present an approximation method. This method contains two steps: trial step and outlier rejection step.
Trial step: Using EM algorithm to find the mapping function corresponding to the search range $R$ and then transform the source points. The purpose of this step is not the mapping function due to its bias induced by the outliers, but the detection of the outliers in the next step.

Outlier rejection step: based on the transformed source point set, for each source point, find target points within the search range $R = R \times a$, where $a$ is the annealing parameter and is equal to 0.93 as suggested in [33]. If there are no target points for this source point, mark it as an outlier. Replace the original source point set with this marked source point and estimate the mapping function again. The difference between this modified LTS and the traditional LTS is that we use the search range instead of $h$ to perform outlier rejection, and therefore no need for the ordering of the residuals. For the outliers in the target point set, they do not involve the computation if they are out of the range. Thus, this method can be used to deal with the outliers in both point sets.

Algorithm 3 describes this EM procedures embedded with LTS. It has two loops. The first one is the EM loop for the correspondence and the second one is the approximation to interpolation loop for the increase of the accuracy of the solution.

### 3.3 Experiments on Brain Shift

Brain shift significantly compromises the fidelity of the IGNS. There are a host of literatures addressing this issue [7, 20, 24, 67, 76, 99]. According to the devices used in the Operating Room (OR) to obtain the intra-operative data, the registration methods can be classified into two categories: the surface method and the volume method. The surface method introduces cameras [99] or a laser ranger scanner (LRS) [7, 24, 67] into
Input: $S$: source point set, $T$: target point set, $R$: search range, $a$: annealing parameter, $\lambda$: trade-off parameter, $\epsilon$: tolerance

Output: $U$: displacement vector

1. Initialize $R$, $a$, $\lambda$ and $\epsilon$

2. repeat

3. LTS trial step:

4. E step: Estimate correspondence $C$ according to equation 3.2

5. M step: Solve $U$ according to equation 3.6

6. Transform $S$ based on $U$: $S \leftarrow U(S)$

7. LTS outlier rejection step:

8. $S \leftarrow S - s_i$ if there are no target points in $\Omega_{R \times a}$

9. recalculate $U$ based on outliers rejected $S$

10. $error \leftarrow ||U_i - U_{i-1}||$ between successive iterations

11. $R \leftarrow R \times a$

12. until $error < \epsilon$

13. repeat

14. $F_i \leftarrow KU_i$

15. $U_{i+1} \leftarrow [K + H^TH]^{-1}[H^TD(C) + F_i]$

16. until Convergence

17. Output $U$

Algorithm 3: Point based NRR.

the OR to obtain the deformed cortical surface, and the volume method uses iMR [20], iUS [76] to obtain the deformed volume. Recently, iCT has been employed by Archip et al. [6] and Elhawary et al. [26] to deal with the navigated tumor ablation of the liver. Although Eggers et al. [25] used iCT for image-guided cranial surgery, they only focused on the rigid registration.

For the consideration of the evaluation of our NRR, we concentrate on the application of the NRR technique for the surface method, which is characterized by:

- only sparse intra-operative information available (scanned surface)

- outliers in both point sets

- entire brain deformation needed

To obtain the deformed cortical surface, we introduce LRS of 3D Digital Corporation
into the OR. Table 3.1 indicates its specifications. LRS can be tracked by attaching four tracked balls on the top of it as shown in Fig. 3.2(a).

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Standard Deviation</th>
<th>Depth of Field</th>
<th>Point Set Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>175 microns</td>
<td>±20 microns</td>
<td>300 – 900mm</td>
<td>Up to 1000 × 1000</td>
</tr>
</tbody>
</table>

In our experience, the optimal Depth of Field is about 400mm, which means the optimal distance from the LRS to the exposed cortical surface is 400mm. Fig. 3.2(b) shows the position of the LRS in the OR, and Fig. 3.2(c) shows a laser scanning from the surface. The initial surface is acquired by extracting the surface from the mesh. The deformed surface is acquired by scanning the exposed cortical surface using a LRS. Both the initial surface and the deformed surface are represented by the point sets. To use the point based method presented in this chapter, we first need to transform the two surfaces into the same space.

![Tracked LRS](image1) ![Positioned LRS](image2) ![LRS laser](image3)

**Figure 3.2: LRS.**

### 3.3.1 Register the LRS Space with the Image Space

There are three coordinate spaces related with IGNS and two spaces related with the tracked LRS. The five spaces are:

- Image space: the space defined by the image data
• LRS space: the space defined by the LRS

• Polaris space: the space defined by the Polaris

• Reference frame space: the space defined by the reference frame, which is fixed on the head of the patient and can be tracked by the Polaris

• Tracked tool space: the space defined by the tracked tool, which is fixed on the LRS and can be tracked by the Polaris

We need to transform the deformed cortical surface, which is acquired in the LRS space, into the Image space.

The transformation is shown in Fig. 3.3. First, transform the LRS space to the Tracked tool space based on the calibration procedure. Then, by Polaris, transform the Tracked tool space to the Polaris space, and transform Polaris space to Reference frame space. This is easily done as the Polaris can track both the Reference frame and the Tracked tool. Finally, transform the Reference frame space to the Image space based on the routine PBR procedure in IGNS.

Define $T_{LRS-Img}$ as the transform from the LRS space to the Image space, $T_{LRS-Tool}$ as the transform from the LRS space to the tracked tool space, $T_{Tool-Polaris}$ as the transform from the Tracked tool space to the Polaris space, $T_{Polaris-Ref}$ as the transform from the Polaris space to the Reference frame space, $T_{Ref-Img}$ as the transform from the Reference frame space to the Image space. The transform from the LRS space to the Image space $T_{LRS-Img}$ can be expressed as:

$$T_{LRS-Img} = T_{LRS-Tool} \times T_{Tool-Polaris} \times T_{Polaris-Ref} \times T_{Ref-Img} \quad (3.10)$$
3.3.2 Results

![Diagram of space transformation](image)

**Figure 3.3**: Space transformation.

![Images of scanned surface and NRR](image)

**Figure 3.4**: Point based NRR.

Fig. 3.4(a) is the scanned cortical surface using LRS. Fig. 3.4(b) includes source and target point sets. We put them together to clearly show their relationship and to illustrate what is partially correlated. The source point set, shown as green points, consist of all the nodes of the mesh. The target point set includes two parts shown as red points and white points, respectively. The top red point set comes from 3.4(a), which is transformed into image space by equation 3.10. Note that the LRS image includes both texture and geometric information. In this work, we only use its geometric information, i.e. the position of the point. The bottom white point set consists of fixed nodes.
From Fig 3.4(b), we can see that the correlation only occurs at the top and the bottom. Fig. 3.4(c) shows the deformed mesh generated by the Algorithm 3. The surface of the mesh is deformed to the scanned surface, but the bottom is still fixed as expected. We zoom in a part of the craniotomy in Fig. 3.4(d) to show the agreement between the scanned surface and the deformed mesh surface. Fig. 3.4(e) shows the deformation field.

![Images of MRI scans](image)

Figure 3.5: Qualitative evaluation.

Fig. 3.5 shows the results of the qualitative evaluation of this method. Fig. 3.5(a) is the preoperative MRI, and Fig. 3.5(b) is the deformed preoperative MRI generated by applying the deformation field shown in Fig. 3.4(e) on the preoperative MRI. Fig. 3.5(c) is the intra-operative MRI, which is used to evaluate the accuracy of the NRR method. In Fig. 3.5(d), we superimpose the extracted boundary of the preoperative MRI on the intra-operative MRI to show the discrepancy before the NRR. After NRR, the extracted boundary of the deformed preoperative MRI is superimposed on the intra-operative MRI. As shown in Fig. 3.5(e), the discrepancy, especially in the vicinity of the tumor, is obviously reduced.

Fig. 3.6 shows the quantitative evaluation of the registration method. In Fig. 3.6, five landmarks are chosen in preMRI, deformed preMRI, and iMRI, respectively. The first two are superficial landmarks and the other three are deep landmarks. For each
landmark, the top bar is the corrected error and the bottom bar is the initial error. The average corrected error reaches \(1.164\text{mm}\). In the superficial part, the average corrected error is \(1.205\text{mm}\), and in the deep part it is \(1.137\text{mm}\). The deep part has a higher accuracy regarding the absolute corrected error, but its relative corrected accuracy (34\%) is lower than that in the superficial part (78\%). This is because the selected superficial landmarks approach the cortical surface, which has larger deformation, but most of them can be corrected.

![Figure 3.6: Quantitative evaluation.](image)

3.4 Conclusion

This chapter presents a novel point based NRR by: 1) combining a biomechanical model with the class RPM framework to deal with sparse point sets, and 2) combining Least Trimmed Square with a search range to deal with partially correlated point sets. Compared to [20, 21, 27], this method does not rely on specific algorithms to find the correspondence. Compared to RPM, this method has a looser requirement for the input
due to its support for the sparse and partially correlated point sets. This method can effectively address the following problem: given sparse and even partially correlated point sets, find the mapping function. Brain shift is a typical application with this kind of input and our experiment shows the effectiveness of this method to deal with it. In the next chapter, we will extend this method to deal with a more challenging problem: correcting deformation induced by tumor resection.
Chapter 4

A Robust NEM Registration Method for Tumor Resection

4.1 Introduction

Brain shift severely compromises the fidelity of Image-Guided Neurosurgery. Most studies use a biomechanical model to estimate the brain shift based on sparse intra-operative data after the dura is opened [7, 20, 27, 39, 65, 99]. Very few studies in literature address brain deformation during and after tumor resection. The difficulty originates from the fact that resection creates a cavity, rendering the biomechanical model defined in pre-operative MRI inaccurate due to the existence of the additional part of the model corresponding to the resected region. In our work, the model accuracy will be improved by 1) removing the tetrahedra in the model corresponding to the resected region and 2) building a heterogeneous biomechanical model, which is facilitated by our multi-tissue mesh generation method [54].
In [66], Miga et al. investigated tissue retraction and resection using sparse available OR data and a finite element model. They used a two step method: (1) remove tissue volume by manual deletion of model elements that coincide with the targeted zone and then (2) apply boundary conditions to the new surfaces created during the excision process. It is challenging to determine the cavity because a portion of it will be filled by surrounding tissues [23]. Our method eliminates the manual removal step by treating the resected region as a variable, which is able to be automatically resolved using a Nested Expectation and Maximization (NEM) framework, an extension of traditional EM optimization [22].

Based on the bijective Demons algorithm, Risholm et al. presented a registration framework to handle retraction and resection [84]. They used a level set method to automatically detect resected regions. Also, in [83], they presented an elastic FEM-based registration algorithm and evaluated it on the registration of 2D pre- with intra-operative images, where a superficial tumor had been resected.

Ding et al. [23] presented a semi-automatic method based on postbrain tumor resection and laser range data. Vessels are identified in both pre-operative MRI and laser range image, then the Robust Point Matching (RPM) method [33] is used to force the corresponding vessels to exactly match each other under the constraint of the bending energy of the whole image. As we discussed in last chapter, RPM does not have a compact support, which, in a real application, will lead to unrealistic deformation in the region far away from the matching points. In another words, RPM is not suitable for estimating deformation using sparse intra-operative data.

Periaswamy et al. [77] presented an intensity-based registration with partial data.
Although their work is not directly related with tumor resection, the use of the EM method for solving two interdependent models motivates our Nested EM framework. In this framework, we use the feature point rather than the intensity as the metric. The \textit{Point Correspondence} will be viewed as an additional variable along with the \textit{Deformation Field} and the \textit{Resected Region}. Rather than using the pure geometric transformation characterized by local affinity but global smoothness [77], we use a heterogeneous biomechanical model to realistically simulate the underlying movement of the brain, which extends the homogeneous model in our previous work [56] and improves the accuracy further, as the experiments show.

Although the maximum non-rigid registration error in our previous clinical studies [5] is improved by about four times compared to rigid registration on average, we observe that \textit{the error is most significant in the vicinity of the tumor resected region—the place where high accuracy is needed the most}. In this chapter, we focus on this problem and present a new non-rigid registration method. The experimental results indicate that the registration accuracy in the vicinity of the tumor resection can be improved even further.

4.2 Method

The flowchart presented in Fig. 4.1 describes the context of our method, in which we will focus on the Nested Expectation and Maximization Non-rigid Registration (NEMNRR). Because NEMNRR uses a heterogeneous biomechanical model as the regularization term, our previous work on multi-tissue mesh generation will be briefly described as well.

The brain is automatically extracted from the skull by a Brain Extraction Tool
(BET) [100], and the ventricle is segmented by manually delineating the ventricle boundary. The resulting two tissue (brain and ventricle) multi-label image is fed into a multi-tissue mesher to produce a heterogeneous model in conjunction with specific biomechanical attributes. Edge detection is performed on both pre- and intra-operative MRI to produce a source point set and a target point set. Classic Canny edge detection [42], facilitated by an open source tool ITK [45], is employed to produce these two point sets.

In this work, we target the specific feature point-based non-rigid registration problem, which can be stated as:

\[ \text{P1: Given a heterogeneous patient-specific brain model, a source point set in pre-operative MRI and a target point set in intra-operative MRI, find Point Correspondence, Deformation Field and Resected Region.} \]

The three variables are simultaneously resolved by a Nested Expectation and Maximization Non-rigid Registration method, with a biomechanical model and two point sets as inputs. The resolved deformation field can be used to warp the segmented pre-operative MRI to improve the navigation accuracy.

To resolve this problem, the deformation field is represented by a displacement vector defined on the mesh nodes, the correspondence between two point sets is represented by a correspondence matrix, and the resected region is represented by a connected submesh. All three variables are incorporated into one cost function, which is minimized by a Nested EM strategy.

Unlike a traditional Point-based Registration (PBR) method, this Nested EM method does not require the correspondence to be known in advance and allows the
input data to be incomplete, and therefore this method can be viewed as a generalized PBR method. Moreover, to improve the accuracy, a heterogeneous biomechanical model is employed to realistically simulate the underlying movement of the brain. This heterogeneous model is built upon a multi-tissue mesh and specific biomechanical attributes of each tissue.

Figure 4.1: The context of the Nested Expectation and Maximization Non-rigid Registration (NEMNRR). The red boxes represent the new technologies we present in this chapter and the gray boxes represent the existing technologies.
4.2.1 3D Multi-tissue Tetrahedral Mesh Generation

In [54], we presented a multi-tissue mesh generation method, which will be briefly described in this section.

The new multi-tissue mesher consists of two steps: 1) start from a homogeneous Body-Centered Cubic (BCC) [14, 68] mesh to identify a coarse multi-tissue mesh by assigning each tetrahedron with a specific tissue label and 2) deform the coarse multi-tissue mesh surfaces to tissue boundaries defined in the multi-label image.

4.2.1.1 Coarse Multi-tissue Mesh

A BCC mesh is an actual crystal structure ubiquitous in nature. The nodes of BCC are grid points of two interlaced grids. The edges of BCC consist of edges of the grid and additional edges between a node and its eight nearest neighbors in the other grid. An advantage of the BCC is that it is highly structured and easily refined during the simulation even after red-green subdivision [68].

Label redistribution is performed on the homogeneous BCC mesh to produce a coarse multi-tissue mesh, which will be deformed subsequently. Given an initial label assignment (Fig. 4.2(a)), labels are redistributed to produce a surface robust against deformation (see blue line shown in Fig. 4.2(b)). If the surface is not close enough to the tissue boundary (dashed line in Fig. 4.2(b)), red-green subdivision will be performed on the tetrahedra across the tissue boundary as shown in Fig. 4.2(c). The subdivision probably impairs the robustness of the surface, and, therefore, label redistribution is performed again to produce a surface that is robust and better approximates the tissue boundary (see Fig. 4.2(d)). The above procedure is repeated until the multi-tissue surface is well...
posed for deformation and close enough to the tissue boundary.

**Figure 4.2:** Coarse multi-tissue mesh generation. (a) L1 and L2 are tissue labels, the dash line is the real boundary and the blue line is the submesh interface. (b) Redistribute labels. (c) Subdivide if not satisfy the resolution criterion (e). (d) Redistribute labels again.

4.2.1.2 Deform the Mesh Surface To the Tissue Boundary

Mesh fidelity is achieved by iteratively deforming the mesh surface toward the tissue boundary. In each iteration, the deformation field, the one and the only variable, is resolved by minimizing the function:

\[
W(U) = \sum_{i=1}^{n} (U^T K_i U + \lambda_i (H_i U - D_i))^T (H_i U - D_i)),
\]

(4.1)

where \(n\) is the number of the tissues, and \(K_i\) is the global stiffness matrix assembled by the tetrahedra within \(i\)-th tissue. \(K_i\) depends on two biomechanical attributes of \(i\)-th tissue: Young's modulus and Poisson's ratio. The building of \(K_i\) has been well documented in [9]. \(H_i\) is the global linear interpolation matrix assembled by mesh nodes. The assembling of \(H_i\) will be presented in next section. \(D_i\) is the distance vector from the \(i\)-th surface to the \(i\)-th tissue boundary. \(\lambda_i\) is used to balance the quality (first term) and the fidelity (second term). Compared with equation (3.4), equation (4.1)
extends it to a heterogeneous model. Moreover, $D$ is not a variable like equation (3.4), but can be calculated directly.

We minimize $W(U)$ by solving:

$$\frac{\partial W}{\partial U} = 0 \Rightarrow \sum_{i=1}^{n} (K_i + \lambda_i H_i^T H_i)U = \sum_{i=1}^{n} \lambda_i H_i^T D_i \quad (4.2)$$

### 4.2.2 Nested Expectation and Maximization Non-rigid Registration (NEMNRR)

In this section, we first develop the cost function step by step from the classic point-based non-rigid registration energy function, then present a Nested Expectation and Maximization framework to resolve it.

#### 4.2.2.1 Cost Function

Given $S = \{s_i\}_{i=1}^{N} \in \mathbb{R}^3$ and $T = \{t_i\}_{i=1}^{L} \in \mathbb{R}^3$, a Source and Target point set, respectively, the point-based non-rigid registration problem can be formulated as:

$$\overline{u} = \arg\min_u \left( \int_\Omega R(u)d\Omega + \lambda \sum_{s_i \in \Omega} \|s_i + u(s_i) - t_i\|^2 \right), \quad (4.3)$$

where the first term is regularization or smoothing energy and the second term is similarity energy. $u$ is the deformation field and $\lambda$ controls the trade-off between these two energies. $\Omega$ is the problem domain, namely the segmented brain.

Brain tissue removal influences both terms in equation (4.3): (i) the regularization in terms of the domain on which it is defined, and (ii) the similarity in terms of additional outliers introduced due to tumor resection. We extend equation (4.3) to equation (4.4)
by specifying the regularization term with the strain energy of a linear elastic model:

\[
(\bar{u}, \bar{c}_j, \Omega') = \underset{\bar{u}, \bar{c}_j, \Omega'}{\text{argmin}} \left( \int_{\Omega - \Omega'} \sigma(u)^t \epsilon(u) d(\Omega - \Omega') + \lambda \sum_{s_j \in \Omega - \Omega'} \| \tilde{s}_j + u(s_j) - \sum_{t_j \in \Omega_R} c_{ij} t_j \|^2 \right), \quad (4.4)
\]

where variable \( \Omega' \) represents the resection region. \( c_{ij} \) and \( \Omega_R \) follow the same definition in equation (3.1). The homogeneous model employed in the regularization term in equation (4.4) is further extended to the following heterogeneous model:

\[
(\bar{u}, \bar{c}_j, \Omega') = \underset{\bar{u}, \bar{c}_j, \Omega'}{\text{argmin}} \left( \sum_{\Omega_i \in \Omega - \Omega'} \int_{\Omega_i} \sigma_i(u)^t \epsilon_i(u) d(\Omega_i) + \lambda \sum_{s_j \in \Omega - \Omega'} \| \tilde{s}_j + u(s_j) - \sum_{t_j \in \Omega_R} c_{ij} t_j \|^2 \right), \quad (4.5)
\]

where \( \cup \Omega_i = \Omega - \Omega', i = 1 \ldots n \).

**Remark** If \( n = 1, \Omega' = \emptyset \), and \( c_{ij} = 1 \), then equation (4.5) is reduced to equation (4.3). This means our method can be viewed as a generalized point-based NRR method characterized by 1) employing a heterogeneous biomechanical model as the regularization, 2) accommodating incomplete data, and 3) without correspondence requirement.

Equation (4.5) is approximated by equation (4.6) using finite element method:

\[
W(U, C, M') = \sum_{i=1}^n U^T K_i U + \lambda \|(HU - D(C))\|^2,
\]

where \( \int_{\Omega_i} \sigma_i(u)^t \epsilon_i(u) d(\Omega_i) \) is approximated by \( U^T K_i U \) as in [9]. \( C \) is the correspondence matrix with entries \( c_{ij} \). The entries of the vector \( D \) are defined as: \( d_i(c_{ij}) = s_i - \sum_{t_j \in \Omega_R} c_{ij} t_j, \forall s_i \in M \setminus M' \), where \( M \) is the non-resected mesh that approximates \( \Omega \) and \( M' \) is the resected mesh that approximates \( \Omega' \). The first term is the strain energy assembled on all elements in \( M \setminus M' \) and the second term is similarity energy defined on all source points \( s_i \in M \setminus M' \).
$H_i$ is the global linear interpolation matrix assembled by registration points (source points). The assembling of $H_i$ based on a general registration points $o_k$ is given in equation (3.5). In mesh deformation, because we use the mesh nodes as registration points (i.e. $o_k$ is the same with one of the four tetrahedron nodes), equation (3.5) is reduced to:

$$h_i = \begin{cases} 
1 & \text{for } o_k = v_c, \\
0 & \text{for } o_k \neq v_c
\end{cases} \quad (4.7)$$

**Remark** Compared with equation (4.6), equation (4.1) has only one variable $U$, and, therefore, can be resolved directly by equation (4.2). The additional two variables in equation (4.6) result in the difficulty of resolving $U$.

Finding $C$ and $M$ is equivalent to outlier rejection. In Fig. 4.3, we use red triangles and rectangles to denote point outliers. The original definition of outliers is extended to include elements in addition to points, and the resected region $M'$ can therefore be viewed as a collection of element outliers. As a result, the problem of resolving the three unknowns (Correspondence, Deformation Field and Resected Region) is transformed into the problem:

**P2:** Given a heterogeneous patient-specific brain model, a source point set in pre-operative MRI and a target point set in intra-operative MRI, reject point outliers from both Source and Target point sets and reject element outliers from the biomechanical model.

We developed a Nested Expectation and Maximization Solver to iteratively reject point and element outliers.
4.2.2.2 Nested Expectation and Maximization Solver

The Expectation and Maximization (EM) algorithm [22] is a general algorithm for maximum-likelihood [41] estimation of the model parameter (unknowns) in the presence of missing or hidden data. Let us reconsider the registration problem in the EM context. The cost function (4.6), from probability (Bayesian) point of view, defines the likelihood function, in which the unknown (model parameters) is the deformation vector $U$, and the missing data are the correspondence $C$ and the resected region $M'$. Assuming $M'$ is known, the more accurate the estimate of $C$ is, the more accurate the estimate of $U$ is and vice versa. Therefore, EM algorithm is employed to solve $U$ and $C$ under a specified $M'$. To resolve $M'$, we treat $U$ and $C$ as one unknown pair $<U,C>$. The more accurate the estimate of $M'$, the more accurate the estimate of $<U,C>$, and the missing data $M'$ can therefore be resolved by another high level EM.

The basic procedure of EM is to alternate between estimating the unknowns and the missing data. In the Nested EM framework shown in Fig. 4.4, the inner EM is used to resolve $U$ and $C$ with $M'$ fixed, and the outer EM is used to resolve $M'$. $M'$ is approximated as a collection of tetrahedra located in a region, corresponding to the tumor cavity in the intra-operative MRI, in the model. $M'$ is initialized as empty and
updated in each iteration of the outer EM. If all the tetrahedra contained in the cavity are collected, the outer EM stops.

**Figure 4.4:** Nested Expectation and Maximization framework.

**Inner EM** Inner EM is used to resolve $U$ and $C$ given $M'$. For a source point $s_i$, its probability corresponding with a target point $t_j$ can be estimated (E step) by equation (3.2).

Once $C$ is estimated, $U$ can be resolved using equation (4.2). The resolved $U$ is used to warp $S$ closer to $T$ and then the correspondence $C$ is estimated again. We illustrate this inner EM in Fig. 4.5, in which the inner EM iterates along a horizontal direction and the outer EM iterates along a vertical direction. In Fig. 4.5, we use subscript $i$ to denote the inner EM, and subscript $o$ to denote outer EM. The superscript is used to denote the iteration number. For example, $E^{k}_i$ denotes the $k$-th iteration of E step in inner EM. In the horizontal direction, inner EM iteratively estimates the correspondence and deformation field until no point outliers are detected. Inner EM begins from a search range (green circle) with a larger radius $R$. For each source point, if there are no target points located in the circle centered at the source point, this source point will be rejected as an outlier. Each target point outside of the search range will be rejected as an outlier.
too. Once all the outliers are rejected, $C$ can be estimated by equation (3.2), and $U$ can be solved by equation (4.2). In the next iteration, $R$ is reduced by multiplying a simulated annealing factor 0.93, which is presented in [33], and the above procedures are repeated. Its pseudo code is presented in Algorithm 4.

![Nested EM diagram](image)

**Figure 4.5:** Illustration of Nested Expectation and Maximization. Row: inner EM, Column: outer EM.

**Outer EM** Outer EM, illustrated in the vertical direction in Fig. 4.5, is used to find $M'$ and $< U, C >$. In M step, $< U, C >$ is resolved by inner EM. In E step, $M'$ is resolved by an element outlier rejection algorithm. $M'$ is approximated by a collection of tetrahedron outliers, which fall in the cavity in the intra-operative MRI. The cavity does not need to be identified in the intra-operative MRI, and it is in fact impossible to distinguish the cavity from the background. The region $BGI$, including the cavity and the background, can be very easily segmented by a simple threshold segmentation method. However, we cannot determine if a tetrahedron is an outlier by simply examining whether or not it
\[ [U, C] = \text{PointOutlierRejection}(M, M', S, T) \]

**Input:** \( M \): non-resected mesh, \( M' \): resected mesh, 
\( S \): source points, \( T \): target points

**Output:** \( U \): displacement vector, \( C \): correspondence matrix

1. Initialize \( R \) and Tolerance \( \epsilon \)
2. \( U \leftarrow I \)
3. repeat
4. Transform \( S \) based on \( U \): \( S \leftarrow U(S) \)
5. \( \text{E step:} \)
6. \( S \leftarrow S \setminus \{ s_i \} \) if there are no target points in \( \Omega_R \) for \( s_i \) \}
7. \( \text{// outlier rejection for } T \)
8. Estimate correspondence \( C \) according to equation (3.2)
9. \( \text{M step:} \)
10. Solve \( U \) according to equation (4.2)
11. Decrease \( R \): \( R \leftarrow R \times 0.93 \)
12. until \( \text{error} < \epsilon \)

**Algorithm 4:** Feature point outlier rejection.

\[ [M', S] = \text{EleOutlierRejection}(M, M', U, BGI) \]

**Input:** \( M \): non-resected mesh, \( M' \): resected mesh, 
\( U \): displacement vector, \( BGI \): background image

**Output:** \( M' \): new resected mesh, \( S \): new source points

1. Obtain deformed remaining mesh \( DM \leftarrow U(M \setminus M') \)
2. Find all elements \( M_1 \) completely contained in the background image \( BGI \) and constitute the largest connected mesh with \( M' \)
3. Map \( M_1 \) in \( DM \) to \( M_2 \) in \( M \setminus M' \)
4. \( S \leftarrow S \setminus \{ s_i \} s_i \in M_2 \)
5. \( M' \leftarrow M' \cup M_2 \)
6. Scale Young’s modulus for the elements across the boundary

**Algorithm 5:** Element outlier rejection.

is located in the \( BGI \) because some tetrahedra might happen to fall in the background instead of the cavity. To make the element outlier rejection algorithm robust, we utilize the fact that the resected region is a collection of tetrahedra, which falls in the \( BGI \) of intra-operative MRI, connect with each other and constitute a maximal connected submesh, which can effectively occlude false outliers. The collection of the outliers proceeds iteratively, and in each iteration, more specifically the E step of outer EM, additional outliers will be added into \( M' \) if they fall in the background \( BGI \) and connect with the maximal connected submesh identified in the previous iteration.
The element outlier rejection algorithm is presented in Algorithm 5. Assuming the current deformation field and resected region is \( U \) and \( M' \), respectively, the new estimated outliers \( M'' \) can be obtained by transforming the remaining mesh \( M \setminus M' \) using \( U \) and then finding all elements that satisfy the requirements: (1) are completely inside the background of the intra-operative image, and (2) are connected with \( M' \).

The outer EM iteratively rejects element outliers using Algorithm 5 and computes \( <U,C> \) using Algorithm 4 until no additional element outliers are detected. The whole pseudo code of the Nested EM algorithm is presented in Algorithm 6.

\[
U = \text{NEMNRR}(MRI, iMRI)
\]

**Input:** MRI: pre-operative MRI, iMRI: intra-operative MRI

**Output:** U: displacement vector

1. Segment brain on MRI and mesh generation for \( M \)
2. Segment background image BGI on iMRI
3. Canny edge detection on MRI to get \( S \)
4. Canny edge detection on iMRI to get \( T \)
5. Initiate \( M' \leftarrow \emptyset \)
6. repeat
7. M step: \( U, C \leftarrow \text{PtOutlierRejection}(M, M', S, T) \)
8. E step: \( M', S \leftarrow \text{EleOutlierRejection}(M, M', U, BGI) \)
9. until \( M' \) does not change

**Algorithm 6:** Nested Expectation and Maximization NRR (NEMNRR).

### 4.3 Results

We conducted experiments on synthetic data and clinical data including low- and high-field MRI. The experiments on low-field MRI represent a typical application, namely using sparse intra-operative information to correct pre-operative MRI. The experiments on high-filed MRI represent another typical application, namely fusing pre-operatively acquired BOLD, not available in the OR, with intra-operative MRI. In addition to the two applications, in this section, we also conduct experiments to compare our method...
with a classic point-based NRR, and compare the homogeneous model with the heterogeneous model.

4.3.1 Experiments on Synthetic Data

To generate a synthetic resected brain, we developed a surgery simulation tool to simulate brain resection as shown in Fig. 4.6(a). The synthetic deformed resected brain is produced by our surface-based registration tool [55, 114], which is capable of deforming the brain based on specific boundary condition: the deformation of the resection surface. The source points $S$ are simulated as the surface nodes of the resected region (green points in Fig. 4.6(b)) before deformation and the target points $T$ (white points in Fig. 4.6(b)) are the surface nodes of the resected region after deformation. All the surface nodes except the green ones are added into $S$ as the outliers (white points in Fig. 4.6(c)). The outliers for $T$, are generated using white Gaussian noise (green points in Fig. 4.6(d)).

Fig. 4.6(e) and Fig. 4.6(f) show that all the source points and target points are correctly detected by Algorithm 4. Most outliers are rejected from $S$ and $T$ except three outliers in $S$ (white points in Fig. 4.6(e)) and the corresponding three outliers in $T$ (white points in Fig. 4.6(f)). Fig. 4.6(g) shows the element outlier removed mesh $M \setminus M'$ produced by Algorithm 5. We intentionally put the non-resected mesh $M$ and the resected mesh $M \setminus M'$ together to show the resected region clearly.

We conduct two experiments to verify our hypothesis which is the removal of element outliers from the model can improve the accuracy of the registration. Both experiments register the non-resected brain with the synthetic resected brain, but one experiment
rejects element outliers in the model and the other does not. In both experiments, we use the same source points and target points as shown in Fig. 4.6(b) so that the variation of the results is uniquely caused by whether outliers are rejected or not. In each experiment, the registration result is compared with the synthetic deformed resected brain (true answer) by subtracting one from the other to produce a discrepancy image. If the registration result is closer to the true answer, the discrepancy should be smoother. Comparing Fig 4.6(h) with Fig 4.6(i), the method involving element outlier rejection demonstrates more accurate result. This experiment validates our hypothesis.

(a) Surgery simulation. (b) Source (green) and (c) Source points with target Points (white). outliers (white).

(d) Target points with noises (green). (e) Estimated source (f) Estimated target points.

(g) Non-resected mesh (h) Discrepancy between non-resection mesh and (i) Discrepancy between resection mesh and true answer.

Figure 4.6: Results from the synthetic data.
4.3.2 Experiments on Clinical MRI

We conducted experiments on intra-operative MRI including low- and high-field MRI. The data comes from the Center of Neurosurgery of Huashan Hospital (Fig. 4.7). Both low- and high-field MRI are registered with pre-operative MRI. The pre-operative MRI and high-field intra-operative MRI were acquired in 8 minutes with the same protocol: pre-operative (high filed intra-operative) MRI (IMRISneuro, IMRIS, Canada), 3D T1-weighted magnetization-prepared rapid gradient echo (MPRAGE) sagittal images with 

\[ \text{resolution} = 256 \times 256, \text{slices} = 176, \text{thickness} = 1.0\text{mm}, \text{FOV} = 256 \times 256. \]

![Image of MRI system](image_url)

**Figure 4.7:** The 3.0 T magnet system (Signa SP, Siemens Medical Systems) of the Neurological Department of Huashan Hospital, FDU, Shanghai, China.

4.3.2.1 Low-Field Intra-operative MRI

Low-field MRI (PoleStar N20, Medtronic, USA), 3D T1-weighted three-dimensional fast spoiled gradient recalled sequence with 

\[ \text{resolution} = 128 \times 128 \text{ matrix}, \text{slices} = 35, \text{thickness} = 4\text{mm}, \text{FOV}160 \times 120 \]

was acquired in 7 minutes. Fig. 4.8(a) is pre-operative MRI and Fig. 4.8(b) is a Region of Interest (ROI) in the intra-operative MRI. Unlike high-field MR, low-field MR is incapable of capturing the whole brain. Fig. 4.8(c) depicts the discrepancy before NEMNRR registration. Specifically, it shows the boundary of (b) superimposed on (a) after rigid registration but before non-rigid registration.
The extracted boundary points on pre- (Fig. 4.8(a)) and intra-operative MRI (Fig. 4.8(c)) will be used as the source points and target points, respectively. Both source points and target points are obtained by ITK implementation of the canny edge detection algorithm. After NEMNRR (see Fig. 4.8(d)), we observe: (i) surface 1 is not deformed to outlier 1', which is not a real brain surface, but the boundary of the ROI, (ii) surface 2 still agrees well with the real boundary as it does before registration, although there are many outliers 2' around it, (iii) surface 3 is correctly deformed to 3', (iv) however, surface 8 is not affected by 8' due to outlier rejection, and (v) the cavity 4 shows the resected region, and 5 shows the remaining tumor. The comparison of another slice (92) is shown in Fig. 4.8(e) and Fig. 4.8(f). After registration, the surface of the ventricle in the deformed pre-operative image matches well with that in the intra-operative image. Surface 7 in the vicinity of the resection is also correctly deformed to 7'.

In addition, Fig. 4.8(g), Fig. 4.8(h) and Fig. 4.8(i) are the results using the method (BMNRR) presented in [20]. This method uses Block Matching [10] to find the correspondence and then drives a homogeneous biomechanical model to estimate the deformation. Comparing Fig. 4.8(g) and Fig. 4.8(i) with Fig. 4.8(d) and Fig. 4.8(f), respectively (same slice as the number denotes), we can see the larger deformation in the vicinity of the tumor still exists after BMNRR, which is caused by two reasons: (1) Block Matching cannot find correct correspondence near the resected region, and (2) the resection region is not removed from the biomechanical model. BMNRR shows results in the deep part of the brain as good as our method (see the ventricle in Fig. 4.8(h) and Fig. 4.8(i)) due to fewer outliers and rich texture information, which are helpful for Block Matching.
Figure 4.8: Results of NEMNRR and the comparison between NEMNRR and BMNRR.

4.3.2.2 High-Filed Intra-operative MRI

Another typical application of this registration method is to merge pre-operatively acquired functional MRI (fMRI) such as BOLD with intra-operative MRI. Unlike other fMRI, such as DTI, BOLD cannot be acquired intra-operatively, which makes a non-rigid registration method the only feasible way to bring BOLD into the Operating Room.

Blood-Oxygen-Level Dependent (BOLD) is a type of specialized MRI scan [79]. It measures the hemodynamic response (change in blood flow) related to neural activity in the brain or spinal cord of humans or other animals. It is one of the most recently developed forms of neuroimaging. Since the early 1990s, it has come to dominate the brain mapping field due to its relatively low invasiveness, absence of radiation exposure,
and relatively wide availability.

BOLD naturally agrees with a reference pre-operative MRI, which is used to register with high-field intra-operative MRI to recover the deformation between them. The recovered deformation will be applied on BOLD to merge it with intra-operative MRI. BOLD was acquired in about 20 minutes with the following protocol: gradient-echo EPI sequence [$TR = 2000ms; TE = 30ms; flip = 90; bandwidth = 100kHz, matrix = 64 \times 64, FOV = 256 \times 256, slices = 20; slice thickness = 4mm, gap = 1mm$].

![BOLD and reference](image1.png) ![High-field intraMRI](image2.png)

Figure 4.9: BOLD and high-field intra-operative MRI.

Fig. 4.9(a) is a BOLD image superimposed on a reference MRI, and Fig. 4.9(b) is a high-field intra-operative MRI. Fig. 4.10 shows the rejected element outliers using Algorithm 5. The quality of the remaining mesh after deformation (maximal deformation magnitude 18.2mm) is still acceptable as shown in Fig. 4.10, in which the minimal dihedral angle is 0.212. The reason is our multi-tissue mesher is very robust against larger deformation [54].

Fig. 4.11 shows the deformation field of the heterogeneous model. Part of the brain is cut off to expose the ventricle and its deformation field. The largest deformation reaches 18.2mm, still in the effective range of the linear elastic biomechanical model.
The larger deformation occurs in the region near the resection, and the ventricle on the tumor side is squeezed inward as the arrows show.

Figure 4.11: Deformation field. The color denotes deformation magnitude and the arrow points to deformation direction. Part of the brain, not including ventricles, is removed to display the deformation field of ventricles.

Fig. 4.12 shows the BOLD reference MRI, the high filed intra-operative MRI, and the corrected reference MRI.

The deformation between the BOLD reference MRI and intra-operative MRI is applied on the BOLD to produce a corrected BOLD. In Fig. 4.13, we merge the corrected
BOLD and intra-operative MRI into one image, which can be used for function-guided neurosurgery navigation.

Remark To the best of our knowledge, there is no a feasible way to directly evaluate the accuracy of the corrected BOLD so far. Resting state fMRI [98] provides the means to intra-operatively acquire fMRI, which probably will become the gold standard in the future, but for now this technique is still under investigation. Nevertheless, the accuracy of the corrected fMRI can be indirectly evaluated by measuring the accuracy of corrected anatomical MRI based on the fact that function moves with anatomy in the same way, which has been widely accepted within neurosurgery community.

To quantitatively evaluate the registration results, seven anatomical points as shown in Fig. 4.14 are selected on pre-operative, intra-operative, and corrected pre-operative MRI, respectively. We want to evaluate the accuracy of the method in different parts of the brain, so we intentionally select feature points distributed in the superficial and deep parts, including non-rigid and rigid points.

The error before registration is measured by the magnitude of the displacement between two points located in pre- and intra-operative MRI, respectively. The error after registration is measured by two points located in corrected pre- and intra-operative...
Figure 4.13: BOLD deformation. The first two rows correspond to two different slices. Left image in the first row: before correction. Right image in the first row: corrected BOLD. The last row: superimpose uncorrected BOLD on the corrected BOLD to show the deformation.

MRI, respectively. In these seven anatomical points, the left and right anterior horns of the lateral ventricle (LAH, RAH) are selected to evaluate the accuracy in the deep brain. Three right lateral fissure points (RLF1-3) are selected to evaluate the accuracy in the vicinity of the resected region. Two rigid points basilar sulcus (BS) and anterior commissure (AC) are selected to evaluate whether this method impairs the accuracy of these points, used for the rigid registration.

We conducted experiments on two low-field MRI and two high-field MRI. The results are presented in Fig. 4.15, which shows that after NEMNRR, the accuracy in the region near the resection is improved significantly. The accuracy of the rigid registration points is affected slightly. BMNRR shows good results in the deep part, but not in the vicinity of the tumor.
Figure 4.14: Seven anatomical feature points. Superficial points: Three right lateral fissure points (RLF1-3), Deep points: left and right anterior horns of the lateral ventricle (LAH, RAH), and two Rigid points: basilar sulcus (BS) and anterior commissure (AC).

The relative accuracy improvement is defined as the ratio between the corrected error (error before registration - error after registration) and the error before registration. We summarize the results in Table 4.1 and Table 4.2.

Table 4.1: Relative accuracy improvement $[BMNRR, NEMNRR]$ (%) for superficial points. Negative value means the error is not reduced but increased.

<table>
<thead>
<tr>
<th>Superficial points</th>
<th>RLF1</th>
<th>RLF2</th>
<th>RLF3</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1</td>
<td>[-44.2, 90.7]</td>
<td>[-14.9, 81.9]</td>
<td>[11.5, 92.0]</td>
</tr>
<tr>
<td>case2</td>
<td>[14.0, 80.1]</td>
<td>[-7.6, 81.2]</td>
<td>[4.6, 95.0]</td>
</tr>
<tr>
<td>case3</td>
<td>[56.8, 72.8]</td>
<td>[64.9, 72.2]</td>
<td>[38.0, 72.3]</td>
</tr>
<tr>
<td>case4</td>
<td>[13.1, 84.7]</td>
<td>[13.5, 86.3]</td>
<td>[42.6, 88.6]</td>
</tr>
</tbody>
</table>

4.3.2.3 Heterogeneous Model VS. Homogeneous Model

The single tissue mesh and multi-tissue mesh are shown in Fig. 4.16. To specifically measure the influence of the model on the registration, we employ the multi-tissue mesh in both models. As a result, the influence of the discrepancy of the geometry and
Figure 4.15: Quantitative evaluation of the accuracy using four cases including two low-field MRI (case1 and case2) and two high-field MRI (case3 and case4).

Table 4.2: Relative accuracy improvement \([BMNRR,NEMNRR]\) (%) for deep and rigid points. Negative value means the error is not reduced but increased.

<table>
<thead>
<tr>
<th></th>
<th>Deep points</th>
<th>Rigid points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAH</td>
<td>RAH</td>
</tr>
<tr>
<td>case1</td>
<td>59.1, 74.3</td>
<td>58.0, 40.5</td>
</tr>
<tr>
<td>case2</td>
<td>59.0, 59.0</td>
<td>37.0, 77.7</td>
</tr>
<tr>
<td>case3</td>
<td>53.0, 53.0</td>
<td>37.1, 55.4</td>
</tr>
<tr>
<td>case4</td>
<td>62.0, 75.4</td>
<td>74.2, 77.7</td>
</tr>
</tbody>
</table>

topology between two meshes can be eliminated. The only difference between the two models is the biomechanical parameters. The homogeneous model uses Young's modulus \(E = 3000\text{Pa}\), and Poisson's ratio \(\nu = 0.45\) for all tetrahedra while the heterogeneous model replaces Young's modulus with \(E = 10\text{Pa}\) and Poisson's ratio with \(\nu = 0.1\) for the ventricle [109].

We pay more attention to the ventricle on the tumor side, since it is near the resected region, and therefore more likely to be influenced by the resection. We extract the ventricles corrected by the homogeneous and the heterogeneous models, respectively,
and then superimpose them on the intra-operative MRI (shown in Fig. 4.17(a) and Fig. 4.17(b)). The ventricle from the heterogeneous model matches the ventricle of the intra-operative better than the homogeneous model.

To quantitatively evaluate the result, four statistical measures are computed to give a comprehensive quantification: Dice ratio (the ventricle overlap ratio); false positive (the percentage area of the ventricle labeled in corrected pre-operative MRI but not labeled in intra-operative MRI); false negative (the percentage area of the ventricle not labeled in corrected pre-operative MRI but labeled in intra-operative MRI); ABE (the
average ventricle boundary errors) defined as:

\[
\bar{d} = \frac{\sum ||s_i - t_i||}{|S|},
\]

where \(s_i\) is an edge point of the ventricle in the intra-operative MRI, \(t_i\) is its closest edge point in corrected pre-operative MRI, and \(S\) is the set of \(s_i\).

Table 4.3 summarizes the results for five slices, showing the improvement of the accuracy using the heterogeneous model.

Table 4.3: Quantitative comparison between homogeneous model and heterogeneous model for five slices. [homogeneous, heterogeneous]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dice ratio</td>
<td>0.79, 0.85</td>
<td>0.81, 0.85</td>
<td>0.72, 0.75</td>
<td>0.77, 0.81</td>
<td>0.69, 0.79</td>
</tr>
<tr>
<td>False positive</td>
<td>0.42, 0.37</td>
<td>0.38, 0.16</td>
<td>0.34, 0.33</td>
<td>0.17, 0.13</td>
<td>0.19, 0.11</td>
</tr>
<tr>
<td>False negative</td>
<td>0.21, 0.15</td>
<td>0.19, 0.15</td>
<td>0.28, 0.25</td>
<td>0.23, 0.19</td>
<td>0.31, 0.21</td>
</tr>
<tr>
<td>ABE (in mm)</td>
<td>3.03, 2.68</td>
<td>2.98, 2.61</td>
<td>2.99, 2.78</td>
<td>2.12, 0.17</td>
<td>2.81, 1.77</td>
</tr>
</tbody>
</table>

4.4 Conclusion

We present a novel non-rigid registration method to compensate for brain deformation induced by tumor resection. This method does not require the point correspondence to be known in advance and allows the input data to be incomplete, so it can be considered as a more general point-based NRR. This method uses the strain energy of the biomechanical model to regularize the solution. To improve the fidelity of the simulation of the underlying deformation field, we build a heterogeneous model based on a multi-tissue mesher. To resolve the deformation field with unknown correspondence and resected region, we develop a Nested EM framework, which can effectively resolve these three variables simultaneously.
Our experiments target two typical applications: using low-field MRI to correct brain deformation, and using high-field MRI to bring pre-operative BOLD into the OR. The experimental results show that the registration method, in the vicinity of the tumor resection, is more accurate than the state-of-the-art NRR we used in clinical practice [5]. Moreover, the experiment on the comparison between the heterogeneous model and homogeneous model demonstrates the effectiveness of the heterogeneous model in improving the registration accuracy.
Chapter 5

A Robust Hybrid Registration Method for Cardiac Motion Tracking

5.1 Introduction

In the previous two chapters, we focus on the development of a robust point-based NRR. In this chapter and the next chapter, we will pay attention to intensity based-NRR. For intensity-based NRR, the variation of the intensity imposes severe challenges on both mono- and multi-modality registration. To overcome this difficulty, we will present a robust hybrid method for Cross-Correlation (CC) based mono-modality registration and a Top-to-Down K-means clustering for Mutual Information (MI) based multi-modality registration.

Cardiac magnetic resonance imaging has proved its effectiveness in determining
patient-specific myocardial motion/functional information via the cine imaging, as well as detection of myocardial infarction appearing hyperintense in the DE-MRI. Recent studies comparing myocardial tissue viability, revealed in the DE-MRI, to the functional deficits, measured with cine MRI [101], showed that the so-called "peri-infarction zone" defined in DE-MRI correlates well with the dysfunctional myocardial region defined in cine. This information is potentially valuable for reperfusion therapy as regional motion of infarction zone defined before the therapy is assessed to evaluate the recovery of myocardium.

Although the clinical value of joint DE-MRI and cine image assessment is exhibited, standard clinical cardiac MR protocols usually acquire two sets of images across multiple measurements with variant imaging plane prescription and multiple breath-holdings. Misalignment and local deformation often appear between cine and DE-MRI, even the imaging plane remains unchanged for two acquisitions by careful prescription, mainly due to inconsistent cardiac phases used for acquiring cine and DE-MRI, imperfect cardiac gating, and respiratory motion. It is more problematic for patients with arrhythmias as unstable cardiac cycles make it unreliable to identify the matching cine frame acquired in the same cardiac cycle as the DE-MRI.

Without an accurate mapping of the infarction zone to cine images, regional myocardial changes in motion pattern caused by suspicious scars could only be visually assessed. The accurate alignment and deformation correction between cine and DE-MRI is thus a necessity for the successful joint assessment, where one aim is to propagate the infarction to all other cine frames throughout whole cardiac cycle and enable the quantitative regional motion pattern analysis.
Compared to the large amount of studies conducted on myocardial motion correction within one acquisition, there exists little research focusing on aligning cardiac MR images acquired across acquisitions with different pulse-sequences. One approach performing joint analysis is to manually segment images from multiple acquisitions, and to match the resulting AHA model [13]. Other approaches rely on aligning epicardial surfaces which are delineated before analysis [85]. More recently, a 2D-3D rigid image registration method was proposed to align DE-MRI and perfusion slices to the 3D whole heart coronary angiography volume [1], which enables the visualization of infarction in the 3D context.

The contribution of this work is to develop dedicated post-processing algorithms for aligning DE-MRI with corresponding cine image and propagating suspicious infarction zones to all other cardiac phases. Infarction regions delineated in the DE-MRI can be used to define the region-of-interest (ROI) for the quantification of regional abnormality of myocardial motion. To achieve these goals, we propose to align DE-MRI to cine image using a hybrid registration algorithm, unifying both intensity and feature points into one cost function. The intensity term is used to match two images on a coarse level, playing a role of regularization and dominating the alignment of normal myocardium, while the feature point term is robust against contrast changes between DE-MRI and cine as in the latter infarction zone bearing little contrast compared to normal myocardium is largely invisible. The propagation of infarction zone throughout the cine is achieved by estimating myocardial deformation in the cine series using a variational non-rigid registration algorithm with inverse consistent constraint.
5.2 Method

To align the DE-MRI to cine and propagate suspicious infarction, two types of deformation need to be estimated. The first corrects the mis-alignment between DE-MRI and cine and the second quantifies myocardial motion within cine series. Fig. 5.1 illustrates the proposed procedure.

As multiple cine images are required to cover the whole cardiac phase while one DE-MRI image is usually acquired at a specific temporal phase, the cine image, which is most similar with the DE-MRI, is selected as the reference to which the DE-MRI is registered. Assume the k-th phase is the reference image. The deformation $d_k$, from $c_k$ to the DE-MRI $E(p_i)$, is recovered by the hybrid registration method, and both forward and inverse deformation fields $d_{l,i}, l = 1 \ldots n, i = k$ are recovered by the variational method. Once all deformation fields $d_{l,i}, l = 1 \ldots n$ are computed, the DE-MRI and
infarction can be propagated.

5.2.1 Select the Reference Frame in the Cine

The cine frame, which is most similar with the DE-MRI, is selected as the reference. If available in the database, the trigger time is used to match the DE-MRI. For cine series where trigger time is not recorded, the Cross-Correlation is computed between every cine image and DE-MRI. The frame with largest CC value is picked.

5.2.2 Compute Deformation Fields within Cine Series

To propagate the suspicious infarction from the reference to all other cine frames, the deformation between cine images is estimated using a fast variational non-rigid registration algorithm [15]. This approach can be considered an extension of the classic optical flow method. In this framework, a dense deformation field is estimated as the solution to a calculus of variation problem, which is solved by performing a compositional update step corresponding to a transport equation. The regularization is added by low-pass filtering the gradient images, which are in turn used as velocity field to drive the transport equation. To speed up the convergence and avoid local minima, a multi-scale image pyramid is created. We selected the local cross correlation as the image similarity measure, because its explicit derivative can be more efficiently calculated than mutual information, and is still general enough to cope with intensity fluctuation and imaging noise between two adjacent perfusion frames.

Registration of time series such as MR cine is usually performed by selecting a reference phase to which all other phases are registered. This approach is not sufficient as both DE-MRI images and the infarction zone, which is represented as a contoured
region, are propagated throughout the cardiac phases. Specifically, deformation fields pointing to the reference phase are required to warp the DE-MRI image while the inverse deformations pointing from reference phase to other frames are needed to warp the infarction contours. We thus extend the above-mentioned registration algorithm to estimate the inverse consistent deformation fields.

A deformation field $\Phi_{pq}$ is inverse consistent if $\Phi_{pq} \cdot \Phi_{pq}^{-1} = I$ and $\Phi_{pq}^{-1} = \Phi_{qp}$. $\Phi_{pq}$ is retrieved by minimizing the inverse consistent similarity metric:

$$J_{icCC} = J_{CC}(f_p, f_q, \Phi_{pq}) + J_{CC}(f_q, f_p, \Phi_{qp})$$

(5.1)

Here $J_{CC}$ is the local cross correlation. $f_q$ and $f_p$ are two cine phases. Their deformation is $\Phi_{pq} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$.

We have developed an efficient update scheme of the iterative gradient descent, in order to minimize the inverse consistent similarity in reasonable time [43]. In essence, each deformation field is alternately updated during descending the gradient of similarity measure, resulting in an accurate computation of the inverse deformation and a quasi-symmetric registration algorithm. The extra computational effort for inverse consistent deformable registration is only about 10%-15% when compared to Hermosillo et al. [29]. The achieved inverse consistency not only allows for propagating both images and contours between any two cardiac phases, but also often leads to more accurate quasi-symmetric image registration.

The variational deformable registration method is robust for cine images, as each adjacent image pair shows similar image content and contrast. Unfortunately, it is less suitable to register DE-MRI to cine reference phase, as the former often presents a
Before registration
(a) Before registration

(b) After registration

Figure 5.2: A DE-MRI image is registered to the selected cine frame using the variational registration. The warped DE-MRI shows unrealistic deformation due to enhanced infarction bearing no contrast in cine.

strongly enhanced infarction zone which bears no contrast in the cine series. As a result, the pixel-wise variational registration tends to generate unrealistic large deformation, which degrades the image quality of warped DE-MRI images even with aggressive regularization, as demonstrated in Fig. 5.2.

5.2.3 Register DE-MRI to the Reference Cine

To cope with inconsistent visibility or occlusion of the infarction zone between DE-MRI and cine, and to produce robust registration, we propose a hybrid registration algorithm, which unifies intensity-based and point-based similarity into one cost function. This cost function contains two terms: a point matching term and an intensity matching term. Specifically, the point matching term is robust against contrast changes and occlusions between DE-MRI and cine. The intensity term enforces the alignment of myocardium with normal contrast uptake, playing a role of global regularization. The underlying deformation is modeled as a Free-from deformation [93], which is a piece-wise cubic polynomial. Compared to pixel-wise variational registration, Free-form deformation is more robust against image content changes.

Free-form deformation (FFD) FFD can be manipulated by a regular control grid with spacing $s_x \times s_y$ for 2D image. FFD is computationally efficient because the
deformation at any point is only influenced by its surrounding 4 × 4 control points.

For a point \( p \) with coordinate \((x, y)\), assume its 4 × 4 control points are \( p_{ij}, i, j = 0, \ldots, 3 \). Denote \( d_{ij} \) as the displacement vector associated with the control point \( p_{ij} \), the interpolation at point \( p \) is defined as:

\[
T(p|d_{ij}) = \sum_{i=0}^{3} \sum_{j=0}^{3} B_i(u)B_j(v)d_{ij},
\]

where \( u = x/s_x - \lfloor x/s_x \rfloor \), \( v = y/s_y - \lfloor y/s_y \rfloor \). \( B_i \) is the \( i \)-th basis function of B-splines [93].

**Cost function** Given reference image \( R(p_i), i = 1 \ldots N \) and its feature point set \( \{s_j\}_{j=1}^{M} \), and floating image \( F(p_i) \) and its feature point set \( \{t_j\}_{j=1}^{M} \), we define the following minimization problem:

\[
\hat{D} = \arg\min_D \left( \frac{\lambda}{M} \sum_M \|T(s_j|D) - t_j\|^2 - \frac{\sum_N (R(p_i) - \bar{R})(F(T(p_i|D)) - \bar{F})}{\sqrt{\sum_N (R(p_i) - \bar{R})^2 \sum_N (F(T(p_i|D)) - \bar{F})^2}} \right)
\]

where the first term is point matching and the second term is intensity matching. \( R \) is the reference image and \( F \) is the floating image. \( \bar{R} \) and \( \bar{F} \) are the mean intensity of \( R \) and \( F \), respectively. \( D \) is the unknown parameter set \( \{d_{ij}\} \). \( \lambda \) is used to balance the influences of both terms. The value of \( \lambda \) depends on the metric in the intensity term. We experimentally select Cross-Correlation (CC) as the intensity metric and \( \lambda \) is set to be 0.5. Equation 5.3 is solved by L-BFGS optimization, which, compared to simple gradient descent, is more efficient for high dimensional optimization problems [71].

**5.2.4 Propagate the DE-MRI**

Once all deformation fields \( d_l, l = 1 \ldots n \) are computed, the DE-MRI \( E \) can be propagated to yield all \( n \) cardiac phases: \( E_l^*, l = 1 \ldots n \). First, the DE-MRI \( E \) is deformed
to $E^*_k$ using $E^*_k = E(d_k(p_i))$. Then, $E^*_k$ is propagated to the remaining $n - 1$ phases using $E^*_l = E^*_k(d_l(p_i)), l = 1 \ldots n, l \neq k$. Note the propagation of delineated infarction contours requires the inverse deformation fields pointing from reference to other $n - 1$ phases. It is provided by the inverse consistent registration of cine series. To better present propagated DE-MRI, three propagation schemes as shown in Fig. 5.3 are implemented: whole image propagation, contour propagation, and ROI propagation. Whole image propagation resamples the whole DE-MRI to the cine coordinates. Contour propagation only deforms the scar boundary. ROI scheme transforms the scar region and superimposes it directly on cine images.

5.3 Results

Both TrueFISP cine and inversion recovery TurboFlash delayed enhancement imaging were performed on 6 patients with suspicious myocardial infarction using standard clinical protocols. DE-MRI was acquired between 10 and 30 minutes after administering the contrast agent. Experiments were conducted using a 1.5T Siemens Avanto scanner. For every subject, the slice prescription between cine and DE-MRI acquisitions was kept unchanged to minimize the through-plane displacement. The proposed analysis workflow was applied to all datasets, and outputs were first inspected. For all cases the hybrid reg-
Figure 5.4: Comparison between intensity-based method and the hybrid method. (a) cine image. (b) DE-MRI. (c) aligned DE-MRI by CCPD. (d) before registration. (e) CC registration. (f) CCPD registration.

Registration produces better alignment. As an illustration of typical performance, Fig. 5.4 shows a comparison between intensity based registration (Cross-Correlation, CC) and hybrid method (Cross-Correlation with Point Distance, CCPD). We delineate a contour on the aligned DE-MRI by CC registration and a contour on the aligned DE-MRI by CCPD registration. The delineated contour is superimposed on the cine image to show the registration result.

Table 5.1: Quantitative measures of DE-MRI to cine registration.

<table>
<thead>
<tr>
<th></th>
<th>Cine-Original DE-MRI</th>
<th>Cine-CC moco</th>
<th>Cine-CCPD moco</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dice ratio</td>
<td>0.65 ± 0.06</td>
<td>0.64 ± 0.16</td>
<td>0.80 ± 0.08</td>
</tr>
<tr>
<td>False positive</td>
<td>0.32 ± 0.08</td>
<td>0.30 ± 0.15</td>
<td>0.14 ± 0.08</td>
</tr>
<tr>
<td>False negative</td>
<td>0.39 ± 0.08</td>
<td>0.42 ± 0.18</td>
<td>0.26 ± 0.08</td>
</tr>
<tr>
<td>MBE (in mm)</td>
<td>3.40 ± 2.56</td>
<td>3.33 ± 2.75</td>
<td>2.12 ± 1.57</td>
</tr>
</tbody>
</table>

The quantitative evaluation was performed by manually delineating the myocardium on the cine reference frame and aligned DE-MRI image. Four statistical measures are computed to give a comprehensive quantification: Dice ratio (the myocardium overlap ratio); false positive (the percentage area of myocardium labeled in cine but not labeled in DE-MRI); false negative (the percentage area of myocardium not labeled in DE-MRI).
cine but labeled in DE-MRI); MBE (the myocardium boundary errors, defined as the minimal distance between myocardium contours, endo and epi, extracted from the cine and DE-MRI slices). Table 5.1 summarizes the results, showing superior performance of the hybrid approach. For these 6 datasets, in-plane resolution is $1.18 \sim 1.36 \text{ mm}^2$. Compared to doing nothing, CC method shows worse performance (lower Dice ratio and higher False negative), demonstrating its characteristic susceptible to the contrast change.

![Graph](image)

Figure 5.5: Area and thickness of both infarction and healthy myocardium over one cardiac cycle.

Fig. 5.6 shows the propagated DE-MRI and the infarction contours. Suspicious infarction can degrade myocardial contraction. To highlight the potential of the proposed workflow for abnormal motion pattern detection, we delineated the scar region in the DE-MRI and label the myocardial segment containing the scar. Both the contour and segment are propagated to all cardiac phases using the estimated forward/inverse deformation fields. At each phase $p$, the area of the infarction zone $A_p$ is computed by counting the number of the internal pixels. The thickness $T_p$ is computed by calculating the epi/endo distance of the segment. To alleviate the inter-subject variability, $A_p$ and $T_p$ are normalized with respect to phase 0, i.e. $A_p = A_p/A_0$, $T_p = T_p/T_0$. For the
comparison, the normal myocardium is also delineated, of which the area and thickness were computed. Fig. 5.5 shows the changes of the area and the thickness across one cardiac cycle for one test case. The area and thickness of healthy myocardium is found to change more significantly over cardiac phases compared to infarction zone.

![Figure 5.5](image)

**Figure 5.5:** The changes of the area and the thickness across one cardiac cycle for one test case.

![Figure 5.6](image)

**Figure 5.6:** The first row is the cine images; the second row is propagated DE-MRI, and the third row is propagated contour superimposed on the cine. The propagated contours are zoomed in at last row. The image with a green box in the first row corresponds with $C_k$ in Fig.5.1, and the image with a green box in the second row corresponds with $E_k^*$. Due to space limit, we only present the results at three different phases.

To quantify the change potentially caused by the suspicious infarction, we use $(A_p - A_0)/A_0$ to represent the relative area change and $(T_p - T_0)/T_0$, basically the segment strain ratio, to represent the relative thickness change. The mean and variance of 6
Table 5.2: Area/Thickness change %. ACI: Area Change of Infarction zone. ACN: Area Change of Normal myocardium. TCI: Thickness Change of Infarction zone. TCN: Thickness Change of Normal myocardium.

<table>
<thead>
<tr>
<th>Cases</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI</td>
<td>4.5 ± 0.1</td>
<td>4.5 ± 0.2</td>
<td>8.4 ± 0.4</td>
<td>9.7 ± 1.1</td>
<td>3.1 ± 0.1</td>
<td>6.3 ± 0.2</td>
</tr>
<tr>
<td>ACN</td>
<td>13.1 ± 0.7</td>
<td>4.6 ± 0.2</td>
<td>6.7 ± 0.3</td>
<td>2.5 ± 0.0</td>
<td>10.6 ± 1.1</td>
<td>8.0 ± 0.2</td>
</tr>
<tr>
<td>TCI</td>
<td>2.7 ± 0.1</td>
<td>3.8 ± 0.1</td>
<td>5.9 ± 0.3</td>
<td>7.2 ± 0.3</td>
<td>3.7 ± 0.1</td>
<td>5.3 ± 0.2</td>
</tr>
<tr>
<td>TCN</td>
<td>23.5 ± 5.1</td>
<td>19.9 ± 4.9</td>
<td>15.5 ± 2.6</td>
<td>7.6 ± 0.7</td>
<td>20.0 ± 3.1</td>
<td>14.6 ± 1.1</td>
</tr>
</tbody>
</table>

Cases are listed in Table 5.2. Case 1 and 5 show noticeable decrease of both area and thickness changes for the infarction, while thickness dropped more in cases 2, 3 and 6. Interestingly, case 4 shows the contrary that relative area change increases for the infarction although the registration and propagation performed well, which was verified by visual reading. While it is known that contrast enhancement of the myocardium is not a specific sign for myocardial infarction [40], we are reluctant to draw any physiological conclusions here. On the other hand, these initial experiments do reveal the feasibility of joint DE-MRI and cine assessment which could lead to more thorough clinical studies of regional wall motion changes related to ischemic heart diseases.

5.4 Conclusion

This chapter presents dedicated post-processing algorithms to align DE-MRI images to cine series, and to propagate the suspicious infarction zone to all cardiac phases. These warped infarctions define the ROI for the quantification of regional abnormality of myocardial motion, which enables the joint cine and DE-MRI assessment. Key algorithmic steps include aligning DE-MRI image to cine using a hybrid registration algorithm combining both intensity and point-based similarity terms. The myocardium deformation within the cine series is recovered by the inverse-consistent non-rigid registration, which
enables the propagation of both delayed enhancement images and delineated scar regions. Initial experiments show the effectiveness of the proposed method and highlight its potential to perform quantitative motion pattern analysis. We are now proceeding to further validate these methods and apply them to the study of myocardial motion abnormality associated with proved or suspicious ischemic heart diseases.
Chapter 6

A Robust Multi-modality Registration Method for MRI and CT

6.1 Introduction

Image registration can be classified into two categories: mono-modality and multi-modality registration. Multi-modality registration is more complex than mono-modality because the subjects are imaged in different ways, resulting in no direct relation between the intensities of two images.

Archip et al. [6] presented a feature point-based method to non-rigidly register pre-procedural MRI with intra-procedural unenhanced CT images for improved targeting of tumors during liver radiofrequency ablations. This method employs block matching to identify deformation on sparsely distributed registration points and then applies this
sparse deformation on a biomechanical model to derive the entire brain deformation. This method heavily relies on the result of block matching. However, intensity-based block matching is not effective in estimating the correct displacement between two blocks located in different modality images whether we use correlation or mutual information as the metric.

Currently, mutual information, presented by Viola and Wells [108], is the most popular similarity measure employed by multi-modality registration. MI measures the statistical dependence of two images and does not rely on the relation of the intensity. Loeckx et al. [58] presented a conditional mutual information measure to deal with spatially-varying intensity inhomogeneity. This method extends a traditional 2D joint histogram to 3D by incorporating spatial location as an additional dimension along with intensity pair.

Mattes et al. [61] used MI as a similarity measure for PET-CT image registration in the chest. The motions between two images are modeled with a global rigid transformation and local cubic B-splines. This deformation model allows closed-form expression for the gradient of the cost function. The visual inspection, conducted by two experts in specific anatomic locations, reported errors were in the 0- to 6-mm range.

Mutual information requires the number of bins, an interval of intensity, to be decided a priori and then splits the intensity range into equidistant bins. This intensity splitting does not take the intensity distribution into account, and therefore probably leads to misalignment. Z.F. Knops et al. [46] overcame this difficulty by applying K-means on joint histogram. This approach yields varying bin sizes and achieves a more natural clustering of intensities. They evaluated their method on rigid MRI, CT, and PET
registration. Unlike their work, we only apply K-means on CT instead of on both CT and MRI.

In our work, we evaluate the combination of K-means and MI on the non-rigid registration (NRR) of MRI and CT. A Top-to-Down K-means clustering method is developed to generate a clustered CT (labeled CT), and then the resulting clustered CT is non-rigidly registered with MRI, termed as Cluster-to-Image registration, by modeling the underlying movement as Free-Form deformation [89]. We compare this non-rigid Cluster-to-Image registration method with 1) ITK implementation [45], an equidistant bin method, termed as Image-to-Image, and 2) Cluster-to-Cluster method (registration of two clustered images). Our preliminary experiment demonstrates this Cluster-to-Image approach significantly increases the accuracy of NRR.

6.2 Method

We use clustered CT to register with original MRI instead of clustered MRI. CT has a large range of intensities, usually from -1000 (Hounsfield units) to positive several thousands, and, therefore, K-means clustering is able to effectively deal with CT and strengthen the amount of information. On the contrary, MRI has a small range of intensities. As a result, different tissues are probably grouped into one cluster, resulting in the loss of information. We illustrate this point using Fig. 6.1.

Using clustered CT to register with MRI is equivalent to registering the original CT with varying bin sizes, determined by clustering, with MRI. A high number of bins, i.e. small bin size, is preferred for MRI. Different small bin sizes in MRI do not influence the registration result once the bin sizes of CT are determined using K-means clustering.
Figure 6.1: K-means clustering. (a) is original MRI and (b) is K-means clustered (labeled) image, in which midbrain and white matter with label 180, gray matter and skin with label 162 fall into the same cluster. (c) is CT, whose window position and width are carefully adjusted, and (d) is clustered CT.

We clarify this point from the definition of MI [108]:

\[ I(A, B) = H(A) + H(B) - H(A, B), \]  

(6.1)

where \( H(A) \) and \( H(B) \) are Shannon entropy of image A and B, respectively. \( H(A, B) \) is the joint entropy calculated as \( H(A, B) = \sum_{a,b} p(a,b) \log p(a,b) \), where \( p(a,b) \) is the joint probability of gray value \( a \) in image A and gray value \( b \) at the corresponding voxel in image B. The Shannon entropy is a measure of dispersion of a probability distribution. A distribution with a single sharp peak corresponds to a low entropy value, whereas a dispersed distribution yields a high entropy value. In other words, the less the combinations of \( (a,b) \) there are, the lower the entropy is. Now we are ready to use Fig. 6.2 to illustrate the influence of the bin size on the registration.

Assume the blue region in the left image of Fig. 6.2 (a) corresponds to the green region in the right image. For simplicity, the transformation is limited to the translation only. In equation 6.1, \( H(A) \) and \( H(B) \) are used to make \( I(A, B) \) insensitive to the overlapping region [108], which can be ignored since we only focus on the alignment of the blue region and the green region. If the blue region is totally matched with the
Figure 6.2: The influence of the bin size to the registration. (a) shows the misalignment of blue region with green region leads to additional (blue, background) combination, and therefore a higher joint entropy. (b) shows some details of blue region can be distinguished using a small bin size. The misalignment leads to additional (yellow, background), (red, background), and (green, background) combinations. However, a small bin size does not change the registration result. (c) shows a large blue region is produced by a large bin size. The registration result is not unique.

- The green region, $-H(A, B)$ should reach a maximum value because the misalignment (as shown in Fig. 6.2 (a)) will lead to an additional (blue, background) combination, which will disperse the distribution. If we use a small bin size for the left image, some detail structures (yellow and red regions in Fig 6.2 (b)) are distinguished. However, this does not influence the registration result because any misalignments will lead to additional combinations. The only difference between (a) and (b) is the maximum value of $I(A, B)$ in case (b) is smaller than that in case (a). If we use a large bin size, a possible large grouped blue region is generated. Any translation that makes the green region totally covered by the blue region (Fig. 6.2 (c)) is a solution. The above discussion means that if one image is correctly clustered, a small bin size or a large bin number for another image is preferred.

- The non-rigid Cluster-to-Image registration is implemented in two steps: cluster CT first, and then non-rigidly register the clustered CT with MRI. We use K-means for CT clustering. K-means requires the number of clusters as input. A small number is likely to combine different tissues together, but a large number is likely to separate one tissue into different clusters. We determine the optimal number of clusters using a
Top-to-Down method by initializing K-means with a larger number of clusters and then gradually combining any two sufficiently close clusters.

### 6.2.1 Top-to-Down K-means Clustering

Let $x$ be a random variable with $N$ observations: $x_0, \ldots, x_{N-1}$. K-means is used to find the center of the cluster $\mu_k, k = 0, \ldots, K - 1$, and the assignment of data points to clusters by minimizing [11]:

$$ J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| x_n - \mu_k \|^2, \quad (6.2) $$

where $r_{nk}$ is a binary indicator variable describing which of the $K$ clusters contains data point $x_n$. $r_{nk} = 1$ and $r_{nj} = 0$ for $j \neq k$ denotes $x_n$ is assigned to cluster $k$.

Expectation and Maximization (EM) algorithm [22] is employed to find $r_{nk}$ and $\mu_k$ simultaneously. The EM algorithm proceeds iteratively and in each iteration two successive steps are involved:

**E step:** minimize $J$ with respect to $r_{nk}$ with $\mu_k$ fixed.

$$ r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \| x_n - \mu_j \|^2 \\ 0 & \text{otherwise} \end{cases} \quad (6.3) $$

**M step:** minimize $J$ with respect $\mu_k$ with $r_{nk}$ fixed.

$$ \mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}} \quad (6.4) $$

K-means is sensitive to the initialization and requires priori knowledge on the number of clusters. We overcome these difficulties by initializing K-means with a large number of
clusters and then iteratively combining the two closest clusters if the distance between
them is below some predefined threshold. This Top-to-Down K-means algorithm is
described in Algorithm 7.

\[
[B, K] = \text{K-means}(A, K, \xi)
\]

**Input:** \(A:\) image, \(K:\) the number of initial clusters, \(\xi:\) predefined cluster distance

**Output:** \(B:\) clustered image, \(K:\) the number of final clusters.

1. \(S \leftarrow \{0, \ldots, K - 1\}\)
2. \(x_i \leftarrow A(i)\) // \(x_i\) is the gray value at position \(i\) of image \(A\)
3. repeat
   4. Initialize \(\mu_k\) with \(S\)
   5. repeat
      6. Estep: Estimate \(r_{nk}\) according to equation 6.3
      7. M step: Solve \(\mu_k\) according to equation 6.4
   8. until no change of the cluster centers
   9. Find two closest clusters \(\mu_i\) and \(\mu_j\)
   10. if \(\|\mu_i - \mu_j\| < \xi\) then
      11. \(S \leftarrow S - \{\mu_i, \mu_j\}\)
      12. \(S \leftarrow S + \{(\mu_i + \mu_j)/2.0\}\)
      13. \(K \leftarrow K - 1\)
      14. end if
   15. until no two clusters combined
   16. Generate clustered image \(B:\) \(B(i) = k\), if \(r_{ik} = 1\)

**Algorithm 7:** Top-to-Down K-means clustering.

### 6.2.2 Non-rigid Registration of Clustered CT with MRI

We employ Free-Form Deformation [89] as a non-rigid transformation to model a 3D
deformable object, and Mutual Information to measure the statistical dependence be-
tween two images. The mutual information between reference image \(R\) (a clustered CT)
and the transformed floating image \(F(T(x, y, z|d_{ijk}))\) (a pre-operative MRI) can be ex-
pressed as a function of the transformation parameter vector \(D\), a concatenation of all
control point displacements \(d_{ijk}\) [61].
\[ S(D) = - \sum_{l} \sum_{k} p(l, k|D) \log \frac{p(l, k|D)}{p_F(l|D)p_R(k)} \]  

(6.5)

where \( p(l, k|D) \), \( p_F(l|D) \), and \( p_R(k) \) are joint probability distribution, marginal distribution of floating image and marginal distribution of reference image, respectively.

\( l, 0 \leq l \leq L_F \), and \( k, 0 \leq k \leq L_R \) are histogram bin indexes in the floating image and the reference image, respectively. For the reference image, \( L_R \) is set to be equal to the number of the clusters, i.e. \( K \). For the floating image, a large bin size is preferred. We conducted experiments on different bin sizes: \( K, 2K, 3K, 4K, 5K \) and found there was little difference for the results if \( L_F \geq 2K \).

The solution of function 6.5 can be resolved by L-BFGS optimization, which is particularly suited for high dimensional optimization problems [71].

### 6.3 Results

We conducted experiments on the non-rigid registration of MRI (dimension: \( 256 \times 256 \times 76 \), spacing: \( 0.9375 \times 0.9375 \times 2 \)) and CT (dimension: \( 512 \times 512 \times 75 \), spacing: \( 0.453 \times 0.453 \times 2 \)). MRI has been rigidly registered with CT. The Top-to-Down K-means clustering results, non-rigid registration results and the comparisons among Cluster-to-Image, Image-to-Image and Cluster-to-Cluster methods are presented in this section.

#### 6.3.1 K-means Results

The results of Algorithm 7 with different inputs \( A = MRI, K = 32, \xi = 2 \) and \( A = CT, K = 32, \xi = 2 \) are shown in Fig. 6.1. For MRI, 19 clusters out of the initial 32 clusters are combined with others even with a very small cluster distance 2. For CT, 31
Figure 6.3: The rigidly registered MRI (a) is non-rigidly registered with CT (b). The resulting MRI (c) is merged with CT and two merged slices are shown as (d) and (e).

clusters are generated, including some unremoved noises (scattered small white regions in Fig. 6.1). The clustered CT will be used in both Cluster-to-Image and Cluster-to-Cluster registration, and the clustered MRI will be used in Cluster-to-Cluster registration.

6.3.2 Non-rigid Registration Results

Non-rigidly registered MRI and its fusion with CT are shown in Fig. 6.3. We qualitatively compare our non-rigid registration method with Cluster-to-Cluster and traditional equidistant bin (Image-to-Image) methods. The results are presented in Fig. 6.4. It clearly shows that the Cluster-to-Image method matches the soft tissue boundaries better than the other two methods.

To quantitatively evaluate the result, we select 7 detectable feature points in CT and compare the registration accuracy among different registration methods with respect to these anatomical points. The Cluster-to-Image method demonstrates the highest accuracy, as shown in Table 6.1.
Figure 6.4: The comparison of the results. The row is the index of the slice and the column is the registration method. The bin number we use in the Image-to-Image method is 256, which yields the best result among 32, 64, 128, 256. The bin number in the Cluster-to-Image method is 31 (the number of clusters) for clustered CT, and $2K = 62$ for MRI. For the Cluster-to-Cluster method, the bin numbers are 31 for clustered CT, and 13 for clustered MRI respectively. Some detectable boundaries of soft tissues of CT, such as the cerebellar hemisphere, midbrain, and ventricles, are extracted, highlighted, and overlapped on registered MRI. The green arrows point to the boundaries exhibiting significant improvement of the accuracy using the Cluster-to-Image method.

6.4 Conclusion

We present a Cluster-to-Image non-rigid registration method to register MRI with CT. A Top-to-Down K-means method is developed to cluster CT. The clustered CT is non-rigidly registered with MRI by employing FFD as non-rigid transformation. This method overcomes the difficulty of Image-to-Image method to determine the bin size in MRI in the absence of the knowledge of the bin size in CT. Moreover, it also avoids the shortcoming of the Cluster-to-Cluster method regarding the loss of information. The preliminary
Table 6.1: Accuracy evaluation (mm) on 7 detectable feature points of CT: 1) anterior horn of right lateral ventricle (AHRLV), 2) pons (PONS), 3) anterior horn of left lateral ventricle (AHLLV), 4) posterior horn of right lateral ventricle (PHRLV), 5) posterior horn of left lateral ventricle (PHLLV), 6) septum pellucidum (SP), and 7) splenium of corpus callosum (SCC).

<table>
<thead>
<tr>
<th>Anatomical points</th>
<th>AHRLV</th>
<th>PONS</th>
<th>AHLLV</th>
<th>PHRLV</th>
<th>PHLLV</th>
<th>SP</th>
<th>SCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rigid registration</td>
<td>7.55</td>
<td>3.61</td>
<td>6.32</td>
<td>6.71</td>
<td>6.40</td>
<td>7.14</td>
<td>4.59</td>
</tr>
<tr>
<td>Non-rigid Cluster-to-Image</td>
<td>2.45</td>
<td>1.00</td>
<td>1.41</td>
<td>1.73</td>
<td>0.71</td>
<td>2.00</td>
<td>1.41</td>
</tr>
<tr>
<td>Non-rigid Image-to-Image</td>
<td>4.69</td>
<td>2.24</td>
<td>3.0</td>
<td>3.16</td>
<td>5.74</td>
<td>2.00</td>
<td>2.45</td>
</tr>
<tr>
<td>Non-rigid Cluster-to-Cluster</td>
<td>7.35</td>
<td>2.83</td>
<td>6.08</td>
<td>6.40</td>
<td>5.48</td>
<td>7.14</td>
<td>4.36</td>
</tr>
</tbody>
</table>

experiment demonstrates that this method is capable of increasing the accuracy of the non-rigid registration of MRI and CT.
Chapter 7

Multi-tissue Mesh Generation for Accurate Registration

7.1 Introduction

Multi-tissue mesh generation of medical images is a necessary procedure for building a heterogeneous biomechanical model, which has numerous applications such as physical model-based non-rigid registration, segmentation, and surgery simulation. However, there is little literature addressing this issue so far.

Several groups [12, 64, 80] presented multi-tissue mesh generation methods based on Delaunay refinement. However, elements with small dihedral angles (aka, slivers) are likely to occur in Delaunay meshes because elements are removed only when their radius-edge ratio is large; their dihedral angle quality is completely ignored. Meyer et al. [64] showed at least 0.6% slivers occurred in their experiments on frog data. Boltcheva et al. [12] and Pons et al. [80] employed a sliver exudation postprocessing technique [16]
to remove slivers and showed a very good quality mesh (minimal dihedral angle is larger than 4 degrees).

Unlike these Delaunay-based methods, Zhang et al. [113] presented an octree-based method to generate a tetrahedral and hexahedral mesh. This method first identifies the interface between two or more different tissues and non-manifold nodes on the boundary. Then, all tissue regions are meshed with conforming boundaries simultaneously. Finally, edge-contraction and geometric flow schemes are used to improve the quality of the tetrahedral mesh. In our work, we incorporate mesh quality, smoothing and fidelity into one point-based registration (PBR) framework.

Molino et al. [68] presented a crystalline, red-green strategy for mesh generation. This method starts with a Body-Centered Cubic (BCC) mesh and then deforms it to match the object boundary. The geometry is represented by a signed distance function, and the refinement is performed by a red-green strategy. This BCC-based approach shows a very good quality mesh because the quality of BCC mesh is high, and its regular refinement still leads to a BCC mesh. However, this approach is limited to a single tissue.

The contribution of this chapter is a novel mesh generation method which is characterized by 1) multi-tissue mesh, 2) tissue-dependent resolution, 3) natural control of the trade-off among quality, fidelity, and smoothness on tissue level.

7.2 Method

Our approach requires multi-label images as input, in which label 0 denotes the background, and positive integers indicate different tissues. The approach consists of two
steps: 1) coarse mesh generation (CMG) and 2) tissue-aware PBR, as shown in Fig. 7.1.

CMG includes two substeps:

1. BCC mesh:

   Use BCC mesh to subdivide the object space into connected tetrahedra. Note that this step does not distinguish different tissues. All tissues with label larger than zero belong to the same object (non-background object). The resulting BCC mesh is homogeneous.

2. Coarse tissue dependent resolution multi-tissue mesh generation (CMesh):

   This step specifies which tissue each tetrahedron belongs to and then yields a submesh for each tissue. Each tissue is capable of automatically adjusting its resolution based on its geometric complexity and the predefined subdivision criterion.

The resulting coarse multi-tissue mesh of step 1 includes different submeshes and each submesh has its own resolution. The discrepancy between the surface of the submesh and its corresponding boundary in the multi-label image is corrected by a tissue-aware PBR method. This step includes three substeps:

1. detect edges for each tissue in the multi-label image to obtain target point set

2. extract surface nodes for each submesh to obtain source point set

3. deform the surface of each submesh to its corresponding boundary based on PBR

The framework of this approach is shown in Fig. 7.1. Each step listed in this framework will be discussed in detail in the following sections.
7.2.1 Coarse Mesh Generation

The purpose of the coarse mesh generation is to obtain the source points, which will be used in the tissue-aware PBR method. The coarse mesh needs to take into account the following criteria: 1) multi-tissue input, 2) good conditioning for the subsequent PBR, and 3) fewer tetrahedra.

This part includes two steps as shown in Fig 7.1. Body-Centered Cubic provides an initial lattice, which has been well documented in [30,68]. For the completeness of this chapter, we will briefly describe its properties and red-green subdivision, then focus on how CMesh generates and refines submeshes.
7.2.1.1 BCC Mesh

BCC mesh is an actual crystal structure ubiquitous in nature. It is highly structured and easily refined initially or during the simulation. The nodes of BCC are grid points of two interlaced grids like the blue grid and the green grid in Fig. 7.2(a). The edges of BCC consist of edges of the grid and additional edges between a node and its eight nearest neighbors in the other grid.

![Diagram](image)

(a) A portion of the BCC lattice. (b) Red-green subdivision

**Figure 7.2:** BCC lattice and red-green subdivision (These two figures come from [68]).

The refinement of BCC mesh is performed by a red-green strategy. Initially, all BCC lattice tetrahedra are labeled with red. A red tetrahedron can be subdivided into eight children (1:8 refinement) and each child is labeled with red as shown in Fig. 7.2(b). There are three choices for the internal edge of the tetrahedron. If the shortest one is selected, the resulting eight child tetrahedra are exactly the BCC tetrahedra except the size is one half of the original BCC. So, the quality of the refined mesh can be guaranteed using this red (regular) subdivision. This is the reason that we select BCC as the initial tetrahedral mesh, although our method is general enough to start from any tetrahedral mesh. This red subdivision will lead to T-junctions at the newly created edge midpoints where neighboring tetrahedra are not refined to the same level. To remove the T-junctions, a green subdivision, including three cases, is performed. These three
cases are:

1. there is one edge with T-junction

2. there are two opposite edges with T-junctions

3. there are three edges of a face with T-junctions

The green subdivision according to these three cases is shown in Fig. 7.2(b). All the child tetrahedra of the green subdivision are labeled with green. This irregular green subdivision will reduce the quality of the tetrahedron, so all the child tetrahedra will be removed, and red subdivision is performed on their red parent when higher resolution is desired.

7.2.1.2 CMesh

CMesh is used to identify the submesh for each tissue in BCC mesh and subdivide it if necessary. We define a label operation table, based on which label redistribution method is used, to produce different submeshes. A predefined subdivision criterion is used to determine which submesh needs to be further subdivided. If a submesh needs to be subdivided, in order to reduce the number of the tetrahedra, only its boundary tetrahedra are further subdivided (multi-resolution).

In Fig. 7.3, we illustrate how CMesh identifies and subdivides submeshes. First, CMesh assigns each tetrahedron with a label of the tissue, to which most the tetrahedron belongs (Fig. 7.3(a)). As a result, an initial multi-tissue mesh is produced. However, this multi-tissue mesh is not well conditioned for subsequent deformation because more than one face, i.e. four nodes of one tetrahedron, are probably on the interface. We
term this kind of tetrahedron as a badly conditioned tetrahedron. In this case, deforming four nodes easily crushes the tetrahedron. We prefer a submesh only with two kinds of tetrahedra: inner tetrahedron (no faces on the interface) and boundary tetrahedron (only one face on the interface). To reach this end, we redistribute the label of the badly conditioned tetrahedra according to the operations defined in Table 7.1 to generate a well conditioned multi-tissue mesh (Fig. 7.3(b)). After label redistribution, we check if each submesh needs to be further subdivided. If it satisfies the criterion for the resolution, defined in Fig. 7.3(e), the algorithm stops. Otherwise, it subdivides (Fig. 7.3(c)) and redistributes labels (Fig. 7.3(d)). The above procedures repeat until the desired resolution is reached. The submesh produced by this label redistribution method not only has good conditioning, but also reaches conformity with its neighboring submeshes.

![Figure 7.3: Coarse multi-tissue mesh generation.](image)

(a) L1 and L2 are tissue labels, the dash line is the real boundary and the blue line is the submesh interface. (b) Redistribute labels according to operation table 7.1. (c) Subdivide if not satisfy the resolution criterion defined in (e). (d) Redistribute labels again. (e) Resolution criterion: 0.85 is the subdivision threshold, an experiment value evaluated on MRI, visible human and brain atlas. Points represent voxels and colors represent different tissues. $S_1$ is the voxel set within the blue submesh (blue dash lines) and $S_2$ is the voxel set within the blue tissue (blue curves).

**Operation table** The operation table decides how to redistribute the label of a tetrahedron based on its relation, termed as configuration, with face-adjacent tetrahedra. The purpose of the operations defined in Table 7.1 is to move the bad conditioned
tetrahedra to their neighboring submeshes. If all the badly conditioned tetrahedra are removed from one submesh, this submesh and its neighboring submeshes will reach good conditioning at the same time. We clarify this point by taking case 5 defined in table 7.1 as an example. If the four face-adjacent tetrahedra of a given tetrahedron $T$ have labels: $< L, L1, L1, L1 >$, denoted as $< L, 3L1 >$ for simplicity, the label of $T$ will be reassigned with $L1$ because its three faces are on the interface between submesh $L$ and $L1$. Fig. 7.3 uses case 5 for redistribution. Because we use 2D triangles instead of 3D tetrahedra in Fig. 7.3, case 5 is degenerated from $< L, 3L1 >$ to $< L, 2L1 >$. In summary, the operations defined in Table 7.1 move a tetrahedron to its face-adjacent submesh if the tetrahedron is not an inner (case 1) or boundary tetrahedron (case 2). As a result, no tetrahedra with more than one face on the boundary exist, which leads to a well conditioned mesh for the subsequent deformation.

Table 7.1: Operation case table for tetrahedron $T$ with label $L$.

<table>
<thead>
<tr>
<th>Case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4L</td>
<td>3L1</td>
<td>2L1</td>
<td>2L1</td>
<td>1L3</td>
<td>1L1,1L1,1L2</td>
<td>1L1,1L1,1L2,1L3</td>
</tr>
<tr>
<td>Operation</td>
<td>$T =$ inner tetra</td>
<td>$T =$ boundary tetra</td>
<td>$T . label = L1$</td>
<td>$T . label = L1$</td>
<td>$T . label = L1$</td>
<td>$T . label = L2$</td>
<td>$T . label = L1$</td>
</tr>
</tbody>
</table>

**Criteria for subdivision** In a multi-label image, a tissue is defined with a set of voxels with the same intensity, say $L$. Heuristically, the closer the surface of a submesh is to the boundary of a tissue, the more voxels of the tissue are located in the submesh, and the more voxels with label $L$ the submesh has. To quantitatively evaluate the similarity between the submesh and the tissue region, we define two voxel sets:

1. $S1$: all the voxels in the submesh (the points within two dashed lines in Fig. 7.3 (e))
2. $S_2$: all the voxels in the tissue region (the points within the curve in Fig. 7.3 (e))

$S_1 \cap S_2$ defines the point set shared by the submesh and the tissue region. We expect the common region to be similar to the submesh and the tissue region. We use $\frac{|S_1 \cap S_2|}{|S_1|}$ to measure the similarity between the common region and the submesh, and $\frac{|S_1 \cap S_2|}{|S_2|}$ to measure the similarity between the common region and the tissue region. So, the subdivision criterion can be defined as:

$$\frac{|S_1 \cap S_2|}{|S_1|} < \text{threshold} \quad \text{and} \quad \frac{|S_1 \cap S_2|}{|S_2|} < \text{threshold} \quad (7.1)$$

where threshold is an input parameter. $0 \leq \frac{|S_1 \cap S_2|}{|S_1|} \leq 1.0$ and $0 \leq \frac{|S_1 \cap S_2|}{|S_2|} \leq 1.0$, so $0 \leq \text{threshold} \leq 1.0$.

The reason that we simultaneously use two values as the criterion is to avoid case a and case b in Fig. 7.4. Moreover, we do not simply use $\frac{|S_1 \cap S_2|}{|S_1|}$ in order to avoid case c in Fig 7.4.

Figure 7.4: Three special cases. The circle represents the tissue region and the polygon represents the submesh. For simplicity, the voxels are not shown. All three cases show a big discrepancy between the tissue boundary and the submesh boundary. However, for case (a), because the tissue is totally covered by the submesh, $\frac{|S_1 \cap S_2|}{|S_1|}$ has the highest value 1.0. For case (b), because the submesh is totally covered by the tissue region, $\frac{|S_1 \cap S_2|}{|S_2|}$ has the highest value 1.0. For case (c), $\frac{|S_1 \cap S_2|}{|S_1|}$ can be equal to be 1.0, if the submesh and tissue region have the same number of voxels.

The criterion relies on the number of the voxels, and is, therefore, susceptible to the
resolution of the multi-label image. For instance, if the resolution is very low, we cannot find any voxels in a tetrahedron. To overcome this difficulty, up-sampling is performed automatically if no voxels are detected in a tetrahedron. To improve the performance, we do not perform up-sampling in the whole image, but restrict it to the bounding box of the tetrahedron.

7.2.2 Tissue-aware PBR

This step is used to 1) deform the coarse mesh close to the boundary, 2) maintain the quality of the coarse mesh, and 3) generate a smooth mesh. The coarse mesh needs to be deformed to the boundary. Unlike the interpolation method used in [68], we treat the deformation as a point-based registration. This method iteratively deforms the mesh towards the boundary of the multi-label image. In each iteration, the deformation will be viewed as a point-based registration problem. Each surface of the submesh will be registered with its corresponding boundary in the image. The advantage of this approach is that quality, smoothing, and fidelity can be incorporated into the same registration framework.

7.2.2.1 Source point Set and Target Point Set

Two point sets are needed in this registration framework: source and target point sets. The source points are the surface nodes of the mesh and the target points are the edge points in the multi-label image. The source point set is obtained by extracting the surface nodes of each submesh. The target point set is obtained by canny edge detection, which is facilitated by ITK implementation [45]. For each source point, the target point closest
to it will be viewed as its potential correspondence. It is computationally intensive to search for the closest point in all the target points. We associate each source/target point with a label to denote which tissue it belongs to, restricting the search only to the target points, which have the same label with the source point.

Fig. 7.5 shows the source point set and the target point set produced by visible human data. These intermediate results for other data will not be shown in Section 7.3.

![Figure 7.5: Point sets. The source point set (b) includes all the surface nodes of the coarse mesh (a), and the target point set (d) is the edge points in the multi-label image (c).](image)

**7.2.2.2 Register Source Points with Target Points**

The classic PBR [20] is used to register two images: floating image and reference image. The PBR is based on the concept of energy minimization. A sparse set of registration points within the floating image are identified. The displacement between the floating and the reference images is estimated using Block Matching [10] at each registration point. These displacements are applied as a boundary condition on a biomechanical model to derive the entire brain deformation.

In our work, we extend this PBR method and use it in the mesh generation field.
In our mesh generation, the registration points will be fixed to the nodes of the mesh instead of the feature points. The displacement of these registration points is estimated by taking fidelity, smoothing, and quality into account. The homogeneous biomechanical model used in [20] is generalized with a more flexible tissue-aware model as shown below:

\[
W(U) = \sum_{i=1}^{n} (U^T K_i U + \lambda_i (H_i U - D_i)^T (H_i U - D_i))
\]

(7.2)

where \( n \) is the number of the tissues, and \( K_i \) is the global stiffness matrix assembled by the tetrahedra within \( i \)-th tissue. \( K_i \) is related with two biomechanical attributes of \( i \)-th tissue: Young’s modulus and Possion’s ratio. The building of \( K_i \) has been well documented in [9]. \( H_i \) is the global linear interpolation matrix assembled by registration points.

Each registration point \( o_k \) with number \( k \) contained in tetrahedron with vertex numbers \( c_i, i \in [0 : 3] \) contributes to four \( 3 \times 3 \) submatrices: \( [H]_{k0}, [H]_{k1}, [H]_{k2}, \) and \( [H]_{k3} \). \( [H]_{kc} \) is defined as: \( [H]_{kc} = \text{diag}(h_i, h_i, h_i) \). The linear interpolation factor \( h_i \) is calculated as:

\[
[1, 0, 0, 0] [v_{c0} \ v_{c1} \ v_{c2} \ v_{c3}]^{-1} [v_{c0} \ v_{c1} \ v_{c2} \ v_{c3}]
\]

(7.3)

where \( v_{ci} \) is the node with number \( c_i \). Because we use the node as the registration point, which means \( o_k \) is same with one of the four nodes, equation 7.3 is reduced to:

\[
h_i = \begin{cases} 
1 & \text{for } o_k = v_{ci} \\
0 & \text{for } o_k \neq v_{ci} 
\end{cases}
\]

(7.4)

\( U \) is the global unknown displacement vector at the mesh nodes, and \( D_i \) is the distance vector at the \( i \)-th surface nodes. The first term of the energy function represents the
biomechanical strain energy, a measure of the mesh deformation. The second term represents the matching error between the source point set and the target point set, a measure of the fidelity.

We term energy function (7.2) as a tissue-aware model, because it is able to use \( \lambda_i \) to balance the quality and fidelity for the \( i \)-th tissue no matter whether this model is homogeneous (same Young's modulus and Possion's ratio for all tissues) or not.

Distance vector \( D \), omitting subscript \( i \) for simplicity, reflects the fidelity between source points and target points. To incorporate smoothing into the registration framework, we calculate \( D \) according to the relaxed target position by classic Laplacian smoothing. Generally, mesh smoothing is performed as a postprocessing after the mesh generation. However, this will lead to the smoothing out of control of the biomechanical model, so we reflect the smoothing as we calculate \( D \), and, therefore, naturally incorporate it into the energy function (7.2). The \( i \)-th entry \( d_i \) of distance vector \( D \) is calculated as follows.

Let the source point corresponding to \( d_i \) be \( s \), its normal be \( n \), and the set of its neighboring nodes be \( S \). The normal \( n \) is calculated by averaging the normals of the surface faces, which share the source point \( s \). For each point \( p_i \in S \), calculate its closest target point \( t_i, i = 1 \ldots m \). For \( s \), calculate its closest target point \( q \). The relaxed (smoothed) position of \( s \) is \( s' = \frac{\sum_{k=1}^{m} t_k + q}{|S| + 1} \). Projecting \( s' - s \) onto the normal of \( s \) leads to:

\[
d_i = (\frac{\sum_{k=1}^{m} t_k + q}{|S| + 1} - s) \cdot n
\]  

(7.5)

We illustrate the calculation of \( d_i \) in Fig. 7.6.
Figure 7.6: The calculation of $d_i$ of node $s$. $p_1$ and $p_2$ are two neighboring nodes of $s$. $t_1$, $t_2$ and $q$ are the closets points corresponding to $p_1$, $p_2$, and $s$, respectively. Their average position is $s'$. Project $s' - s$ on unit normal $n$ of node $s$ to produce $d_i$.

We minimize $W(U)$ by solving:

$$\frac{\partial W}{\partial U} = 0 \Rightarrow \sum_{i=1}^{i=n} (K_i + \lambda_i H_i^T H_i) U = \sum_{i=1}^{i=n} \lambda_i H_i^T D_i$$  \hspace{1cm} (7.6)

Once we obtain $U$, we can update the positions of the nodes of the mesh. This procedure will be repeated until the average error between source points and target points is below a predefined tolerance or the iteration reaches maximum number. The average error is evaluated by:

$$\bar{d} = \frac{\sum \|s_i - t_i\|}{|S|},$$  \hspace{1cm} (7.7)

where $s_i$ is a source point, $t_i$ is the closest target point of $s_i$, and $S$ is source point set.

This average error is also used to evaluate the fidelity in Section 7.3.

The whole method, including coarse mesh generation and PBR based deformation, is presented in Algorithm 8.
$M = $MultiTissueMesher($MultiLabelImage, tolerance$)

**Input:** $MultiLabelImage, tolerance$

**Output:** $M$: tissue dependent resolution multi-tissue mesh

1. **Coarse Mesh Generation:**
2. Generate BCC mesh $M$
3. Assign label for each tetrahedron in $M$
4. **repeat**
5. Label redistribution according to Table 7.1 to yield multi-tissue mesh $M$
6. **for** each subMesh **do**
7. **if** satisfy the subdivision criterion (equation (7.1)) **then**
8. Subdivide $M$ along the boundary using red-green strategy
9. **end if**
10. **end for**
11. **until** no subdivision

12. **PBR Deformation:**
13. Generate source point set by surface extraction from $M$
14. Generate target point set by edge detection from $MultiLabelImage$
15. **repeat**
16. Calculate $D_i$ using equation (7.5)
17. Assemble $K_i$
18. Assemble $H_i$ using equation (7.4)
19. Solve $U$ using equation (7.6)
20. Deform $M$ using $M \leftarrow M + U$
21. Calculate error $\tilde{d}$ using equation (7.7)
22. **until** reach maximum iteration or $\tilde{d} < tolerance$
23. Remove the tetrahedra with label 0 from $M$

**Algorithm 8:** Multi-tissue mesh generation.

### 7.3 Results

To fully evaluate this method, we first conduct an experiment on MRI, which includes two tissues: brain and ventricle. Then, we use two nerves in visible human data to evaluate the tissue-aware quality control. Finally, we qualitatively and quantitatively evaluate this method on a non-manifold data brain atlas.

#### 7.3.1 Real MRI

The ventricle has different biomechanical attributes from other tissues in the brain, and so it is often used to build a heterogeneous biomechanical model [109]. We evaluate our method on this simple heterogeneous model: the ventricle and the rest of the brain,
in which the Young’s modulus $E = 10Pa$, Poisson’s ratio $\nu = 0.1$ for ventricle, and $E = 3000Pa$, $\nu = 0.45$ for the rest of the brain [109]. The results are shown in Fig. 7.7. Fig. 7.7(a) is the multi-label image, in which labels 128 and 255 denote the ventricle and the brain, respectively. Fig. 7.7(b) is the coarse multi-tissue mesh and Fig. 7.7(c) is the final (deformed) multi-tissue mesh. The deformed mesh is cut through and magnified in Fig. 7.8(a). Fig. 7.8(b) is the wireframe view of two submeshes and Fig. 7.8(c) is the extracted ventricle. The subdivision threshold we used to produce Fig. 7.7(b) is 0.85. With this parameter, the outer boundary of the brain is not further subdivided, but its inner interface with the ventricle is subdivided twice. Fig. 7.8(b) clearly shows that the ventricle has higher resolution than the brain.

**Figure 7.7**: Multi-tissue mesh generation for MRI data. (a) is the multi-label image. The coarse multi-tissue mesh (b) is generated with subdivision threshold 0.85. (c) is the deformed multi-tissue mesh. The numbers of source points and target points are 4497 and 31241, respectively.

From Fig. 7.7(a), we can see that the segmented brain and ventricle are not smooth, but the brain submesh (Fig. 7.7(c)) and the extracted ventricle submesh (Fig. 7.8(c)) are very smooth. It demonstrates that this method has a low requirement for the segmentation due to the incorporation of the smoothing into the PBR framework.

To show the conformity of the interfaces, we first extract two submeshes: the brain

140
Figure 7.8: (a) is the closeup of the inner ventricle. (b) is the wireframe view of the two submeshes and (c) is the extracted ventricle.

and the ventricle. The extracted brain is shown in Fig. 7.9(a), in which the hole is induced by the extracted ventricle. The extracted ventricle is shown in Fig. 7.9(b). We want to insert the ventricle into the hole to show the conformity on the interface between the ventricle surface and the hole surface, so the ventricle surface should not be too smooth to distinguish surface triangles, otherwise the conformity is not easily observed.

Figure 7.9: (a) is the brain with a ventricle hole. (b) is the extracted ventricle surface. (c) is the wireframe view of the hole. The front surfaces of the brain are culled to show the hole.

To show the conformity, we need to visualize the two surfaces on the interface simultaneously. So, the hole should be visualized in a different way from the ventricle. We use a wireframe to show the hole in Fig. 7.9(c). Note that the front surface of the brain in Fig. 7.9(c) is culled to clearly show the hole. Fig. 7.10(a) is the result of inserting the
ventricle into the hole. Fig. 7.10(b) is the closeup of the interface of the two surfaces.

We conducted our experiment on Dell PowerEdge (2 x dual-core Opteron 2218, 2.6 GHz CPU) with a runtime of about 5 minutes.

![Figure 7.10](image-url)  
(a) Conformity  
(b) Closeup

**Figure 7.10:** (a) shows the conformity of the interface. The part in the rectangle is enlarged in (b).

### 7.3.2 Visible Human

We also evaluate our method using visible human data\(^1\). Its multi-label image is shown in Fig. 7.11(a). This data includes three tissues: two nerves (dorsal thalamus (DT) with label 50 and caudata nucleus (CN) with label 100) and the brain with label 255. Fig. 7.11 and Fig. 7.12 show the results of this data. We use the same subdivision threshold 0.85 for this data. Fig. 7.12(b) and Fig. 7.12(c) clearly demonstrate the tissue-dependent resolution: nerve CN with resolution 1 (subdivided once), nerve DT with resolution 2, and the brain with resolution 0.

We use this data for the evaluation of the tissue-aware control of the quality. The results are shown in Fig. 7.13. The top three figures are the closeup of DT and CNP.

\(^1\)http://www.nlm.nih.gov/
Figure 7.11: Multi-tissue mesh generation for visible human data. (a) is the multi-label image. The coarse multi-tissue mesh (b) is generated with subdivision threshold 0.85. (c) is the deformed multi-tissue mesh. The numbers of source points and target points are 5828 and 26060, respectively.

Figure 7.12: (a) is the closeup of the inner. (b) is the wireframe view of the three submeshes, and (c) is the extracted two nerves.

$(\lambda_{DT} = \lambda_{CNP} = 1.0)$, the dihedral angle distribution of the tissue DT, and the dihedral distribution of the tissue CNP. The bottom three figures are the results as we fix $\lambda_{CNP}$, but reduce $\lambda_{DT}$ to 0.25. The left two figures do not show a big difference, but the two middle figures clearly show the quality of DT improves from $[13.6, 76.1]$ to $[15.1, 80.6]$ because we pay more attention to the quality of DT. The two right figures do not show any big differences because we do not change $\lambda_{CNP}$. Compared to the MRI experiment, more time is needed (9 minutes), because more tissues are involved.
Figure 7.13: Tissue-aware quality control. The two values in the bracket are minimum and maximum dihedral angles.

7.3.3 Brain Atlas

We use brain atlas 2 to evaluate this method on non-manifold surfaces. The multi-label image is shown in Fig. 7.14(a) and the final multi-tissue mesh, produced with the same trade-off parameters ($\lambda_1 = \lambda_2 = ... \lambda_6 = 1.0$), is shown in Fig. 7.14(b).

![brain atlas image](image1)

(a) Brain atlas

![final multi-tissue mesh image](image2)

(b) Final multi-tissue mesh

Figure 7.14: Multi-tissue mesh for brain atlas. Five tissues along with the rest of the brain (a) are discretized. 43: right caudate nucleus (RCN), 53: left caudate nucleus (LCN), 98: right anterior horn of lateral ventricle (RAHLV), 99: left anterior horn of lateral ventricle (LAHLV), 140: corpus callosum (CC). The numbers of source points and target points are 6225 and 39136, respectively.

---

2http://www.spl.Harvard.edu/publications/item/view/1265
We magnify the interfaces of these tissues to show the conformity in Fig. 7.15 in a different point of view from Fig. 7.10. Fig. 7.16 has three subfigures and shows the fidelity, tissue-dependent resolution, and quality, respectively. The fidelity part shows the comparison of the fidelity before PBR (left) and after PBR (right). The figure is generated by cutting through the mesh and overlapping it with the same slice of the multi-label images. The black arrows point to the places where bigger improvement of the fidelity occurs. Compared with the inner structures, the brain shows bigger improvement of the fidelity. The reason for this difference is that the brain, compared with the inner structures, has lower resolution and, therefore, lower fidelity. Since we do not pay more attention to the inner structures (the same \( \lambda_i \) for all tissues), the tissue with lower fidelity improves fidelity more. The fidelity is evaluated using equation (7.7) and is listed in Table 7.2. In the resolution part, the mesh is cut through to show the tissue-dependent resolution. In the quality part, we present the distribution of the dihedral.
angle and aspect ratio under different trade-off parameter $\lambda$ ($\lambda_1 = \lambda_2 = ... \lambda_6 = \lambda$). The values in brackets are the minimum and the maximum values for the whole mesh. The values for each submesh are listed in Table 7.2. As we increase $\lambda$ from 1.0 to 1.5, i.e. paying less attention to the quality, the minimum dihedral angle reduces from 4.57 to 3.96 and the maximum aspect ratio increases from 8.80 to 15.83. It takes about 14 minutes to generate the final multi-tissue mesh.

![Figure 7.16: The evaluation of fidelity, tissue dependent resolution, and quality on the brain atlas.](attachment:image.jpg)

A good quality mesh is characterized by the absence of slivers, i.e. tetrahedra with a very small dihedral angle, or aspect ratio close to 1. One observation from the quality part is the number of tetrahedra with a ratio around 1 increases from 20000 to 40000, even when we pay less attention to the quality (increase $\lambda$ from 1 to 1.5). This can be explained by the fact that many tetrahedra happen to improve their quality as they are...
deformed to the boundary.

Table 7.2: Quantitative evaluation for the multi-tissue mesh on the brain atlas. The atlas is regularized as spacing: 1mm × 1mm × 1mm, size: 240 × 240 × 259. The parameters are: subdivision threshold=0.85, λ = 1.0.

<table>
<thead>
<tr>
<th>Nerve structures</th>
<th>RCN</th>
<th>LCN</th>
<th>RAHLV</th>
<th>LAHLV</th>
<th>CC</th>
<th>Other (brain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect ratio (Quality)</td>
<td>[1.03,3.75]</td>
<td>[1.07,3.01]</td>
<td>[1.02,6.84]</td>
<td>[1.03,4.07]</td>
<td>[1.03,3.96]</td>
<td>[1.02,3.80]</td>
</tr>
<tr>
<td>Dihedral angle (Quality)</td>
<td>[13.36,79.80]</td>
<td>[24.7,72.60]</td>
<td>[10.06,79.12]</td>
<td>[17.74,78.40]</td>
<td>[13.56,78.14]</td>
<td>[4.57,84.15]</td>
</tr>
<tr>
<td>Average distance (Fidelity)</td>
<td>0.80</td>
<td>0.91</td>
<td>0.79</td>
<td>0.82</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Number of tetras</td>
<td>2944</td>
<td>612</td>
<td>9480</td>
<td>3849</td>
<td>14937</td>
<td>109466</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>814</td>
<td>220</td>
<td>2589</td>
<td>1136</td>
<td>3766</td>
<td>21487</td>
</tr>
</tbody>
</table>

7.4 Conclusion

This chapter presents a BCC-based multi-tissue mesh generation approach. This method inherits the advantages of BCC lattice mesh and extends it to a multi-tissue mesher by dealing with conformity using label redistribution based on a predefined operation table. This method can reach tissue-dependent resolution by using red-green subdivision under the guide of a subdivision criterion. The flexible control of the quality, fidelity, and smoothing is obtained by incorporating these properties into a PBR framework. The experiments on the data ranging from MRI, to visible human, to brain atlas demonstrate the effectiveness of this method.
Bibliography


YIXUN LIU, ANDRIY FEDOROV, ERIC BILLET, RON KIKINIS, AND NIKOS CHRISOCHOIDES. Improved block matching for non-rigid registration of brain MRI. In Proc. of SEECCM09, 2009.


155


Yixun Liu

Yixun Liu is currently a research assistant in the Center for Real-time Computing of Computer Science Department at William and Mary, where he follows Professor Nikos to do research about Medical image non-rigid registration, Brain shift in IGNS, Multi-tissue mesh generation for medical image, and Parallel computing.

Yixun Liu is also a Master of Biomedical Engineering, and worked in The Key Laboratory of Image Processing and Computer Aided Surgery of Fudan University and Fudan Digital Therapy Corp. in China from 2003 to 2007.