A 3D brain digital atlas is an important tool for neuroscience research. Many different imaging technologies such as MRI, CT, PET, microscopic imaging, and wet section photography, etc., are available. Among the technologies, cryosectioning followed by wet section photography can yield the highest resolution image of a full brain section. However, the images in the raw image sequence are not aligned with each other and therefore must be spatially registered. This dissertation describes several new methods employed by the atlas construction process. The most important components of the image registration process are the objective function and the optimization strategy. Pairwise image registration is inappropriate due to the lack of consideration of global coherence. A novel objective function called minimum entropy of bad prediction (MEBP) is proposed. MEBP is based on information theory and can be used for multi-modal image registration as well as image sequence alignment (ISA). In ISA, MEBP concurrently take multiple images into consideration and therefore can yield better alignment result. The optimization algorithm is a new hybrid method composed of density-based clustering algorithm, multi-resolution method and simplex method. This new method is less data-specific and more suitable for semi-automatic or automatic image registration. The image post-processing and volume compression are other important components in atlas construction. The commonly existed Swiss-cheese type image defect is treated with wavelet-based method and long range correlation. An octree
variant, scalable hyperspace file (SHSF), was developed to encode the volumetric data set. SHSF can facilitate volume accessing so that efficient virtual brain slicer and surface viewer are possible to implement. Since a high-resolution brain digital atlas can occupy hundreds of megabytes to tens of gigabytes storage space, in order to support applications on the atlas, volume compression algorithms are necessary. Three compression algorithms for different purposes are proposed. They include lossless method, near-lossless method and progressive lossy-to-lossless method. The progressive method is the most versatile and generalized coder. The various topics covered in this dissertation are compiled into a coherent whole with the digital brain atlas as the central scene.
Abstract

A 3D brain digital atlas is an important tool for neuroscience research. Many different imaging technologies such as MRI, CT, PET, microscopic imaging, and wet section photography, etc., are available. Among the technologies, cryosectioning followed by wet section photography can yield the highest resolution image of a full brain section. However, the images in the raw image sequence are not aligned with each other and therefore must be spatially registered. This dissertation describes several new methods employed by the atlas construction process. The most important components of the image registration process are the objective function and the optimization strategy. Pairwise image registration is inappropriate due to the lack of consideration of global coherence. A novel objective function called minimum entropy of bad prediction (MEBP) is proposed. MEBP is based on information theory and can be used for multi-modal image registration as well as image sequence alignment (ISA). In ISA, MEBP concurrently take multiple images into consideration and therefore can yield better alignment result. The optimization algorithm is a new hybrid method composed of density-based clustering algorithm, multi-resolution method and simplex method. This new method is less data-specific and more suitable for semi-automatic or automatic image registration. The image post-processing and volume compression are other important components in atlas construction. The commonly existed Swiss-cheese type image defect is treated with wavelet-based method and long range correlation. An octree variant, scalable hyperspace file (SHSF), was developed to encode the volumetric data set. SHSF can facilitate volume accessing so that efficient virtual brain slicer and surface viewer are possible to implement. Since a high-resolution brain digital atlas can occupy hundreds of
megabytes to tens of gigabytes storage space, in order to support applications on the atlas, volume compression algorithms are necessary. Three compression algorithms for different purposes are proposed. They include lossless method, near-lossless method and progressive lossy-to-lossless method. The progressive method is the most versatile and generalized coder. The various topics covered in this dissertation are compiled into a coherent whole with the digital brain atlas as the central scene.
To my parents, Qifang Tian and Yijiang Zhao

To my lovely wife, Qiaoqiao Ruan

To my kids, Joy and Ray
Acknowledgments

This dissertation would not have been possible without the love from my family members. Being my parents’ only son, my studying abroad is utterly responsible for their being lonely. Their unconditioned love often make me crying during the long night while I study on my bench.

I also owe my lovely wife a great deal. Without her love, I can’t imagine having today’s accomplishment. She encourages me while I am low in mood. She spends much more time with the kids than I do, so that I can have the time for my research and future career.

My twin kids, Joy and Ray, also contributed to the completion of this dissertation. These two new family members bring me so much happiness and also responsibilities. They are the catalysts that helped me overcome the great “Energy Barrier” to this work.

Beside the family, this work would not have been possible without Prof. Geneva G. Belford. She guided me through the research project and encouraged me to publish my research results in various academic forums. When I was tangled with many threads of cross related ideas, she is the one helping me clearing my thoughts. When the paper submission deadline is a few days away, she is the one helping me fixing the glitches and doing several revisions a week. She guided my baby steps in my academic life. Other than being a great academic advisor, Prof. Belford is also a great mentor in my personal life. I owe many of my life transitions to her.

It is Prof. Michael Gabriel who introduced me to this wonderful research subject and financially supported this work. His kindness, his generosity and his great insight in neuroscience research influenced this work tremendously.
I also thank Prof. Chenxiang Zhai and Prof. Jiawei Han for serving on my doctoral committee. Last but not least, I owe this work to several faculties and staffs in the finance department and graduate college. Without their moral and financial support, I wouldn’t be able to survive through the graduate program.
# Table of Contents

Chapter 1  **Introduction**  .................................................. 1  
1.1 Brain Digital Atlas .................................................. 1  
1.2 The Raw Image .................................................. 4  
1.3 Problem Statement .................................................. 5  
1.4 Dissertation Organization .................................................. 8  

Chapter 2  **Image Sequence Alignment Objective Function**  .......... 10  
2.1 Background .................................................. 11  
2.1.1 Transformations .................................................. 12  
2.1.2 Fundamental Components ............................................ 14  
2.1.3 Feature Space .................................................. 15  
2.1.4 Similarity Metric .................................................. 16  
2.1.5 Search Space .................................................. 19  
2.1.6 Image Sequence Alignment ............................................ 20  
2.2 MEBP as a PSM .................................................. 20  
2.3 MEBP as a GAME .................................................. 23  
2.3.1 The Plate Stacking Problem ............................................ 23  
2.3.2 Keep the Problem Simple ............................................ 24  
2.3.3 The Prediction Method ............................................ 25  
2.4 Results .................................................. 28  

Chapter 3  **Hybrid Optimization Algorithm**  .......................... 32  
3.1 Background .................................................. 32  
3.1.1 Local Optimization .................................................. 32  
3.1.2 Global Optimization .................................................. 34  
3.1.3 Optimization for Image Registration/Alignment .................................................. 35  
3.2 Introduction to the Hybrid Optimization Algorithm .................................................. 37  
3.3 Clustering Behavior of the Genetic Algorithm .................................................. 39  
3.4 Clustering Analysis .................................................. 41  
3.4.1 The 2^n-ary Tree Structure .................................................. 41  
3.4.2 GA’s Termination Condition .................................................. 42  
3.4.3 Identifying the Clusters .................................................. 42  
3.5 The Optimization Method .................................................. 46  
3.5.1 The Optimization Pyramid .................................................. 46
6.4.1 Overview of the Coding Method ........................................ 112
6.4.2 Wavelet Coefficients Tree ........................................... 114
6.4.3 Important Properties .................................................. 115
6.4.4 Traverse the Wavelet Coefficients Tree ............................. 116
6.4.5 Comments on the Lossy Coder ....................................... 121
6.4.6 Bit-Error-Resilient Coding .......................................... 121
6.4.7 Results for Progressive Lossy-to-Lossless Compression .......... 122

Chapter 7 Conclusion .......................................................... 124

Appendix A Brief Introduction to Mathematical Morphology ............ 128

Appendix B Image Pre-Processing .......................................... 131

References ............................................................................ 141

Biography .............................................................................. 153
# List of Figures

1.1 The plates stacking problem ........................................... 8  
2.1 Distribution of the prediction error .................................. 22  
2.2 The plates stacking problem ........................................... 23  
2.3 Illustration of imaginary rays ........................................ 25  
2.4 Intensity prediction along an imaginary ray using linear regression  26  
2.5 An illustration of GAME versus PSM .................................. 29  
2.6 GAME results in a 1D to 2D alignment case ......................... 31  
2.7 PSM performance comparison ......................................... 31  
3.1 Flow chart of a genetic algorithm .................................... 36  
3.2 Example of search space and component images ....................... 37  
3.3 GA’s clustering behavior ............................................. 40  
3.4 Optimization pyramid for traditional multi-resolution approach ........ 46  
3.5 Optimization pyramid for cluster-assisted hybrid approach ........... 47  
4.1 2D Hilbert curve .................................................. 55  
4.2 Raster display of the same image in different scan order ............... 55  
4.3 2D SCD and Hilbert curve .......................................... 56  
4.4 Automaton for 2D Hilbert curve .................................... 58  
4.5 The primitive figures of a 3D Hilbert curve ........................... 60  
4.6 Signature extraction ............................................. 65  
4.7 Image defect concealment results ................................... 67  
5.1 Linear code comparison ............................................. 71  
5.2 Illustration of Z-order ............................................. 72  
5.3 Layered data structure ............................................. 73  
5.4 Speedup diagram ................................................ 80  
5.5 Oblique sections .................................................. 81  
5.6 Sagittal section images generated with different levels of detail ........ 82  
5.7 Binarization and quadtree ......................................... 83  
5.8 Triangular surface mesh of a rabbit brain ............................ 83  
6.1 Visualization of different prediction methods ......................... 92  
6.2 Visualization of the binning method .................................. 98  
6.3 The parallelized compression algorithm ................................ 104
# List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of the PSMs</td>
<td>28</td>
</tr>
<tr>
<td>2.2</td>
<td>Weight function comparison in global alignment</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Analogies used in simulated annealing</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Analogies used in genetic algorithm</td>
<td>35</td>
</tr>
<tr>
<td>3.3</td>
<td>Comparison of optimization methods in image registration</td>
<td>49</td>
</tr>
<tr>
<td>4.1</td>
<td>State transition matrix for 2D Hilbert curve generator</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>State transition matrix for 3D Hilbert curve generator</td>
<td>61</td>
</tr>
<tr>
<td>5.1</td>
<td>Maximum chunk size for different length SLCs</td>
<td>74</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparison of different octree data structures</td>
<td>75</td>
</tr>
<tr>
<td>6.1</td>
<td>Comparison of different compression approaches</td>
<td>105</td>
</tr>
<tr>
<td>6.2</td>
<td>Comparison of different compression approaches II</td>
<td>113</td>
</tr>
<tr>
<td>6.3</td>
<td>Quantitative comparison of JPEG2000 and my progressive coder</td>
<td>122</td>
</tr>
<tr>
<td>B.1</td>
<td>Summary of the raw image problems</td>
<td>135</td>
</tr>
</tbody>
</table>
List of Acronyms

BPP  Bits Per Pixel
BPV  Bits Per Voxel
CAD  Computer Assisted Design
CAM  Computer Aided Manufacture
CC   Cross Correlation
CT   Computed Tomography
DC   Direct Current
DCT  Discrete Cosine Transformation
DFT  Discrete Fourier Transform
DHT  Discrete Hadamard Transform
DPCM Differential Pulse-Code Modulation
DSM  Nelder-Mead Downhill Simplex Method
FFT  Fast Fourier Transform
FOV  Field Of View
GA   Genetic Algorithm
GAME Global Alignment MEtric

GIF Graphics Interchange Format

GNU GNU’s Not UNIX ¹

GZIP GNU Zip

I/O Input / Output

IEEE Institute of Electrical and Electronics Engineers

IPOP Initial POPulation

ISA Image Sequence Alignment

IWT Integer Wavelet Transformation

LOD Levels of Detail

LZ77 Lempel-Ziv Coding 1977 ²

LZ78 Lempel-Ziv Coding 1978 ³

LZW Lempel-Ziv-Welch ⁴

JE Joint Entropy

JPEG Joint Photographic Experts Group

JTC Java-like Threads for C++

L-system Lindenmayer System

LC Linear Code

¹GNU is a recursive acronym for “GNU’s Not UNIX”.
²dictionary-based lossless data compression
³dictionary-based lossless data compression
⁴dictionary-based lossless data compression
Despite its rather misleading name, it is a functional programming language.

commercial data compression software

The term “QED” is actually an abbreviation and stands for the Latin quod erat demonstrandum, meaning “which was to be demonstrated.”
RIU  Ratio Image Uniformity

RLE  Run Length Encoding

RM  Registration Metric

ROI  Region Of Interest

SA  Simulated Annealing

SAD  Sum of Absolute intensity Differences

SCD  Swiss Cheese type image Defect

SFN  Society For Neuroscience

SHSF  Scalable HyperSpace File

SLC  Scalable Linear Code

SMP  Shared Memory Multi-processor

SNR  Signal-to-Noise Ratio

SPIE  Society of Photo-optical Instrumentation Engineers

SPIHT  Set Partitioning In Hierarchical Trees

SSD  Sum of Squared intensity Differences

VIR  Variance of Intensity Ratios

WINZIP  commercial data compression software

WT  Wavelet Transformation
Chapter 1

Introduction

Medical imaging has become a daily topic in science and technology development. In this rapidly developing field, there are still many problems to be solved. Research topics have migrated from pure 2D image processing towards more complicated ones. This field is undergoing transitions such as from 2D analysis to 3D analysis, from traditional image Processing problems to image archiving, image understanding, and image information retrieval. The research demonstrated in this dissertation follows the current trend and blends different blades from image processing, high-performance computing, optimization, information theory, data mining, neuroscience, etc. Despite the application-oriented flavor, this work is truly fueling the development of medical imaging.

1.1 Brain Digital Atlas

Brain atlases have been widely used in neuroscience research to study the histology of brain structure, diagnose brain disease, guide brain surgery, etc. Conventional atlases are printed photos. A digital brain atlas is essentially a digitized brain image. A three-dimensional digital brain atlas is a volumetric data set. With today’s technology, there is no way to obtain the high resolution three-dimensional volume image directly. What must be done is to stack a sequence of two-dimensional images together.

Manual image editing and alignment is still the most widely used method to create a dig-
ital atlas. However, people are not only interested in a particular animal’s brain atlas. The foreseeable huge amount of manual editing overhead blocks the way towards mass production of brain digital atlases. Many other tasks also need to be automated. In neuroscience research, the imaging techniques such as MRI, CT, PET, etc., can’t provide enough anatomical details and therefore high resolution histology images must be obtained. Although high resolution images can be obtained by sectioning the brain tissue, the images can no longer be stacked without being registered first. In this dissertation, I addressed a real-life neuro-imaging problem that involves aligning an image sequence. This specific problem is almost a daily task for neuroscience research. Some approaches [1] have been employed to solve this problem; however, they are still far from satisfactory and are labor intensive. To align the image sequence, I developed an image alignment objective function [2, 3] and a global optimization method [4]. I also developed an image defect concealment algorithm [5]. Details about image alignment and defect concealment are discussed in chapter 2 through chapter 4.

Given an aligned image sequence, in order to be able to access the huge volumetric data easily, I developed a data structure called Scalable Hyper-Space File, SHSF [6, 7], which is essentially a variation of the octree data structure. To reduce the excessive overhead in encoding the octree nodes, I used two levels of indirection and linear code in SHSF. The resulting data structure is much more compact, scales well and can be easily built and accessed using a parallel computer. For more detail about the SHSF data structure, please refer to chapter 5. The development of the SHSF data structure also leads me to another important issue. The data set is much bigger than those obtained from other imaging techniques. Rather than dealing with data sets of tens of megabytes, I am working on image sets in the range of several hundred megabytes to several hundred gigabytes. A data compression method is inevitable. I developed several data compression coders. The first one was a lossless coder [8, 9]. I use a context-based non-linear prediction method to preprocess the volume data set in order to effectively lower the entropy of the prediction error. The prediction error is further encoded using Huffman code. Unlike conventional methods, the
volume is divided into cubical blocks to take advantage of the data’s spatial locality. Instead of building one Huffman tree for each block, I developed a novel binning algorithm that builds a Huffman tree for each group (bin) of blocks. Combining all the methods above, I achieved an excellent compression rate compared to other lossless volume compression methods. In addition, the SHSF data structure is also used to index the huge volume so that parallel construction, on-the-fly accessing of compressed data without global decompression, fast previewing, efficient background compressing, and scalability can be achieved. The second coder is a near-lossless coder [10, 11, 12]. The near-lossless coder sacrifice some accuracy to improve the speed and compression rate. The quantization scheme is the core of the near-lossless coder. The third coder is a generalized image compression coder [13]. This coder is a progressive coder that can be used as either a lossy coder or a lossless coder. It can also be used on 2D images as well as 3D volumetric data sets. Details of the compression coders can be found in chapter 6.

A digital brain atlas can serve as the basis for many neuro-informatics applications [14]. With the digital brain atlas, people will be able to visualize the brain surface and 2D brain section images in any oblique direction, etc. Besides its use for visualization, the digital atlas can be the front end for a 3D spatial database of brain information. Many neuroscientists want to be able to specify regions, structures, neurons or other objects, such as points in the brain, so that they will be able to relate the anatomy to their study. They also want to associate experimental data¹ with the brain objects, so that the enormous amount of data can be organized in a proper way. Defining brain regions or building a spatial database is beyond the scope of this dissertation.

¹For example, time series data recorded by a probe at a recording site needs to be associated with the location of the probe and the surrounding anatomical structure.
The image set I worked with was prepared by cryosectioning a New Zealand white rabbit brain into 40-micron serial histology sections, mounting the sections onto glass slides, placing them on a light table, and photographing them with a 5X macro lens. This method provided greater structural detail and higher contrast than other methods such as CT, MRI, PET, etc. Other researchers tried to photograph the cutting plane of a frozen tissue block while sectioning. This method provided neither nearly close resolution nor contrast. The higher resolution and contrast come at the cost of severe misalignment and image defects. Raw image problems include debris, air bubbles, background noise, uneven lighting, rigid body image misalignment, loss of unattached structures, non-rigid image deformation, tissue overlapping or folding, image out-of-focus, etc. Some of the raw image problems can be solved by image pre-processing; some of them are image registration problems. After registration, the resulting three-dimensional volume image can be polished further for better visual appearance. To fix image problems other than misalignment is not the main focus of this dissertation; for detail about image preprocessing, please refer to appendix B. Image defect concealment is covered in chapter 4.

The rabbit brain cryosection image set is a typical data set. Severe image defects are almost unavoidable in cryosection images no matter how carefully the sectioning is performed. This is therefore a general problem that is encountered by the neuroscientists on a daily basis. The uncompressed size of the image can easily go beyond a few megabytes. To align the cryosection image sequence is a challenging task. Scientists from the Neuronal Pattern Analysis group tried various commercial software such as Analyze, Align, etc. An image sequence of close to 1000 slices took a seasoned image analyst about half a year to align with the help of available commercial software. This doesn’t sound like a feasible way to build a three-dimensional, high resolution digital brain atlas. To facilitate neuroscience research, an

---

2Researchers in the Neuronal Patterns Analysis group at the Beckman Institute provided the image set.
automatic or semi-automatic alignment method is needed.

1.3 Problem Statement

The difficulty of building an optimally aligned three-dimensional digital brain atlas from the raw data lies in the heavy computation cost and the lack of ground truth\(^3\). Most image registration problems have the following components: a registration quality metric, a search space, a search strategy and a feature space. A more comprehensive literature review is in chapter 2; here is only a brief overview. The feature space is formed by the extracted image information. It can include landmarks, image intensity, edges, etc. Often the image size has a direct influence on the size of the feature space. The search space is normally a high dimensional space; its number of dimensions is determined by the degree of freedom in terms of allowable image transformations. The registration metric, RM, is an objective function used to determine how well the images are registered. This objective function is evaluated over and over to find the best image transformation. A simple and efficient objective function can significantly improve the registration performance. An objective function that accurately represents physical reality can improve the registration quality. There is active research to try to find an accurate and efficient objective function. The search strategy decides how to make the next move in the search space in order to find the optimum transformation quickly. A naive strategy is exhaustive search which can be easily proved to be impractical. The optimization technique has a close relationship with the search strategy. In fact, the search strategy is a crucial component of the optimization technique. A good optimization technique can reduce the number of iterations required to find the optimum and therefore is as important as the other components in the image registration problem. The image interpolation method is an additional component that may influence the registration performance. After each optimization iteration, a new transformation is generated. Image

\(^3\)There is no direct way to obtain the 3D image of an animal brain, therefore no standard is available to assess the quality of a 3D digital brain atlas.
interpolation is used to generate the new transformed image for comparison\textsuperscript{4}. Although it is hard to include the interpolation method in the before mentioned four major components, due to its repetitive calling during each iteration, a fast interpolation can be advantageous. However, research [15] showed that a slower yet more accurate interpolation of the image can reduce the possible local minima during the optimization process. So far, the fastest image interpolation method is the nearest neighbor interpolation. Nearest neighbor interpolation isn’t the best interpolation method. An accurate interpolation method may help to reduce the number of optimization iterations; however, it costs more per iteration. In contrast, a fast interpolation method may have negative side effects on optimization and therefore require more iterations for the optimization method to converge. I adopt the greedy method that uses nearest neighbor interpolation.

Building a 3D digital brain atlas from a sequence of cryosection images is a general yet specific problem. It is a general problem because it is regarded as a widely needed and difficult problem in neuroscience. It is a specific problem because it is a special topic in the image registration context. Unlike other image registration problems, its uniqueness comes from the following. First of all, the image size is extremely large. The uncompressed raw data has tens of megabytes for each image and there are a few hundred to a few thousand slices in a typical image sequence. People have to use DVD as the storage and backup media for such a data set. Obviously, scalability is a major issue. Secondly, some brain tissues are disconnected after sectioning, therefore producing many small pieces of disconnected component images. Matching component images with their corresponding components in the neighboring slices is important if automatic image registration is desired. Without a carefully designed image matching algorithm, a false match is likely to happen. Third, no matter how carefully the cryosectioning is performed, severe image defects are unavoidable. Many factors in the experiment contribute to image defects and these factors are rather biology experiment

\textsuperscript{4}A rotation of an arbitrary angle can transform a pixel (on a discrete grid) to a location that is no longer on the discrete grid and therefore image interpolation is very important in image transformation.
issues, which is not a concern from the image analysis point of view. Experiment techniques need to be improved to produce better raw data set. From the image analysis standpoint, how to fix the damaged image is an important and extremely difficult task. Fourth, a 3D volume image generated from a pairwise aligned 2D image sequence is not necessarily a good representation of the original biological specimen. Figure 1.1 demonstrates a plate stacking problem. Given a few plates marked alphabetically, I want to stack them together so that every plate has maximum overlap with its immediate neighbor plates. There are many ways to stack them together and the resulting stacks can be very different. Without introducing further constraints, all the stacks shown are fully compliant with the plate stacking rule. The plates can be regarded as the image slices and the maximum overlap can be regarded as the pairwise similarity metric. The pairwise similarity metric only uses the information from the immediate neighbors. If the plot of the similarity between the two neighboring slices has a flat basin, it is likely that the resulting digital atlas is pairwise perfect yet globally distorted. Additional global information needs to be considered to form a new global alignment metric (GAME). Fifth, to find the transformation that best aligns the images is an optimization problem in which local minima often trap the optimization procedure. A global optimization heuristic such as simulated annealing or a genetic algorithm can possibly be used to solve this problem.

Image postprocessing is the final stage of building a 3D brain digital atlas. After registering the images, the sequence of images are stacked together to form a 3D image\textsuperscript{5}. There are two major components at this stage; namely, image defect concealment and volume compression. The defects in the 3D image can be missing image block, minor image misalignment, rough image boundaries, etc. If those defects are not fixed, they can be easily picked up by virtually slicing the 3D atlas in an oblique direction. The missing image blocks need to be restored based on the information from the neighboring voxels. As for the minor image mis-

\textsuperscript{5}In the perspective of this research, 3D brain digital atlas, 3D image and volumetric data set refer to the same thing and will be used alternatively in this dissertation.
alignment and rough image boundaries, they are in the form of image noise and therefore can be filtered out. Finally, in order to efficiently store, transfer and access the large volumetric data set, a good compression method is desirable. Lossy and lossless compression methods can be used for different purposes. Compression methods will be covered in chapter 6.

1.4 Dissertation Organization

This dissertation is divided into a number of self-contained chapters. Each chapter discusses the background of the topic followed by my contributions. Chapter 2 defines an objective function for image sequence alignment. The image sequence is said to be aligned when this function reaches its global optimum. Chapter 3 proposes a hybrid optimization method
based on the genetic algorithm, clustering analysis, hierarchical technique, and the simplex local optimization method. Chapter 4 treats Swiss-cheese-type image defects. The defect restoration method is critical for good visual appearance. Chapter 5 describes the SHSF data structure, which can facilitate applications based on the digital brain atlas. Chapter 6 offers three methods for volumetric data compression. Namely, a lossless method, a near-lossless method and a progressive lossy-to-lossless method. Chapter 7 concludes this dissertation with some future research directions.
Chapter 2

Image Sequence Alignment Objective Function

A brain anatomical image sequence obtained through histology posed a new challenge to medical image registration. Aligning hundreds to thousands of image slices using a pairwise registration technique may cause error propagation or introduce random error. Information across multiple adjacent image slices must be considered for the alignment. I developed a new similarity metric called Minimum Entropy of Bad Prediction (MEBP) that is suitable for pairwise image registration and Image Sequence Alignment (ISA). MEBP is intensity-based\(^1\), but it outperforms almost all other intensity-based metrics. When MEBP is used in ISA, it scales well. MEBP has been applied to a rabbit brain digital atlas construction, and it is applicable to many similar problems. This chapter starts by discussing the background of image registration. It is followed by a section describing MEBP as a Pairwise Similarity Metric (PSM) and MEBP as a Global Alignment MEtric (GAME).

Before I start the discussion, there are two important concepts that need to be defined. The adjective “Global” carries wide meanings. In ISA, it means a certain operation is carried over a long sequence of images. In image transformation and other low level image operations, it means a certain operation or property that belongs to or is applicable to all the pixels in the image. The adjective “Local” is the antonym of “Global”. In ISA, it is not explicitly used. In image transformation and other low level image operations, it means a

\(^1\)Intensity is the grayscale level of a pixel or voxel. This dissertation only deals with grayscale images.
certain operation or property that only applies to some pixels, depending on their location in the image. These two adjectives appear at many places in this dissertation. Their exact meaning depends on the context.

2.1 Background

Image registration  [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 15] and image alignment are two closely related concepts. Image registration can be defined as a mapping between two images both spatially and with respect to intensity. Let’s denote point \( \mathbf{x} = \{x, y\} \). If two images are defined as two 2D arrays of a given size denoted by \( I_{\text{ref}} \) and \( I_{\text{float}} \) where \( I_{\text{ref}}(\mathbf{x}) \) and \( I_{\text{float}}(\mathbf{x}') \) each map to their respective intensity values, then the mapping between images can be expressed as:

\[
I_{\text{ref}}(\mathbf{x}) = r(I_{\text{float}}(T(\mathbf{x}'))) \tag{2.1}
\]

where \( T \) is a 2D spatial-coordinate transformation; i.e., \( T \) is a transformation which maps two spatial coordinates, \( x \) and \( y \), to new spatial coordinates \( x' \) and \( y' \), and \( r \) is the radiometric or illumination adjustment. In the rest of the dissertation, I use \( I_{\text{float}}^T \) to denote the transformed float image. The registration problem is to find the optimal spatial and intensity transformations so that the images are matched either for the purposes of determining the parameters of the matching transformation or to expose differences of interest between the images. [16] Image alignment is a special case of image registration. Image alignment deals with a sequence of 2D image slices, in which every pair of adjacent images are co-registered and the entire sequence forms a 3D image that closely represents the 3D image of the original object. With today’s technology, a 3D image cannot be directly obtained from the original object; therefore, there is no standard to compare with. A cross section of the virtual 3D image can be examined to determine the alignment quality. From the definition of image alignment, pairwise image registration is clearly the very first step towards ISA.
2.1.1 Transformations

In equation 2.1, \( T \) is the transformation that maps one spatial coordinate system into another. The image registration literature contains several types of transformations. From the simplest to the most complicated, they are the rigid body, affine, perspective, global polynomial and local transformation. To simplify the discussion and also due to the scope of this research, only two-dimensional transformations are discussed from here after. According to the literature [16], a rigid body transformation is composed of a combination of primitive transformations such as rotation, translation and scale change. Many people use equation 2.2 to formulate the rigid body transformation.

\[
T_{\text{rigid}}(x) = sRx + \vec{t}
\]

\[
= \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \cdot \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}
\]

\[
= \begin{pmatrix} s_x \cos \theta x - s_x \sin \theta y + t_x \\ s_x \sin \theta x + s_y \cos \theta y + t_y \end{pmatrix}
\]

(2.2)

where \( \vec{t} \) is a translation vector; \( s \) is a scalar matrix, and \( R \) is a rotation matrix. In addition, a rigid body transformation assumes \( s_x = s_y > 0 \). A rigid body transformation retains an object’s shape and relative size within the image. To transform a pixel to a new location, a sequence of primitive transformations are computed. A more efficient approach would be to combine the primitives so that the final coordinate positions are obtained directly from the initial coordinates, thereby eliminating the calculation of intermediate coordinates. By expanding the 2 by 2 matrix representations to 3 by 3 matrices and using a homogeneous
coordinate\(^2\) triple \(x = \{x, y, 1\}\), equation 2.2 can be reformulated as

\[
T_{\text{rigid}}(x) = t \cdot s \cdot R \cdot x
\]

\[
= \begin{pmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
s_x & 0 & 0 \\
0 & s_y & 0 \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
cos\theta & -sin\theta & 0 \\
sin\theta & cos\theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
\]

\[
= \begin{pmatrix}
s_xcos\theta & -s_xsin\theta & t_x \\
s_ysin\theta & s_ycos\theta & t_y \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
\]

\[
= \begin{pmatrix}
s_xcos\theta \cdot x - s_xsin\theta \cdot y + t_x \\
s_ysin\theta \cdot x + s_ycos\theta \cdot y + t_y \\
1
\end{pmatrix}
\]

(2.3)

In this research, I assume \(s_x = s_y = 1\) (if necessary, the image is adjusted earlier), and therefore the scalar term is removed from future discussion. Affine transformations are generalizations of rigid body transformations. They have the general properties that parallel lines are transformed into parallel lines and finite points map to finite points. A general affine transformation matrix has the form:

\[
T_{\text{affine}} = \begin{pmatrix}
a_{11} & a_{12} & b_1 \\
a_{21} & a_{22} & b_2 \\
0 & 0 & 1
\end{pmatrix}
\]

(2.4)

Due to the non-commutative property of matrix multiplication, a different order of transformations can yield a totally different result. Therefore, the entries in the affine transformation matrix are usually expressed in this general form[29]. A shear transformation is one type of
affine transformation. A reflection transformation is also affine; however, it normally does not occur in medical image registration [22]. Projective transformations and the more general perspective transformations account for distortions due to the projection of objects at varying distances from the sensor onto the image plane. Knowledge of the distance of the objects relative to the sensor is required to use a perspective transformation. The global polynomial transformation is one of the most general global transformations and can account for many types of distortions so long as the distortions do not vary too much over the image. Unlike the previously mentioned transformations, a local transformation maps points in the image depending on their location. Many deformable image registration methods employ such local transformations. In the scope of this dissertation, perspective transformations, global polynomial transformations, local transformations, etc., are not considered due to the simple and consistent imaging system. I will focus on the discussion of rigid body transformations (without the scalar factor) to correct global misalignment.

2.1.2 Fundamental Components

Image registration can be roughly divided into four fundamental components: a feature space, a search space, an optimization strategy and a similarity metric. The feature space contains the extracted image information that will be used for matching. The search space can be a high dimensional space and its number of dimensions is the transformation’s degrees of freedom. For example, the rigid body transformation has three degrees of freedom, namely the translation on the x-axis, the translation on the y-axis and degree of rotation. The three degrees of freedom yield a three dimensional search space that contains all the possible rigid body transformations within a certain range. Due to the discrete nature of the images, only discrete and finite search spaces are considered throughout this dissertation. Obviously, it won’t be necessary to consider continuous search spaces because when the increment of the search step is under a certain small amount, the transformed image will be the same as before. Image registration or alignment is in fact an optimization problem. The optimization
objective function is the registration or alignment metric, which governs the relative merit of each alignment. To assist image registration, a gold standard is required to determine the accuracy of the image registration algorithm. So far, when people say an image is registered, in fact, they mean the image is aligned based on the metric. Does the metric truly represent the real meaning of alignment in the physical object? This question doesn’t have an answer yet. Hopefully, the similarity metric is close enough to be considered as a good standard. In any case, the similarity metric is extremely important for image registration because it is both an optimization objective and a control factor of registration quality.

### 2.1.3 Feature Space

Features such as points, edges, statistical information or pixel intensity can be used for registration. Manually or automatically extracted feature points are called fiducial marks or salient features. Those points can be the points of locally maximum curvature on contour lines, centers of windows having locally maximum variances, line intersections, manually placed marks, etc. Point-based registration algorithms [1] can be efficient yet lack sufficient accuracy. Edge-based algorithms use internal edges and contour lines to match the images. Edges of single pixel width are difficult to get and also problematic; therefore, the registration quality can be rough. Statistical information such as moment invariants, centroid, mass center, etc., can provide some kind of quick and coarse initial estimation. They cannot be used for accurate registration and sometimes they produce false positive results. Pixel intensity is by far the most widely used feature space. Pixel intensity possesses the most amount of raw information, does not require a feature extraction routine and is the most robust compared to other feature spaces. The drawback of using intensities is the heavy computation cost per iteration. The choice of the feature space has direct influence on the image registration quality and the computational complexity. It is also closely related to the search strategy and similarity metric.
2.1.4 Similarity Metric

The similarity metric must match the feature space. In medical imaging, similarity metrics based on pixel intensity are the most widely used. They do not require a rather complicated feature extraction algorithm because all the pixels are used to evaluate the similarity between two images. Pixel intensity-based metrics can have better accuracy [16]; however, their run times are higher than similarity metrics that use points, edges, etc. Sum of squared intensity differences (SSD) [23], sum of absolute intensity differences (SAD) [22] and cross correlation (CC) are the few most intuitive similarity metrics.

\[
SSD(I_{ref}, I_{float}) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (I_{ref}(i, j) - I_{float}^T(i, j))^2
\]  
(2.5)

\[
SAD(I_{ref}, I_{float}) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |I_{ref}(i, j) - I_{float}^T(i, j)|
\]  
(2.6)

where \(N\) is the side length of the image\(^3\). The SSD will be zero if the reference image and the float image match exactly. To save the multiplication in SSD, SAD may be used. SAD is also less subject to biases from outlier intensities.

\[
CC(I_{ref}, I_{float}) = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left((I_{ref}(i, j) - T_{ref}) \times (I_{float}^T(i, j) - T_{float}^T)\right)}{\sqrt{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (I_{ref}(i, j) - T_{ref})^2} \sqrt{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (I_{float}^T(i, j) - T_{float}^T)^2}}
\]  
(2.7)

Cross correlation is another traditional statistical method. It estimates the degree to which two series are correlated. Because the average intensity is subtracted, this method can tolerate global illumination changes between the images. In the literature [18, 30, 25], cross correlation is also used in the frequency domain to help determine the rigid body transformation. The idea is to decouple the translational changes and the rotational changes by using a finite fast Fourier transformation, finite FFT. In particular, a translation in the spatial domain becomes a phase change in the frequency domain. A rotation in the spatial domain

\(^3\)For simplicity in discussion, the images are assumed to have a square shape and side length \(N\) pixels.
has the same angle as rotating the image in the frequency domain [22]. The modulus of an image’s power spectrum is translation invariant and therefore it is possible to obtain the rotation angle by using FFT and cross correlation. Compared to SSD and SAD, cross correlation has a higher computational cost. Another closely related similarity metric is variance of intensity ratios (VIR) [31] which is also known as ratio image uniformity (RIU) [22]. It is defined as follows:

\[
    r(i, j) = \frac{I_{\text{ref}}(i, j)}{I_{\text{float}}(i, j)}
\]

\[
    \tau = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} r(i, j)
\]

\[
    VIR = \sqrt{\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (r(i, j) - \tau)^2}{\tau} \tag{2.8}
\]

Maximizing VIR maximizes the uniformity of the ratios of the image intensity. It is less intuitive and not widely accepted.

SSD, SAD, VIR and cross correlation can only be used with the images acquired from the same image modality because pixels at the same location from different image modality can have totally different grayscale value\(^4\). Another major thread of intensity-based similarity metrics does not have the limitation of image modality. They are developed based on information theory. The amount of information contained in an image can be quantified by entropy [32] \(H\).

\[
    H = - \sum_i p_i \log p_i \tag{2.9}
\]

\(H\) is the average amount of information supplied by a data stream with a set of \(i\) symbols whose probabilities are \(p_1, p_2, p_3, ..., p_i\). For a 256-level grayscale image, every grayscale is represented by a symbol and therefore it has 256 different symbols. The meaning of symbols changes with the context. Each symbol is distinct and a symbol set has all the primitives that form a stream of data. The concept of joint entropy [19, 27] was an important contribution.

\(^4\)In this dissertation, intensity and grayscale level are used alternatively
When two images are unrelated, if they are stacked together, the total information is the sum of the information contained in those two images. When those two images are nearly aligned, less information is contained in the combined image because the image content in the second image is more likely to be predicted by looking at the first image. Registration based on joint entropy assumes that a perfect registration of two images contains the least amount of information. Joint entropy is therefore defined as

$$H(I_{\text{ref}}|I_{\text{float}}) = -\sum_{i=1}^{n} \sum_{j=1}^{m} p(i|j) \log p(i|j)$$  \hspace{1cm} (2.10)$$

where $n$ is the number of symbols in the reference image and $m$ is the number of symbols in the float image. $p(i|j)$ is the probability of coexistence of the symbol $i$ in $I_{\text{ref}}$ and the symbol $j$ in $I_{\text{float}}$ at the same coordinates. The minimum entropy assumption has the flaw that the least amount of information does not necessarily indicate a perfect match. Mutual information [33] (equation 2.11) was then developed to address this problem.

$$MI(I_{\text{ref}}, I_{\text{float}}) = H(I_{\text{ref}}) + H(I_{\text{float}}) - H(I_{\text{ref}}|I_{\text{float}})$$  \hspace{1cm} (2.11)$$

Studholme [34] pointed out that neither of the previous two entropy-based similarity metrics is overlap invariant. He developed the concept of normalized mutual information [34] (equation 2.12), which appears to behave better when the field of view is changed. In other words, normalized mutual information gives consistent results when the overlap (matched portion) between the two images takes less space due to a change (enlargement) of the viewing area.

$$Y(I_{\text{ref}}, I_{\text{float}}) = \frac{H(I_{\text{ref}}) + H(I_{\text{float}})}{H(I_{\text{ref}}|I_{\text{float}})}$$  \hspace{1cm} (2.12)$$

Sarrut [15] proposed a few additional similarity metrics based on signal analysis and statistical probability. **conditional probability**

**unrelated to the size of the overlap area.**
tics. None of those metrics gained popularity. So far, cross correlation, mutual information and normalized mutual information are the most widely accepted intensity-based similarity metrics.

Other than the intensity-based similarity metrics, point or contour-based metrics\[1, 22\] are also seen in the literature. The idea is to use manually or semi-automatically extracted features such as points, internal edges or outside contour to match the images. In particular, for point-based metrics, two sets of points are extracted from the reference and float image respectively. For every point in the reference image, find the closest point in the float image and then sum up the square of their distance. A match is obtained by minimizing the sum of the square distances between the two point sets. The accuracy of such similarity metrics is relatively lower than the intensity-based metrics and false positive results often occur. Another drawback is the requirement of feature extraction. Either the user must manually specify the feature for comparison or a highly sophisticated feature extraction algorithm must be developed. Manual feature extraction can hardly be accurate because a small characteristic point recognized by the human eye often occupies several to hundreds of pixels in the full resolution image. Automatic feature extraction algorithms are highly application specific and sometimes also require human intervention. Besides, perfect contour information often may not be available. Despite the problems, those point or contour-based metrics have small feature spaces and therefore usually require less computation time than the intensity-based metrics.

2.1.5 Search Space

The search space consists of all the possible combinations of transformations. It is usually a high dimensional infinite space. Although the search space can be infinite, due to the discrete nature of digital images, a fine enough discrete grid will be sufficient. The number of dimensions of the search space is the number of possible independent transformations. For example, if the only transformations are translations on the x, y axis and rotation, the
search space is three-dimensional. A best match can be found when the search space is exhaustively visited. Clearly, this is not a good idea. Optimization strategies are designed to efficiently screen the search space so that certain areas in the search space that apparently don’t contain the solution will be safely and effectively ignored. Optimization strategies will be discussed in chapter 3.

2.1.6 Image Sequence Alignment

The image sequence alignment (ISA) problem hasn’t attracted too much attention so far. The non-invasive nature of MRI, CT or PET often yields relatively well aligned image sets. Pairwise image registration can produce acceptable results. While preparing the digital atlas from cryosection image sets, the images are totally misaligned and tissues can be broken during the image preparation. Theoretical attempts [1] and existing commercial or GNU software such as Align, Analyze, etc., all assume nice image sets with perfect contours and use pairwise image registration for ISA. Countless manual ISAs are performed in the neuroscience labs with the help of general purpose image processing software such as Adobe Photoshop. However, none of the current approaches are satisfactory for digital atlas construction.

2.2 MEBP as a PSM

I developed a new similarity metric MEBP. My assumption is that each of a matched image pair can be used to predict the intensity of the other. The error entropy of the bad predictions is minimal when the two images are aligned. Given a pair of co-registered images, the reference image \( I_{ref} \) can be used to predict the floating image \( I_{float} \) and vice versa. The better the prediction, the better the registration. To quantify the prediction, the prediction error’s entropy is used. More specifically, let’s denote the prediction function as \( P \), and
define the prediction error $I_ε$ as an image

\[ I_ε = I_{float} - P(I_{ref}) \] (2.13)

Prediction function $P$ will be discussed in section 2.3.3. A successful prediction yields a prediction error of zero, but an unsuccessful one yields a non-zero prediction error. Let $E$ be the set that contains all the prediction errors $x$ of the unsuccessful predictions, namely

\[ E = \{x | x \in I_ε \land x \neq 0\} \] (2.14)

MEBP can be formulated as

\[ MEBP(I_{ref}, I_{float}) = H(E) \] (2.15)

where

\[ H(E) = -\sum p_x \log_2 p_x \] (2.16)

and $p_x$ is the probability of $x$ in the error image $I_ε$. MEBP examines the probability of the prediction error, whereas the multi-modality PSMs examine the intensities’ conditional probability. When two consecutive images are perfectly registered, the prediction error in $I_ε$ has a Laplacian distribution and the MEBP is minimum. When two images are not in alignment, the prediction error distribution is spread out. Examples are shown in figure 2.1.

This new PSM can be used for multi-modality image registration. When two corresponding images from two different modalities are aligned, the prediction error tends to have smaller entropy. Tissues that contain the same physical material are likely to have similar intensity in the image. Different imaging techniques may produce different intensities for the same type of tissue; however, the intensity differences for the same type of tissue between
Figure 2.1: In both diagrams, the horizontal axis represents the value of the prediction error and the vertical axis represents the error count. The vertical axis is truncated at 200 counts to better show the counts of the prediction errors. (A) error distribution for two matched images (B) error distribution for the same pair of images that are slightly off registration.

different modalities are likely to remain close. This property suggests a high probability for certain prediction errors. In other words, a good registration between multi-modalities has a small MEBP.

MEBP can be easily extended to a GAME. The extension will be discussed in section 2.3. The GAME version of MEBP takes multiple images into consideration and the prediction function $P$ is different than the one used in its PSM version.

MEBP is fast. In its PSM version, the prediction function $P$ is simple.

$$P(I) = I$$ \hspace{1cm} (2.17)

Let $N$ be the side length of the images and $L$ be the levels of grayscale. Only $N^2$ subtractions and at most $2L$ logarithmic calculations are necessary to compute MEBP. MEBP therefore has a smaller constant factor in its runtime complexity than any of the existing intensity-based PSMs. For example, the famous Mutual Information (MI) requires $N^2$ comparisons and $L^2$ logarithmic calculations. In intensity-based PSMs, it is unlikely that one would be able to reduce the order of runtime complexity. A smaller constant factor is already a big
saving.

2.3 MEBP as a GAME

MEBP is not just another PSM, it can be easily extended to a GAME. I start this section with a plate stacking problem in section 2.3.1. Then I discuss some rules of thumb for global alignment. The GAME mode of MEBP is described in section 2.3.3.

2.3.1 The Plate Stacking Problem

Figure 2.2: A stack of plates of different sizes can have many ways to stack them together. (A) The original stack (B,C,D) Three possible stacks. Stack C contains cumulative systematic error.

Image sequence alignment is like organizing a stack of plates as shown in figure 2.2. Suppose the original stack is figure 2.2A; I don’t know how the plates should be stacked together.
Suppose I only compare two plates at a time and assume the plates must have maximum overlap. A possible resulting plate stack is figure 2.2B. The method used to compare two plates can be considered as the similarity metric used in image registration. Because I don’t know the rules used to put together the original stack, I generate hypotheses based on observation. Figure 2.2C, 2.2D are two other examples. Apparently, some hypotheses are better than others. In medical image registration, there are many similarity metrics. Every metric has its own mathematical model; I can’t assume how the medical samples are formed. PSMs offer us a way to match two images; some error is reasonable. When the errors are accumulated across the image sequence, the result may be no longer acceptable. I can do better by developing a GAME that can take many consecutive images into consideration.

2.3.2 Keep the Problem Simple

Global alignment can easily become a huge optimization crisis when multiple images are involved in the alignment. A naive approach is to optimize the entire system. Assuming rigid body transformations with a scale factor of 1, the total number of degrees of freedom is \(3n\) where \(n\) is the number of images. Registering two images is already a computation-intensive problem; this naive method is extremely undesirable.

I reduce the problem to a simple solution. First of all, not all the images in the image sequence need to be considered at the same time. The pixel intensities are similar only in a small neighborhood, so only a limited number of consecutive images need to be considered together. Second, I can’t afford to compute a non-linear optimization problem in a high dimensional search space. Rather than making every image mobile, I align one image at a time with respect to the previous few already aligned images in the stack. In other words, I assume all the images that are already aligned are trustworthy. The cumulative error is reduced because I consider more than two images at a time. Under my rigid body assumption, the search space is three dimensional. Other than a little bit of extra work on the similarity metric, the global alignment is reduced to a problem that is not too much more complicated
2.3.3 The Prediction Method

MEBP computes the entropy of the prediction error; therefore, to extend MEBP from a PSM to a GAME is essentially extending the prediction method. In the PSM mode, the predicted image is the same as the reference image.

\[ I_{\text{prediction}} = I_{\text{ref}} \]  

(2.18)

In my extension to a GAME, the imaginary rays passing through the image stack as shown in figure 2.3 are used to predict the current image from a stack of previous images. Each ray passes through a number of pixels at the same location across several images. The pixels from the previous images are used to predict the current pixel’s intensity.

Figure 2.3: Imaginary rays that pass through the image stack. a) individual image slice. b) imaginary ray

I use linear regression [35] for prediction and fit the voxel intensities on the imaginary ray to a straight line as depicted in figure 2.4. There is no limit on the number of slices that
Figure 2.4: Intensity prediction along an imaginary ray using linear regression
(A) When there are two images in the stack, the predicted intensity of a pixel on the second image equals its corresponding pixel’s intensity on the first image. (B-D) When there are more than two images in the stack, linear regression is used to predict the pixel intensity on the last image.

a. constant b. regression line c. prediction error d. the sign of the prediction error e. predicted intensity f. real intensity
can be used in the GAME. In standard regression, all the data points are treated equally in computing the regression line. In real images, a pixel that is far away from the current pixel has little influence on the current pixel/voxel’s intensity. Let’s denote $u$ as the distance from the predicted pixel to the pixel in the ray. Instead of using the standard regression equation, I use a weighted linear regression. A weight function $\omega = f(u_i)$ is introduced into the regression so that remote points carry smaller weights. The image intensity is denoted as $I_i$. The weighted linear regression is a least square problem:

$$S = \sum_{i=1}^{n} \omega (I_i - \alpha - \beta u_i)^2$$

(2.19)

where $\alpha$ is the y-intercept and $\beta$ is the slope of the regression line. $S$ is minimized when $\frac{\partial S}{\partial \alpha} = 0$ and $\frac{\partial S}{\partial \beta} = 0$

$$\frac{\partial S}{\partial \alpha} = -2 \sum_{i=1}^{n} (I_i - \alpha - \beta u_i) \omega$$

(2.20)

$$\frac{\partial S}{\partial \beta} = -2 \sum_{i=1}^{n} u_i (I_i - \alpha - \beta u_i) \omega$$

(2.21)

therefore

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \left( \begin{array}{cc} \sum_{i=1}^{n} \omega & \sum_{i=1}^{n} u_i \omega \\ \sum_{i=1}^{n} u_i \omega & \sum_{i=1}^{n} u_i^2 \omega \end{array} \right)^{-1} \left( \begin{array}{c} \sum_{i=1}^{n} I_i \omega \\ \sum_{i=1}^{n} I_i u_i \omega \end{array} \right)$$

(2.22)

When $\omega$ is constant, equation 2.22 is the standard linear regression. Different weight functions, including linear decay (equation 2.23), exponential decay (equation 2.24), quadratic decay (equation 2.25) were tested.

$$f_{linear}(u_i) = 1.0 - \frac{i}{n}$$

(2.23)

$$f_{exp}(u_i) = \frac{1}{\gamma^i}$$

(2.24)

$$f_{quad}(u_i) = 1.0 - \left(\frac{i}{n}\right)^2$$

(2.25)
Table 2.1: Comparison of the PSMs

<table>
<thead>
<tr>
<th></th>
<th>MEBP</th>
<th>SSD</th>
<th>CC</th>
<th>MI</th>
<th>JE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR</td>
<td>94.69</td>
<td>76.78</td>
<td>76.78</td>
<td>99.10</td>
<td>99.10</td>
</tr>
</tbody>
</table>

where \(n\) is the number of slices, and \(\gamma\) is the rate of decay. Empirical data (section 2.4) shows that exponential decay is the best. Even though there is no limit on the number of slices \(n\) that can be used in MEBP, it will be wise to keep it relatively small; e.g., \(n\) between 2 and 4.

Other than the different prediction method, the GAME version of MEBP shares all the other properties with its PSM version. Figure 2.5 illustrates the idea that a GAME can more effectively reduce possible systematic error than a PSM. In terms of reducing random error, a GAME also has more constraints than a PSM and therefore will be more accurate. There is some computation overhead due to the more sophisticated prediction method; however, a GAME still has the same runtime complexity.

2.4 Results

To ease the visualization of the global alignment results. I demonstrate the 2D to 3D alignment using a 1D to 2D case. If I dissect the image in figure 2.6a into a sequence of scan lines and then register the scan lines back together using a PSM, it is no surprise that the reconstructed image is no longer the same as the original image. The difference image is shown in figure 2.6b. The peak signal-to-noise ratio (PSNR) of the reconstructed image can be measured to determine the accuracy of the alignment. The larger the PSNR, the better the alignment. A PSNR of \(+\infty\) means that the aligned image is the same as the original. Table 2.1 shows the PSNR of the reconstructed images using five different PSMs. Table 2.2 compares the PSNR values for three different weight functions and different numbers of scan lines used in the alignment. Comparing the tables, it is easy to see that global alignment
Figure 2.5: An illustration of GAME versus PSM
(Image Sequence A) Apply GAME on a stack of images
(Image Sequence B) Apply PSM on a stack of images
This illustration assumes that both GAME and PSM have the same amount of error. It also assumes the errors propagate through the stack systematically.
Table 2.2: Weight function comparison in global alignment. Table entries are PSNR values.

<table>
<thead>
<tr>
<th>weight function \ n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$+\infty$</td>
<td>133.17</td>
<td>128.32</td>
<td>125.02</td>
<td>118.44</td>
</tr>
<tr>
<td>Exp</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>122.40</td>
</tr>
<tr>
<td>Quad</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>128.32</td>
<td>125.94</td>
<td>113.79</td>
</tr>
</tbody>
</table>

using more than 2 scan lines at a time yields a better reconstructed image. Exponential decay with 2 to 5 scan lines appears to be the best. As shown in figure 2.7, MEBP is also the fastest PSM. The bars represent the relative speed, with the image transformation time as unit 1. The test image’s dimension is $1000 \times 1000$ pixels.
Figure 2.6: (a) A sample brain histology image. (b) Difference image between the reconstructed image and the sample image. The reconstructed image is obtained by registering 1D line segments using a PSM.

Figure 2.7: Relative PSM performance comparison.
Chapter 3

Hybrid Optimization Algorithm

Image registration/alignment problems usually fall into a standard framework. This framework consists of a feature space, a search space, a registration objective function, and an optimization algorithm. The objective function used in ISA is discussed in chapter 2. This chapter focuses on the optimization algorithm. A brief literature review is followed by my new hybrid optimization algorithm.

3.1 Background

An optimization strategy is used to find the extremum in the search space. Most of the methods developed before the 1970s were local optimization strategies, meaning the starting point in the search space determines the extremum they converge to. The convergence point is an extremum but not necessarily the global extremum.

3.1.1 Local Optimization

Local optimization methods include calculus-based methods, gradient-based methods, direct search methods\(^1\), etc.

Calculus-based methods analytically solve the optimization problem by setting the first derivative of the objective function to zero. Lagrange introduced the Lagrange multiplier

\(^1\)Direct search is different than exhaustive search.
for incorporating an equality constraint into the objective function. Calculus-based methods must have a smooth objective function which is twice differentiable. This is unusual for most real-life problems.

Gradient-based methods include Newton’s method, Quasi-Newton method, Gauss-Newton method, etc. The basic idea is to find a directional derivative so that the direction of steepest ascent or descent, depending on whether to maximize or minimize, can be used to obtain the next point. In order to minimize a function $f(\vec{x})$, where $\vec{x} = (x_1, x_2, ..., x_n)$, one starts with a candidate solution $\vec{x}_1$ and iteratively generates $\vec{x}_2, \vec{x}_3, ...$ by:

$$\vec{x}_{k+1} = \vec{x}_k - \alpha_k \nabla f(\vec{x}_k)$$  \hspace{1cm} (3.1)

where

$$\nabla f(\vec{x}_k) = \left[ \frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, ..., \frac{\delta f}{\delta x_n} \right] \vec{x}_k$$  \hspace{1cm} (3.2)

$\alpha_k$ is the step size. In Newton’s method, $\alpha_k$ is the inverse of the Hessian matrix

$$\alpha_k = \left[ \begin{array}{cccc} \frac{\delta^2 f}{\delta x_1^2} & \frac{\delta^2 f}{\delta x_1 \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_1 \delta x_n} \\ \frac{\delta^2 f}{\delta x_2 \delta x_1} & \frac{\delta^2 f}{\delta x_2^2} & \cdots & \frac{\delta^2 f}{\delta x_2 \delta x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta^2 f}{\delta x_n \delta x_1} & \frac{\delta^2 f}{\delta x_n \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_n^2} \end{array} \right]^{-1} \frac{\vec{x}}{\vec{x}_k}$$  \hspace{1cm} (3.3)

Taking the inverse of the Hessian matrix is an expensive operation. Many Quasi-Newton methods use different mechanisms for estimating the Hessian. If Gaussian elimination is used to generate the inverse Hessian, then the method is called the Gauss-Newton method.

Direct search methods refer to methods that do not evaluate the derivative of the objective function. The Nelder-Mead Downhill Simplex method (DSM) is a direct search method. In one-dimensional search space, a simplex is a line. In two-dimensional space, it is a triangle. A simplex is a polyhedron in n-dimensional space, where $n > 2$. Each iteration generates
a new vertex for the simplex. If the new point is better than at least one of the existing vertices, it replaces the worst vertex. In this way, the diameter of the simplex gets smaller and the algorithm stops when the diameter is close enough to zero. DSM is attractive for its robustness and intrinsic parallelism.

3.1.2 Global Optimization

Global optimization strategies are designed not to be trapped by local extrema. Up to now, there are only a few global optimization heuristics such as simulated annealing and the genetic algorithm [36]. The fundamental idea is to introduce randomness into optimization so that it can escape from a local extremum. Neither of the heuristic methods can guarantee that the solution is the global optimum; the methods may not even converge.

Table 3.1: Analogies between annealing in a physical system and an optimization problem

<table>
<thead>
<tr>
<th>Physical System</th>
<th>Optimization Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>feasible solution</td>
</tr>
<tr>
<td>energy</td>
<td>objective function</td>
</tr>
<tr>
<td>ground state</td>
<td>global optimal solution</td>
</tr>
<tr>
<td>rapid quenching</td>
<td>local search</td>
</tr>
<tr>
<td>temperature</td>
<td>control parameter</td>
</tr>
<tr>
<td>careful annealing</td>
<td>simulated annealing</td>
</tr>
</tbody>
</table>

Simulated annealing [37, 38](SA) is based on an analogy taken from thermodynamics. To grow a crystal, the experiment starts by heating the material to a molten state, then the temperature is reduced until the crystal structure is re-established. Bad things happen if the cooling is done too quickly. In particular, some irregularities are locked into the crystal structure and the trapped energy level is much higher than in a perfectly structured crystal. A few analogous concepts between the physical system and the optimization problem

\[ ^2 \text{Also known as Monte Carlo annealing, statistical cooling, probabilistic hill-climbing, stochastic relaxation, and probabilistic exchange algorithm} \]
are listed in table 3.1. Details of simulated annealing can be found in monographs about optimization techniques [38, 39].

<table>
<thead>
<tr>
<th>Biological System</th>
<th>Optimization Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>individual</td>
<td>feasible solution</td>
</tr>
<tr>
<td>fitness</td>
<td>objective function</td>
</tr>
<tr>
<td>most fit individual</td>
<td>global optimal solution</td>
</tr>
<tr>
<td>gene or chromosome</td>
<td>search parameter</td>
</tr>
<tr>
<td>mutate</td>
<td>random search</td>
</tr>
<tr>
<td>mate</td>
<td>explore the search space</td>
</tr>
<tr>
<td>generation</td>
<td>iteration</td>
</tr>
</tbody>
</table>

The genetic algorithm (GA) uses a biological process model to optimize highly complex cost functions. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the “fitness”. A GA does not require the derivative of the objective function and is well suited for parallel computing. The analogies between the biological system and the optimization problem are listed in table 3.2. The big picture of GA is shown in figure 3.1.

Optimization strategies are not limited by the local and global optimization techniques mentioned here. Combinatorics optimization [40] is another major branch, which includes dynamic programming, greedy algorithms [41], etc. Combinatorics optimization is not relevant to this dissertation.

### 3.1.3 Optimization for Image Registration/Alignment

In image registration and alignment, the optimization goal is

\[ T^* = \arg\min_{T \in S_T} C(I_{ref}, T \circ I_{float}) \]  

(3.4)
Figure 3.1: Flow chart of a genetic algorithm.

where \( T^* \) is the transformation that gives the optimum cost \( C \), \( S_T \) is the search space for the transformations, \( I_{ref} \) is the reference image, \( I_{float} \) is the floating image, and \( \circ \) denotes applying transformation \( T \) on \( I_{float} \). Equation 3.4 and equation 2.1 both refer to the image registration/alignment problem. Equation 3.4 views the problem from the optimization aspect, whereas equation 2.1 views it in terms of spatial and illumination transformations.

Brown’s famous survey paper [16] reviewed the early studies of optimization algorithms used in image registration. Due to the expensive objective function, a common tactic is to set an initial guess and then use multi-resolution or local optimization methods or both to find a local optimum of the objective function. For almost all local optimization methods, the initial location in the search space plays a critical role in the accuracy of the registration. Initial guesses are often obtained through data-specific methods. Momentum-based methods, phase correlation, or manual pre-alignment by the domain specialists are a few ways to get the initial guess. Multi-resolution methods [17] usually employ an exhaustive search on each level of resolution. A bad decision at the initial level can be fatal. As shown in figure 3.2, a
3.2 Introduction to the Hybrid Optimization Algorithm

During the process of constructing a three-dimensional digital brain atlas from a sequence of high resolution brain anatomical section images, problems arose. The images have extremely high resolution. A typical section image may have several mega pixels and a typical search space can have more than one local extremum.
image sequence may have several hundred to several thousand such sections. An automatic method is desired. Certain sections have symmetric structures or structures that are similar; these structures often confuse registration algorithms because the initial guess at a coarse resolution may be wrong. The subsequent local optimization can be diverted to a wrong answer. Global optimization algorithms such as the genetic algorithm [36, 38, 37, 21] are more reliable, yet do not guarantee timely convergence. Furthermore, the global optimization heuristics are not desired if the float and reference images are compared at full resolution. A coarse level exhaustive search [24] can be inefficient for big data sets. The ideal solution for this problem should have a few important features. The number of objective function evaluations at full resolution must be reduced to a minimum. The optimization algorithm should have the capability to escape from local minima. The optimization algorithm should converge. All those issues motivated this work. In order to simplify the discussion of the optimization method, I assume a rigid body transformation with a scale factor of 1. The transformation consists of x-axis, y-axis translation, and rotation; therefore, the search space is three-dimensional. I use mutual information [33] as the registration objective function and nearest neighbor interpolation during the image transformation. I also assume the floating image and the reference image co-register at the global optimum.

My new optimization method is a cluster-assisted hybrid optimization algorithm. This algorithm is based on a few assumptions:

- The location of the global extremum of the cost function corresponds to the desired solution.

- The global extremum found at the sub-sampling with low resolution can be used as the initial location for further local optimization.

- The location of the global extremum found using one sub-sampling is inside the basin of attraction of the global extremum for the next sub-sampling with higher resolution.

A clustering algorithm plays an essential role in the new method. Existing clustering
methods [42, 43] can be divided into five major categories; namely, partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. Partitioning methods require prior knowledge of the number of clusters. Hierarchical methods create hierarchical decompositions of the data set and suffer from irreversible merge/split operation decisions. Density-based methods examine the density of points in the neighborhood. The cluster density must be re-computed if the point-set is changed. Grid-based methods are limited by the predefined grids. If a cluster is on the boundary of multiple grid cells, the result can be problematic. Finally, the model-based methods require models based on domain knowledge or prior knowledge of the clusters. For the optimization algorithm discussed here, partitioning methods and model-based methods are easily ruled out due to the lack of prior knowledge about the clusters. Hierarchical methods are error-prone. Density-based and grid-based methods can be improved upon for better performance and accuracy.

3.3 Clustering Behavior of the Genetic Algorithm

The genetic algorithm (GA) is a global optimization heuristic that mimics the evolutionary process of a biological system. It surveys the search space in a random fashion. Every gene represents a high-dimensional point in the search space. During each generation, genes with better fitness survive. A cross-over operation is used to generate offspring from the parents’ genes. Newborns replace the least fit genes. This process iterates until the convergence condition is met. If all the created genes are recorded and visualized as points in the high-dimensional space, after a few generations, the points start to form clusters. The locations of the clusters hint at the local optima in the search space. A sample snapshot of the clusters is shown in figure 3.3. The reason that clusters are formed in the search space is obvious. Around the local optima, the genes usually have better fitness values. These better genes are more likely to be expressed in their offspring. Naturally, they are clustered together.
The traditional genetic algorithm discards the “dead” genes after each generation. Due to the computation cost per evaluation of the objective function and the possible reoccurrence of a gene in a later generation, it won’t be wise to re-compute genes; rather I maintain a library of the genes that have appeared. A gene is essentially a binary; the library is sorted according to the binary value of the gene. The genetic algorithm often takes a long time to converge. Instead of waiting for the genetic algorithm to converge, I terminate it after some clear clusters are formed. The termination condition is discussed in section 3.4. By identifying the clusters and computing the clusters’ bounding boxes, one or more small search spaces can be produced. The subsequent local optimizations are confined within these small search spaces. I assume the global optimum is inside one of those small search spaces. Like other global optimization heuristics, there is no guarantee of finding the global optimum. Yet, experience supports the assumption that this heuristic will lead to the global optimum.
3.4 Clustering Analysis

This section discusses the $2^n$-ary tree structure, GA’s termination condition and the algorithm to identify the clusters in the tree structure. I put the GA’s termination condition in this section because it directly governs the timely construction of the $2^n$-ary tree structure.

3.4.1 The $2^n$-ary Tree Structure

I am proposing a new clustering method using a $2^n$-ary tree structure and connectivity search. This method is designed for high-dimensional clustering analysis. In two-, three-, and four-dimensional space, the $2^n$-ary tree is a quadtree, an octree, and a hextree, respectively. The $2^n$-ary tree can be used in any $n$-dimensional space where $n \geq 1$. Initially, the tree only has a root node and no data points. The root node represents the entire $n$-dimensional space. The data points are inserted into the tree sequentially; the tree nodes have a pre-defined maximum capacity $N_c$. When the insertion of the new point would exceed the node capacity, the node is divided into $2^n$ equal-size children. The children are self-similar to the parent and their side length is one half of the parent’s side length. All the points in the parent node are redistributed into the child nodes according to their location. A count of the points remains in the parent node. The nodes of greater depth usually, but not always, have higher point density than the nodes of a lesser depth. Since the points are not evenly distributed within the parent node, some child nodes may have lower density than their parents even though they have greater depth. Every node’s point density can be computed because its depth $d$ and point count $c$ are known. The depth of the root node is 0. A node’s density $D$ is

$$D = c \times 2^{n \times d}$$

where $n$ is the number of dimensions of the search space.
3.4.2 GA’s Termination Condition

The goal of using a density-based clustering method is to reduce the size of the search space. Let’s define the size of the original search space as 1. Suppose the search space needs to be reduced to $\frac{1}{x}$ of the original search space, where $x > 1$. Denote $S$ as the space taken by a node. The termination conditions are

$$\forall i, D_i > N_c \times 2^n \left(\left\lceil \frac{\log_2 x + 1}{n} \right\rceil - 1\right)$$  \hspace{1cm} (3.6)$$

$$\forall i, S_i < \frac{1}{2x}$$  \hspace{1cm} (3.7)$$

$$\frac{1}{x} > \sum_i S_i \geq \frac{1}{2x}$$  \hspace{1cm} (3.8)$$

All three conditions must be satisfied at the same time. Namely, the density of the nodes must be greater than the threshold defined by equation 3.6. The space taken by a node must be smaller than $\frac{1}{2x}$ and the total space occupied by the dense nodes must satisfy equation 3.8.

3.4.3 Identifying the Clusters

A depth-first traversal of the $2^n$-ary tree is employed to locate all the nodes, called qualified nodes, that satisfy equation 3.6. Every qualified node needs to be assigned a label. Along with the traversal, when a high density node is found, its immediate $3^n - 1$ neighbors are examined. If the neighbor has been visited and is also a qualified node, the neighbor’s label is also assigned to the current node. If there are more than one visited and qualified nodes in the neighborhood, the labels are compared. If all the labels are equal, no action is taken. Otherwise, non-identical labels are marked as equivalent. If none of the neighbors has a label, the current node is assigned the next available new label. The pseudo code for the traversal function is illustrated below.

**procedure** `traverse(treeNode, level)`

**Require:** `treeNode ≠ NULL` and `level > −1` and global variable `label` is initialized and
label graph $G$ is initialized

1: \textbf{if} $\text{Density}(\text{treeNode}) > \text{threshold}$ \textbf{then}

2: \textbf{if} TreeNode has children \textbf{then}

3: \textbf{free}(children)

4: \textbf{mark} TreeNode as terminal node

5: \textbf{mark} TreeNode visited qualified node

6: \textbf{end if}

7: \textbf{initialize} a stack $S$

8: based on TreeNode’s location to find all the immediately adjacent nodes in the search space

9: \textbf{while} $S$ is not empty \textbf{do}

10: $\text{tempNode} \leftarrow S.\text{pop}()$

11: \textbf{if} tempNode is a visited qualified node \textbf{then}

12: \textbf{if} TreeNode hasn’t been assigned a label \textbf{then}

13: assign tempNode’s label to TreeNode

14: \textbf{else}

15: \textbf{if} tempNode’s label $L_1 \neq$ TreeNode’s label $L_2$ \textbf{then}

16: mark an edge in the label graph $G$ that connects $L_1$ and $L_2$

17: \textbf{end if}

18: \textbf{end if}

19: \textbf{end if}

20: \textbf{end while}

21: \textbf{if} TreeNode hasn’t been assigned a label \textbf{then}

22: assign the next unused label to TreeNode

23: add TreeNode’s label to $G$

24: increment label

25: \textbf{end if}
26: end if
27: if treeNode is not a terminal node then
28: for all children do
29: traverse(child)
30: end for
31: end if

After the traversal, a graph of the labels is obtained. Each label is mapped to a vertex on the graph and equivalent labels are connected with a bidirectional edge. Each connected subgraph is assigned a subgraph id. The number of connected subgraphs is the number of clusters.

I use a stack-based algorithm to identify the connected subgraphs. Two stacks are used. Initially, the stacks are empty. The vertices are added according to the following rules. Add all the graph vertices to $S_1$. Remove the top element of $S_1$ and add it to another stack $S_2$. Remove the top element $V$ from $S_2$ and add the vertices that are connected to $V$ to $S_2$. Assign the current subgraph id to $V$. If a vertex has already been visited, it is not added again. Repeat this procedure until the stack $S_2$ is empty. Increment the subgraph id. If the stack $S_1$ is not empty yet, pop the top element from $S_1$ and use $S_2$ to find the next subgraph. Visited vertices are not processed again. All the subgraphs are identified when the stack $S_1$ is empty. The qualified nodes from the $2^n$-ary tree are relabelled with the new set of graph ids. Those $2^n$-ary tree nodes with the same id belong to the same cluster. The clusters’ bounding boxes can be easily computed. All the bounding boxes become the reduced search spaces for the subsequent local optimization. The following pseudo code is accompanying the discussion for clarification.

procedure find connected subgraphs

Require: graph $G$ has been built from the traverse procedure

1: initialize stack $S_1$ and $S_2$
2: push all the graph vertices to $S_1$
3: \( gID \leftarrow 0 \)

4: while \( S_1 \) is not empty do

5: \( temp\text{Vertex} \leftarrow S_1.pop() \)

6: if \( temp\text{Vertex} \) has not been visited then

7: push \( temp\text{Vertex} \) to \( S_2 \)

8: while \( S_2 \) is not empty do

9: \( temp\text{Vertex} \leftarrow S_2.pop() \)

10: mark \( temp\text{Vertex} \) visited

11: \( temp\text{Vertex}'s \) subgraph id \( \leftarrow gID \)

12: for all the vertices \( V_i \) connected to \( temp\text{Vertex} \) do

13: if \( V_i \) has not been visited then

14: push \( V_i \) to \( S_2 \)

15: end if

16: end for

17: end while

18: increment \( gID \)

19: end if

20: end while

My clustering method improves on existing clustering methods. It does not require prior knowledge of the clusters. The density-based approach allows the clusters to have arbitrary shape. The grid-based method and density-based method are among the best existing clustering methods. Unlike the grid-based method, my clustering method is not limited by the predefined grid resolution. The \( 2^n \)-ary tree structure can have finer resolution when the nodes’ densities are high. Compared to the usual density-based methods, introducing new points into the cluster does not require recomputing the point density. Dynamic cluster recomputing is therefore possible.
3.5 The Optimization Method

The optimization method’s two phases are discussed in the following sections. The pyramid representation offers a simplified explanation of this method.

3.5.1 The Optimization Pyramid

![Optimization Pyramid Diagram](image)

Figure 3.4: Optimization pyramid for traditional multi-resolution approach.

An image can be sub-sampled with various resolutions. For different levels of resolution, the search space for image registration is also different. The greater the resolution, the bigger the search space. Therefore, a multi-resolution optimization method for image registration can be visualized as an optimization pyramid. Traditional multi-resolution methods [16, 17] are illustrated in figure 3.4. They usually start with an initial guess or an exhaustive search on the coarse resolution. Based on the optimization results on the previous level, a small search space is defined on the current level. Then, another exhaustive search is carried out on the current level. This procedure continues until the full resolution is reached. In figures 3.4, 3.5, the bold line segments on each level represent the reduced search spaces, whereas the thin line on each level is the full search space on that level.
My new hybrid optimization method uses the cluster-assisted genetic algorithm to reduce the search space on the coarse level and generate a few smaller search spaces. Assuming the global optimum is contained inside one of those small search spaces, I go down the pyramid for a few levels and then do a local optimization on the new level. Unlike the traditional methods, multiple threads of local optimization are performed simultaneously. For each multi-resolution optimization, my algorithm is more aggressive, it jumps down the pyramid a few levels at a time. This decision is due to the simplex method described in section 3.5.2. It can converge faster than the limited exhaustive search. Full resolution images are only used at the very last stage of the hybrid optimization.

3.5.2 The Constrained-Simplex Method

Gradient-based local optimization methods [44] are suitable for many optimization problems; however, they are computationally too expensive for image registrations. The Nelder-Mead downhill simplex method [45] is a good option. It doesn’t require the calculation of derivatives. A simplex is the most elementary $n$-dimensional geometrical entity with $n + 1$ vertices that can be formed in $n$-dimensional space. Each iteration generates a new vertex $v$ for the
simplex. The new vertex must have better $f(v)$ than at least one vertex in the simplex. If not, the simplex is reduced in size. The simplex’s diameter and the $\min(f(v_i))$ get smaller during the iterations. It finally converges when the diameter is reduced below a pre-defined threshold. The Nelder-Mead simplex method assumes an infinite search space without additional constraints. I made a few small modifications so that the local optimization is confined by a small discrete search space defined by a cluster’s bounding box. Due to the discrete search space, the vertices of the simplex must not overlap. Another constraint is the simplex shape constraint. For example, in 2D search space, the three vertices of the 2D simplex must not be co-linear; in the 3D case, the four vertices of the 3D simplex must not be co-planar. Both overlap and shape constraints can be formulated in terms of the determinant of a square matrix. If all the vertices are represented using homogeneous coordinates, a square matrix $M$ with $(n + 1) \times (n + 1)$ elements can be obtained. This matrix $M$ must be non-singular, namely $\det(M) \neq 0$, where

$$M = \begin{pmatrix} v_{1,1} & v_{1,2} & \ldots & v_{1,n} & 1 \\ v_{2,1} & v_{2,2} & \ldots & v_{2,n} & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ v_{n,1} & v_{n,2} & \ldots & v_{n,n} & 1 \\ v_{n+1,1} & v_{n+1,2} & \ldots & v_{n+1,n} & 1 \end{pmatrix} \quad (3.9)$$

and $(v_{j,1}, v_{j,2}, \ldots, v_{j,n}, 1)$ is the $j^{th}$ vertex’s homogeneous coordinates.

### 3.6 Results

I implemented the hybrid algorithm on Windows 2000 using Visual C++ 6.0. The tests were performed on a Pentium IV 1.7GHz machine with 1.5GB memory. The image set contained
938 consecutive image slices from a rabbit brain. Each image has $4000 \times 4000$ pixels. I performed pair-wise registration for the entire set using three methods. The registration was performed between every consecutive image pair. The comparison is shown in table 3.3. The accuracy is based on visual inspection$^3$. The exhaustive search was performed on a sub-sample with a 1:256 sub-sampling rate. In the runtime comparison, the running time of the exhaustive search is multiplied by $2^{16}$ so that it reflects the actual running time with the original image.

Table 3.3: Registration Comparison: A) Exhaustive Search; B) Multi-resolution Method [17]; C) the Hybrid Method

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of mis-registrations</td>
<td>0</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>Average Runtime Ratio, Runtime of C as unit 1</td>
<td>$\approx 2^{16}$</td>
<td>1.8</td>
<td>1</td>
</tr>
</tbody>
</table>

$^3$This is a subjective measure, different people may come up with different results. However, at least qualitative conclusion can be drawn from the results.
Chapter 4

Image Post-Processing

Image post-processing touches up the raw images to improve the images’ visual appearance. Many convolution filters are widely used to enhance the image’s quality. Information about convolution filters is abundant in the literature [46, 47]. This chapter focuses on a special image defect that occurs in the brain digital atlas.

Swiss-cheese-type defects (SCD) in 2D/3D images refer to corrupted small sets of pixels/voxels contained inside an image. The shape defined by the corrupted set can be arbitrary. Unlike image noise, which is a stochastic process, SCD is rather an image error due to various reasons that contribute to the final digital image. Traditional techniques of image interpolation, image enhancement, image restoration, image recovery, image error concealment, etc., do not address such problems. The proposed solution is to patch SCD with realistic sets of pixels/voxels. Assuming the neighborhood of the corrupted set is reliable, interpolation based on the immediately local neighborhood is only applicable to relatively small and smooth corrupted sets. Images of natural scenes often contain similar structures that sparked successful image compression methods. This structure also helps to produce realistic image patches. I offer a solution based on the Harr wavelet transformation and long range correlation. The arbitrary shape of the SCD posed extra complexities. I use Hilbert traversal to transform a 2D/3D image to a 1D signal so that the arbitrary shaped SCD become corrupted 1D segments. The restoration is performed in both 1D and the image’s original 2D/3D space.
4.1 Background

Image interpolation, image restoration, image error concealment, image enhancement, and image reconstruction are related yet different fields. Image interpolation [48, 47, 46] is an important step of image resampling. A discrete image is transformed into a continuous image through image interpolation. The continuous image is subsequently resampled to generate a new image. Resampling is an essential component of many image manipulations, such as image spatial transformations, image quality improvement, etc. It is helpful in correcting image errors that only affect a few pixels/voxels or a smooth area. Image enhancement often deals with noise reduction, contrast control, etc. Image enhancement techniques are designed to manipulate the images so that the images are more vivid and pleasing to the observers. Image reconstruction methods compose new images from existing partial views. Examples are constructing 3D images from 2D image sequences or constructing panoramic images from pictures taken at different angles. Traditional image restoration methods [47, 46, 49] are used to recover the original scene from degraded images. A predominant problem for image restoration is image blurring due to defocusing, air turbulence, optic defects, etc. Image error concealment developed as a result of unreliable still image and video transmission. The concealment methods [50, 51, 52, 53, 54, 55, 56, 57, 58, 59] in the literature share some common properties with the SCD problem discussed in this dissertation.

Still images or video streams transmitted over unreliable communication channels may suffer from bit-errors, packet loss or late arrival. Most of the current image coding standards are block-based and use variable-length code such as Huffman code or arithmetic code. A single bit-error may cause the image to be undecipherable. In a more fortunate case, a single instance of bit-error only causes one to a few corrupted image blocks. The frequency domain bit-error detection and location algorithm [52] is a simple error concealment algorithm. It locates the bit errors and shifts the misplaced image blocks back to their correct locations. The corrupted block is replaced by the previous block in scan order. The patched block
does not look real at all. The maximum smoothness [58] method fills the corrupted image block with the intensity value that is maximally smooth relative to all the surrounding image blocks. An iterative restoration process based on adaptive POCS (projection onto convex set) [56] modelled the concealment problem as a constrained optimization problem. The POCS method is not popular due to the high computational cost. The same author also proposed a multi-directional interpolation [51] method based on the local neighborhood. Gradient voting is used to determine the best interpolation direction and then a weighted average is used to fill in the damaged pixels. Another multi-directional method [54] recursively used a nonlinear filtering scheme to reconstruct the corrupted block from the boundary towards the core. After a few recursive iterations, the corrupted block is filled with estimated image contents. The wavelet-based image concealment method [55] is also seen in the literature. It uses a 2D wavelet transformation to analyze the edge orientation based on the corrupted block’s surrounding blocks. Once the edge orientation is detected, image interpolation is used to compute the patch. There are methods that are specifically designed for discrete cosine transformation (DCT) compressed images. A fuzzy logic approach [53] uses the DC component of each 8x8 surrounding block to interpolate the smooth background of the corrupted block. The high frequency components of the corrupted block are interpolated by fuzzy logic reasoning.

All the above mentioned techniques use the information from the local neighborhood to patch the corrupted block. A recently emerged group of methods [60, 59] use long range correlation to conceal image error. These methods are promising; however, they cannot be easily adopted to solve the SCD problem. Section 4.5.1 will provide a detailed discussion.

4.2 Identify Damage

Damage to an image in the context of this dissertation comes in two flavors. One is the jigsaw shaped rough image boundary (in 3D, it is the rough surface) and the other is the big
block of missing tissue. In order to fix them, they must be identified.

4.2.1 Surface Defects

Surface defects can be identified by using a 3D morphological filter. Appendix A has more detail. Since a 3D morphological filter usually works on binary images\(^1\), I binarize the digital atlas to obtain the original binary mask image. I apply \(n\) rounds of dilation followed by \(n\) rounds of erosion to the binary mask, where \(n\) is a parameter that controls the smoothness of the result in a 3D image. The \(n\) rounds of dilation fill extra voxels into the defects such as dents on the rough surface. After the dilations, \(n\) layers of shell are also wrapped around the original surface. Without eroding away those extra layers of shell, the volume of the digital atlas will be much bigger; therefore, I apply \(n\) rounds of erosion to peel off the extra shells. The voxels filling in the surface defects will remain after the erosions. By subtracting the original image mask from the resulting image, the binary mask for the surface defects can be identified. Traditional image interpolation and image enhancement methods can be used to fix the surface defects.

4.2.2 Swiss-cheese-type Internal Defects

I define a Swiss-cheese-type image defect (SCD) as a connected group of corrupted pixels/voxels. I assume all the pixels/voxels in the SCD are marked as Bad.

\[
SCD = \{v|v = Bad\}
\]  

(4.1)

An SCD also satisfies

\[
\forall v_i, v_j \in SCD, v_i \bowtie v_j
\]

(4.2)

\(^1\)A morphological filter can be used on grayscale images; however, a grayscale morphological filter cannot be used for identifying image defects. I only use the binary morphological filter throughout this dissertation.
where $\triangleright\ll$ denotes connectivity. I assume an 8-connectivity system. Two pixels/voxels $v_i$ and $v_j$ are said to be connected as long as $v_i$ and $v_j$ are linked via a path where every step goes from the current pixel/voxel to its immediate neighbor and all the visited pixels/voxels belong to the same SCD. A SCD’s surrounding pixels/voxels are assumed to be reliable. Taking account of unreliable surrounding pixels/voxels due to causes, such as aliasing, blurring, non-uniform illumination, etc., is beyond the scope of this dissertation.

SCDs are usually missing tissue blocks. They should not be confused with an empty chamber in the brain. Empty brain chambers often extend across several slices and are large. Missing tissue blocks don’t extend across many slices. To find SCDs, some extra steps need to be taken in addition to the 3D morphological filtering. Imagine you are looking at the resulting binary image mask obtained as described in section 4.2.1; the internal defects are the small holes inside the mask. Assume the background voxels are marked true and the mask voxels are marked false. By flood filling [29] the background voxels with boolean false, only the internal defects are still marked true. Following the flood filling, a component labelling, as described in appendix B, is done to identify the individual image blocks. If the block’s volume is bigger than a preset size and the span of the block is bigger than a preset number of slices, then the block is assumed to be a brain chamber. The rest of the blocks are considered to be SCDs.

4.3 Hilbert Traversal

Images, 2D or 3D, are usually perceived or analyzed in their respective number of dimensions either in the spatial domain or frequency domain. Hilbert traversal (as shown in figure 4.1 and figure 4.5II) enables me to transform a multi-dimensional signal to a 1D signal. Due to the locality and slow context change of the Hilbert traversal, the resulting 1D signal maintains excellent local similarity. An example is shown in figure 4.2. If an arbitrarily shaped SCD is traversed by a Hilbert curve, this multi-dimensional signal is transformed
into a number of 1D signal segments. Figure 4.3 is an illustration of the traversal.

Some of the 1D signal segments are short. They may only occupy a few pixels/voxels. Due to the Hilbert traversal’s great locality property, such a small number of corrupted pixels/voxels can be linearly interpolated. A normal raster scan order does not offer such a nice property. A raster scan has an abrupt context change at the end of every scan line. In
addition to that, a raster scan escapes from the current image context too fast; therefore, in raster scan, a linear interpolation for a few missing pixels/voxels may be inappropriate. A Hilbert scan always explores different directions. If a SCD is a missing line in the image, the Hilbert scan will break the line into many small segments; each segment can be easily interpolated from its immediate neighbors. Some of the 1D signal segments are long. Those long segments cannot be linearly interpolated; the restoration algorithm in this case is discussed in section 4.5.2.

A Hilbert traversal can be implemented based on an L-system or state diagrams [61]. The following sections describe those two approaches.

4.3.1 L-system

A Hilbert curve is a Lindenmayer system (L-system) invented by David Hilbert in 1891 [62, 63, 64]. An L-system is a compact way to describe iterative graphics and is created by starting with an axiom. An L-system axiom is usually a sequence of characters. Each
character represents an action. The axiom to build a 2D Hilbert curve is

\[ \begin{align*}
L & \Rightarrow +RF - LFL - FR + \\
R & \Rightarrow -LF + RFR + FL -
\end{align*} \tag{4.3} \]

where \( F \) means “draw a unit line segment in the current direction”, ‘+’ means “turn \( \pi/2 \) radians clockwise”, ‘−’ means “turn \( \pi/2 \) radians counterclockwise”; \( L \) and \( R \) are recursively defined in the axiom and \( L \) is the base axiom. The drawing always starts with the base axiom. To draw a level 0 2D Hilbert curve, \( R \) and \( L \) are replaced by “no action” and the output sequence is

\[ +F - F - F+ \tag{4.4} \]

To draw a level \( n \) 2D Hilbert curve, every occurrence of \( L \) and \( R \) is recursively replaced for \( n \) iterations. Finally, in the output sequence, \( L \) and \( R \) are replaced by “no action” and consecutive “+−” are removed from the output sequence because they cancel out the direction change. The output sequence for a level 1 2D Hilbert curve is

\[ F + F + F - FF - F - F + F - F - F - FF - F + F + F \tag{4.5} \]

To build a 3D Hilbert curve, extra actions are introduced. It’s axiom is

\[ X \Rightarrow \land < XF \land < XF \div F \land >> XF \land F \times >> XF \div F > X \div > \tag{4.6} \]

where \( X \) is the recursion rule, and the symbols “\( \land, <, >, \&,<, \times, \div \)” change the current direction by multiplying the direction vector with a transformation matrix. The matrices for
these actions are defined in equation 4.7.

\[
\times \Rightarrow \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \div \Rightarrow \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \wedge \Rightarrow \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}
\]

\[
\& \Rightarrow \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad < \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}, \quad > \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}
\]

(4.7)

An L-system can be easily realized in a functional language such as ML and Mathematica; however, in an imperative language or object-oriented language such as C, C++ and Java, L-system implementation becomes slightly more difficult. In my implementation, I use a state-diagram-assisted method to realize the 2D and 3D Hilbert traversal. The state diagram approach is derived from the L-system approach, yet it is more intuitive and it avoids computing the current traverse direction. In the 3D case, matrix multiplications are entirely removed. Therefore the state diagram approach is much faster.

### 4.3.2 State Diagram Approach

![State diagram for 2D Hilbert curve](image)

Figure 4.4: The primitive figures of a 2D Hilbert curve, level 1 Hilbert curves formed from the primitives and the state transition diagram for a 2D Hilbert curve generator.

A level 0 Hilbert curve has a limited number of different orientations. Each orientation
Table 4.1: State transition matrix for 2D Hilbert curve generator

<table>
<thead>
<tr>
<th>state \ condition</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>B</td>
</tr>
</tbody>
</table>

has two different traversal orders. For a 2D Hilbert curve, there are only 8 different ways to draw the level 0 curve and these 8 different ways are the 8 different states or primitives to form the level $n$ Hilbert curve. Since a Hilbert curve is recursively formed, when a Hilbert curve is expanded from level $n-1$ to level $n$, each primitive on level $n-1$ is expanded into $2^d$ primitives on level $n$, where $d$ is the dimensionality of the Hilbert curve. The state transition condition is the sequence number in the traversal order. Although there are 8 different states for the 2D Hilbert curve generator, only 4 states actually appeared. Figure 4.4 illustrates the primitives and the state diagram. Such a state diagram can be encoded in a state transition matrix as shown in table 4.1. In the 3D case, there are altogether 48 different states, but only 24 states are used in the state diagram (figure 4.5). The state diagram is coded in the 24 by 8 matrix shown in table 4.2.

4.4 One Dimensional Integer Wavelet Transformation

In the rest of this chapter, I will use a multi-resolution approach to fix SCDs. Wavelet transformation is a natural candidate for multi-resolution methods. This section is a brief introduction about wavelet transformation.

The simplest wavelet transformation is the Harr wavelet transformation [65]. It uses a scale function $\phi(t)$ and a wavelet $\psi(t)$ to represent a function $f(t)$. The representation is the infinite sum

$$f(t) = \sum_{k=-\infty}^{\infty} c_k \phi(t - k) + \sum_{k=-\infty}^{\infty} \sum_{j=0}^{\infty} d_{j,k} \psi(2^j t - k)$$  \hspace{0.5cm} (4.8)
where $c_k$ and $d_{j,k}$ are coefficients to be calculated. The scale function $\phi(t)$ is the unit pulse

$$\phi(t) = \begin{cases} 1, & 0 \leq t < 1 \\ 0, & \text{otherwise} \end{cases}$$ (4.9)

The function $\phi(t - k)$ is a copy of $\phi(t)$, shifted $k$ units to the right. The Harr wavelet is the step function

$$\psi(t) = \begin{cases} 1, & 0 \leq t < 0.5 \\ -1, & 0.5 \leq t < 1 \end{cases}$$ (4.10)

The general Harr wavelet $\psi(2^j t - k)$ is a copy of $\psi(t)$ shifted $k$ units to the right and scaled such that its total width is $1/2^j$.

From a more intuitive and practical point of view, the discrete Harr wavelet transformation transforms a discrete signal into two halves. The first half is the low resolution average and the second half is the high resolution detail. The low resolution half is transformed recursively until it only has one element, the DC component.

The discrete Harr wavelet transformation produces non-integer transform coefficients.
Table 4.2: State transition matrix for 3D Hilbert curve generator

<table>
<thead>
<tr>
<th>state</th>
<th>condition</th>
<th>0</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>A</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>H</td>
<td>H</td>
<td>I</td>
<td>I</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>K</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>N</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>P</td>
<td>Q</td>
<td>Q</td>
<td>A</td>
<td>A</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>O</td>
<td>H</td>
<td>H</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>J</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>N</td>
<td>U</td>
<td>U</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>L</td>
<td>K</td>
<td>K</td>
<td>S</td>
<td>S</td>
<td>O</td>
<td>O</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>U</td>
<td>V</td>
<td>V</td>
<td>B</td>
<td>B</td>
<td>W</td>
<td>W</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>N</td>
<td>D</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>X</td>
<td>X</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>F</td>
<td>O</td>
<td>O</td>
<td>S</td>
<td>S</td>
<td>P</td>
<td>P</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>W</td>
<td>W</td>
<td>F</td>
<td>F</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>G</td>
<td>A</td>
<td>A</td>
<td>R</td>
<td>R</td>
<td>J</td>
<td>J</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>T</td>
<td>X</td>
<td>X</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>I</td>
<td>W</td>
<td>W</td>
<td>F</td>
<td>F</td>
<td>V</td>
<td>V</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>E</td>
<td>R</td>
<td>R</td>
<td>J</td>
<td>J</td>
<td>Q</td>
<td>Q</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>D</td>
<td>N</td>
<td>N</td>
<td>U</td>
<td>U</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>W</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>V</td>
<td>U</td>
<td>U</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>X</td>
<td>T</td>
<td>T</td>
<td>G</td>
<td>G</td>
<td>H</td>
<td>H</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>M</td>
<td>J</td>
<td>J</td>
<td>Q</td>
<td>Q</td>
<td>A</td>
<td>A</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>H</td>
<td>O</td>
<td>O</td>
<td>P</td>
<td>P</td>
<td>K</td>
<td>K</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>R</td>
<td>E</td>
<td>E</td>
<td>X</td>
<td>X</td>
<td>C</td>
<td>C</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>Q</td>
<td>P</td>
<td>P</td>
<td>K</td>
<td>K</td>
<td>S</td>
<td>S</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>S</td>
<td>F</td>
<td>F</td>
<td>V</td>
<td>V</td>
<td>B</td>
<td>B</td>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

My restoration algorithm described in section 4.5.2 computes faster with integer coefficients than the floating point ones. A simple way to obtain integer coefficients is to cast the floating point number to an integer. However, after the forward wavelet transformation, the reverse transformation can no longer reproduce the original image. This implies that the non-defective pixel/voxels can not be faithfully preserved. Even with the non-integer transform coefficients, due to the limited precision of the floating point number, there is still no guarantee that a good pixel/voxel can be faithfully preserved. The integer wavelet
transformation (IWT) is therefore important in the restoration algorithm.

The IWT is derived from the linear wavelet transformation and the linear wavelet transformation is a variation of the discrete Harr wavelet transformation. The background can be found in the reference book [32]. The IWT used here is based on flooring. In the reverse transformation, the truncation effect of the forward transformation due to flooring is considered; therefore, the original data can be exactly reconstructed. Equation 4.11 through equation 4.14 show the forward and reverse IWT.

The original 1D signal is a vector of \( N \) integers \( x_i \), where \( i = 0, 1, \ldots, N-1 \). For simplicity, I assume \( N = 2^m \). After one pass of IWT, the resulting vector \( y \) has two halves. The first half, \((y_0, y_1, \ldots, y_{k-1})\), contains the low resolution average and the second half, \((y_k, y_{k+1}, \ldots, y_{N-1})\), contains the high resolution detail, where \( k = N/2 \).

\[
y_{k+i} = \begin{cases} 
  x_{2i+1} - \lfloor (x_{2i} + x_{2i+2})/2 \rfloor, & \text{for } i=0,1,\ldots,k-2 \\
  x_{2i+1} - x_{2i}, & \text{for } i=k-1
\end{cases}
\] (4.11)

\[
y_i = \begin{cases} 
  x_{2i} + \lfloor y_{k+i}/2 \rfloor, & \text{for } i=0 \\
  x_{2i} + \lfloor (y_{k+i-1} + y_{k+i})/4 \rfloor, & \text{for } i=1,2,\ldots,k-1
\end{cases}
\] (4.12)

The reverse transformation produces a new vector \( z \)

\[
z_{2i} = \begin{cases} 
  y_i - \lfloor y_{k+i}/2 \rfloor, & \text{for } i=0 \\
  y_i - \lfloor (y_{k+i-1} + y_{k+i})/4 \rfloor, & \text{for } i=1,2,\ldots,k-1
\end{cases}
\] (4.13)

\[
z_{2i+1} = \begin{cases} 
  y_{k+i} + \lfloor (z_{2i} + z_{2i+2})/2 \rfloor, & \text{for } i=0,1,\ldots,k-2 \\
  y_{k+i} + z_{2i}, & \text{for } i=k-1
\end{cases}
\] (4.14)

Substituting \( y \) in (4.13),(4.14) by (4.11),(4.12), it is easy to prove that \( \forall i, x_i = z_i \).

For a vector \( x \) with \( N = 2^m \) integers, the IWT runs for \( m \) passes. After the initial pass, only the low resolution half is processed again. The DC component is produced in the last
pass. For each IWT pass, the reverse transformation can reproduce the vector processed by the forward transformation; therefore, this iterative process won’t cause any data lose to the original vector.

4.5 Patching Swiss-Cheese-Type Image Defects

The Hilbert curve and IWT are the tools to fix SCDs. The following sections discuss the long range correlation and the actual restoration algorithm based on the tools.

4.5.1 Long Range Correlation

Adjacent pixels/voxels usually have similar intensities. This local resemblance forms the basis of many image processing techniques, such as image interpolation, image compression, etc. Other than local resemblance, images may also have repetitive patterns. These patterns may not appear in the immediate neighborhood of each other and therefore are considered as long range correlations. This property has been proven true in many image compression techniques. The success of fractal-based image compression [66, 32] is good evidence for the existence of long range correlation.

Searching, copying and pasting are the basic operations of using long range correlation to fix an SCD. I call this a window matching method. As mentioned in the preface of this chapter, the surrounding pixels/voxels of a SCD are assumed to be reliable. These boundary pixels/voxels are the signature of the SCD. If the same signature can be found in another part of the image and the contents contained by the boundary pixels/voxels are not corrupted, then I can copy the good contents and paste them onto the SCD. It may not be possible to find exactly the same signature in the image. The sum of squares of intensity differences, denoted as $S$, can be used as the similarity metric. The best fit is where $S$ is minimum. When a tie occurs, the one which is closer to the SCD wins. The search operation is expensive, especially when the SCD has an arbitrary shape. A primitive method limits
the search range to reduce the run time. Since the search range is small, the best fit may be missed. To resolve this problem, I developed a multi-resolution method which is described in section 4.5.2. In addition to that, I carry out the window matching method in two different spaces. The signature extraction is in 1D space and the search is performed in the image’s original 2D/3D space. The restoration is again performed in 1D and later the 1D signal is transformed back to the image’s original space.

4.5.2 Restoration Algorithm

As described in section 4.5.1, given the SCD’s signature $\sigma$, the restoration algorithm searches for the maximally similar signature $\sigma_{\text{max}}$ in the image. The contents enclosed by $\sigma_{\text{max}}$ are used to fix the SCD. The entire boundary of a SCD can be used as the signature; however, it is computationally expensive to use this signature. In addition, a thin boundary may be unreliable for various reasons. In the literature, to balance the efficiency and accuracy, a thick boundary and a small search range is adopted [59]. Such a method has a $O(n^2m)$ running time, where $m$ is the side length of the defect$^2$ and $n$ is the side length of the image.

When the image containing the SCDs are Hilbert traversed, the SCDs are transformed into 1D segments. The length of the 1D segment is denoted as $L$. For a short 1D segment, where $L < 4^d$ and $d$ is the number of dimensions of the image, the 1D segment is simply linearly interpolated. For a long 1D segment, its signature is extracted and similar ones are searched within the entire image. The $\sigma$ used in the algorithm is a multi-resolution one. Figure 4.6 illustrates a 2D signature example. Extraction of a 1D segment’s signature proceeds as follows.

I The Hilbert traversed 1D segment is IWT transformed.

II The signature is extracted level by level from the highest resolution to the lowest resolution.

$^2$assuming the defect has a square shape.
Figure 4.6: Illustration of the multiple pass IWT with signature extraction. A 2D image is assumed in this example. The original 1D signal is the first two rows of elements. Four IWT passes from the highest resolution to the lowest resolution are shown below the original 1D signal. A) Good pixel/voxel or low frequency component of the image. B) SCD pixel/voxel or low frequency component of the SCD. C) Elements used in the signature $\sigma$. D) High frequency component of the SCD.

III The IWT stops when the 1D segment is reduced to one element.

IV From the highest resolution to the lowest resolution, every $d$ resolution level is used to extract the signature $\sigma$, where $d$ is the number of dimensions of the image. These levels are called candidate levels.

V For each candidate level, the two elements next to the two ends of the segment’s low frequency component are used for $\sigma$.

VI If the length $L$ of the 1D segment doesn’t satisfy $\log_2 L - \lfloor \log_2 L \rfloor = 0$, at the levels where the low frequency component has an odd number of elements, the length of the low frequency component is extended by one element.

To search for the signature in the original image $Img$, $Img$ must be IWT transformed
in $Img$’s native dimensionality. Every signature element has its corresponding level in the $d$-D IWT transformed image. All the signature elements can be reversed to their original locations in $Img$ using a simple state-diagram-assisted method [61]. For every pair of signature elements from the same resolution level, their relative positions can be computed. The search starts at the lowest resolution level, the signature elements at that level are used. The sum of squares of intensity differences is used as the similarity metric. The top $p$ percent of the matches are examined again in the next resolution level. The percentage $p$ can be determined empirically based on the image. Iterate the search procedure from the lowest resolution to the highest resolution. At the highest resolution, only the best match can be selected. When tie occurs, the one with the smallest Euclidean distance to the SCD wins. If there still is a tie, pick one randomly. Assuming $p < \frac{1}{2^d}$, this search method has an $O(\frac{n^2}{m^2})$ running time.

My new restoration algorithm is performed in two different spaces. The signature extraction is done in the 1D space so that arbitrary shaped SCDs can be accommodated. The search is performed in the image’s native $d$-dimensional space. I restore the image by first patching the 1D signal and then transforming it back to the image’s $d$-dimensional space.

4.6 Results

Due to the lack of 3D standard images and the difficulty of visualizing a 3D data set, I demonstrate the results using the 2D images in figure 4.7. The PSNR of the restored Lena and Barbara images are 26.3 dB and 29.4 dB respectively.
Figure 4.7: (a) Original Lena image. (b) Lena image with random SCDs. (c) Restored Lena image. (d) Original Barbara image. (e) Barbara image with random SCDs. (f) Restored Barbara image.
Volumetric Data Set Encoding

Concatenating the aligned and post-processed images, a volumetric data set can be formed. It is necessary to encode the volumetric data set so that it can be effectively used in applications. In this chapter, I present a new data structure, the Scalable Hyper-Space File (SHSF). The SHSF is a generalized data structure that can represent a hyperspace of any dimension. The two-dimensional SHSF is a scalable linear quadtree and the three-dimensional SHSF is a scalable linear octree. Unlike the normal linear quadtree and octree, the data structure uses a scalable linear coding scheme. It recursively uses fixed-length linear code to encode the hyperspace, which is efficient in terms of storage space and accessing speed. The structure lends itself well to pipelined parallel operations in encoding the volumetric data set, so that it enjoys excellent performance even though the huge data set imposes heavy disk I/O requirements. The data structure can provide different levels of detail; therefore it can be used in an environment where the bandwidth and computation power is limited, such as the Internet and slow desktop computers.

5.1 Background

The Octree [67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83] and Quadtree [67, 84, 85, 80, 83] have been widely studied in many areas, such as computer graphics[70, 75, 83], computer vision[68, 69, 76], CAD/CAM[68], medical imaging[72, 74, 79, 81, 82, 83],...
robotics, high performance computing, spatial indexing, clustering analysis, etc. The Octree and Quadtree are decomposition-based data structures that recursively subdivide themselves into non-overlapping, space-filling, equal size, and self-similar subunits until they meet certain criteria. There are a wide range of criteria, depending on the application. For an image processing application, the decomposition stops when the subunits are of a single color. Some applications map their domain-specific measurements onto color space so that they can also use the Octree and Quadtree data structures. The color space can be black and white, different levels of grayscale or even true color. The more generic form is grayscale, because all the other color models can often be transformed into a grayscale representation. In high performance computing, spatial indexing, and clustering analysis, the decomposition criterion is the number of objects residing in the subunit. Whenever the number of objects drops under a pre-defined threshold, the subdivision stops.

In the literature, the Octree and Quadtree can be categorized on the basis of their in-memory representation. The simplest representation uses pointers to link the parent node and the child nodes. This is useful when the tree is relatively small. When the tree grows, the number of tree nodes grows exponentially, and the overhead of the excessive number of pointers prevents the data structure from scaling well. Another common approach uses a linear data structure that avoids using pointers; instead it uses linear code to address the subunits. The linear coding scheme helps to reduce the memory cost and the data can be easily flushed onto the stable storage. The drawback of the linear data structure is that the linkage between the parent node and the leaf nodes does not exist, and therefore it requires an additional search to locate the desired subunit. Even though the linear data structure takes less space than the pointer-based structure, the scalability problem is still not solved, because the linear code grows with the data set. See section 5.2 for details. In addition, the linear structure is a single chunk of data; the search cost also grows with the data set size.

In the area of digital brain atlases, according to Bijan Timsari, digital atlases are
categorized into voxel-based\cite{87, 72, 88, 74, 79, 89, 90, 81, 82, 83} and vector-based\cite{86} models. The early work on voxel-based models earned a reputation for low resolution and low contrast\cite{86} for various reasons. The huge amount of data (Gigabytes or Terabytes) has become the major obstacle in the building of high resolution digital atlases. Researchers\cite{89, 91} are hoping that advances in computer hardware can solve the problem. However, people will have higher resolution image sets when the future computers can handle today’s data. The vector-based models simply can not handle complex images and are difficult to extend to three-dimensional space \cite{92}.

5.2 Data Structure

The novelty of the SHSF data structure lies in the scalable linear coding scheme. The SHSF data structure can handle any n-dimensional space where $n \geq 1$. The discussion here focuses on the three-dimensional SHSF data structure. In the rest of this chapter, when I talk about SHSF, the three-dimensional SHSF data structure is assumed, which is a variation of the Octree data structure. Early work on the Octree\cite{74, 79, 81, 82} used a pointer-based in-memory data structure. The individual octants can be easily located by following the pointers. The pointer-based structure does not scale well because of the overhead incurred by the pointers. For 256-level grayscale image processing, each octant carries a 1 byte payload and 8 pointers. Usually, each pointer requires 4 bytes. Thus the ratio of overhead to payload is 32 to 1. This puts serious constraints on the processable volume size. Linear code\cite{70, 83} (see figure 5.1), also known as location code, reduces the overhead significantly. Each octant can be subdivided into 8 subunits; each subunit requires 3 bits to encode its location relative to the parent octant. The 3 bits are called the direction code\cite{70}. If the maximum depth of a leaf node is $n$, the size $s$ of the linear code is $s = 3 \times n$ bits (see figure 5.1B). To ease the octant encoding and decoding process, researchers tend to use a predefined linear code size for all the octants and the code is also rounded to the next 8 bits (1 byte)\cite{68, 72, 83}.
If the volume’s side length is $1024\ (2^{10})$, it requires at most 10 subdivisions to go down to the smallest leaf octant. The predefined linear code size should be 30 bits, which rounds to 4 bytes. This number increases when the volume grows bigger. In other words, a bigger volume has higher overhead to encode octants. Any changes made to the volume size will cause global reorganization. Let’s denote $D$ as the direction code and $n$ as the depth of the node. The conventional linear code $C$ (in binary) (figure 5.1B shows an example) is formed as follows:

$$C = D_0D_1...D_n,$$

That is, to build the linear code, the direction codes are concatenated. The linear Octree follows a three-dimensional z-order (figure 5.2 illustrates two-dimensional z-order curves) to record all the octants; therefore it has a single chunk of data. This does not scale well because the entire chunk must reside in the memory for most of the Octree operations, such as octant search.

The scalable linear code (SLC) does not use direction codes, instead, it builds itself from the coordinates of the octant’s origin. It has the same addressing capability as the conventional linear code, meaning that the SLC can address the same volume size with the same code length as the conventional linear code. SLC is of fixed length. From the discussion
of the conventional linear Octree, it’s obvious that a fixed length code cannot address the entire volume if the volume size exceeds the code’s addressable space. I introduce another layer on top of the Octree hierarchy. Figure 5.3 is a two-dimensional illustration of the layered data structure. Suppose the SLC can address a $2^s \times 2^s \times 2^s$ volume and the actual volume size is $2^n \times 2^n \times 2^n$, where $n > s$. From the bottom up, I use the SLC to encode all the $2^s \times 2^s \times 2^s$ bottom level cubes. This leaves me a $2^{n-s} \times 2^{n-s} \times 2^{n-s}$ cube with all those bottom level cubes as its units. This process is repeated until the entire volume is encoded. Despite the different unit sizes, all of the $2^s \times 2^s \times 2^s$ volumes are encoded separately and each volume is represented by a chunk of data. The chunk itself is a self contained linear Octree. Within the chunk, the octants are ordered according to the z-order curve. The parent chunk has all the information to address its child chunks. When the SHSF is on stable storage, the parent chunk records the offset and chunk size of the child chunks. Figure 5.3 diagram (4) shows the file organization of the SHSF on the disk. When the SHSF is loaded from the disk, the root chunk is memory resident and the parent chunks record the pointers linking to the in-memory child chunks or file offset linking to the child chunks on the disk. Which chunk is in the memory and when its occupied memory is reclaimed depends on the operation and the operation’s memory access pattern. The SHSF has natural support for different levels
Figure 5.3: Layered data structure of the two-dimensional SHSF. (1) The original image, with all the subdivision lines, is represented as chunk C. Chunk C’s unit length is $\frac{1}{8}$ of the original image’s side length. (2) Chunk C1 represents a subunit of C. Its side length is chunk C’s unit length, which is $\frac{1}{8}$ of the original image’s side length. Its unit length is $\frac{1}{64}$ of the original image’s side length. (3) The two-layer hierarchical representation of the original image. Each chunk forms a self-contained linear Quadtree. The chunks are marked accordingly (C1, C2 and Cn). (4) SHSF file structure. The order of the chunks can be random. However, within each chunk, a strict z-order is preserved. The linear representation of each chunk is marked accordingly.
of detail (LOD). It is unnecessary to load all the chunks at the same time. Compared to the
conventional linear Octree which requires the entire data structure to be memory resident,
the SHSF has better scalability. To improve the performance of the I/O operation, the chunk
size should be close to the disk block size. A one-byte SLC can handle a $2^2 \times 2^2 \times 2^2$ volume.
Each octant in the level 0 chunk uses 2 bytes (1 byte for the SLC and 1 byte for the color
information). Each octant in the level $n$ chunk where $n > 0$ uses 10 bytes (1 byte for the
SLC, 1 byte for the color information, 4 bytes for the child chunk offset and 4 bytes for the
child chunk length). The maximum chunk size for this case is 640 bytes. Table 5.1 shows the
maximum chunk size for different length SLCs. One or two byte SLCs have an advantage
over other SLCs because their corresponding chunk size is close to the disk block size and
the coding overhead\(^1\) is also low.

<table>
<thead>
<tr>
<th>SLC length (byte)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum chunk size</td>
<td>640 bytes</td>
<td>352 KB</td>
<td>192 MB</td>
<td>13GB</td>
</tr>
</tbody>
</table>

If the conventional linear code is divided into a few levels, it seems that it can also be
efficiently used in the SHSF. The SLC requires fewer bit operations and the subdivision
doesn’t need to be tracked for encoding purposes. Suppose the SLC length is $B$ bytes, the
SHSF is $d$ dimensional, and the chunk level is $L$. The SLC $C_{\text{slc}}$ can be represented (in
binary) as:

$$C_{\text{slc}} = C_0 C_1 \ldots C_d,$$

(5.2)

Where

$$C_i = D(i)_{b \times L} D(i)_{b \times L+1} \ldots D(i)_{b \times L+1} D(i)_{b \times L},$$

(5.3)

$$b = \left\lfloor \frac{8B}{d} \right\rfloor,$$

(5.4)

\(^1\)The coding overhead is the ratio between the SLCs and the payload.
\( C_i \) represents the bits taken off the coordinate \( D(i) \) of the \( i^{th} \) dimension. \( D(i)_j \) represents a bit in \( D(i) \). For example, a three-dimensional SLC is two bytes in length. The level 0 code for this SLC is

\[
X_4 X_3 X_2 X_1 X_0 Y_4 Y_3 Y_2 Y_1 Y_0 Z_4 Z_3 Z_2 Z_1 Z_0 ,
\]

(5.5)

It is 15 bits in length. The level 1 code is

\[
X_9 X_8 X_7 X_6 X_5 Y_9 Y_8 Y_7 Y_6 Y_5 Z_9 Z_8 Z_7 Z_6 Z_5 ,
\]

(5.6)

It is also 15 bits in length. See figure 5.1A2 for a one-byte SLC example. A three-dimensional SLC stitches the bits taken from the three coordinates together whereas the conventional linear code concatenates all the direction codes together. The number of bit operations \( N_{\text{slc}} \) required to build the SLC is fixed; however the number of bit operations required to build conventional linear code \( N_{\text{linear}} \) can vary and it is often more than \( N_{\text{slc}} \). No history information is required to build the SLC because it is directly taken off the coordinates. No matter how big the volume is, the SLC always has fixed length. This behavior is critical to the scalability of the SHSF data structure. Table 5.2 compares the size of different variations of the Octree in the worst case.

Table 5.2: In the worst case (all the neighbor voxels are of different colors), the SHSF data structure is the most space efficient. The comparison is based on a 1024 \( \times \) 1024 \( \times \) 1024 volume with a 256-level grayscale color model. A one-byte SLC is used.

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Pointer-based Octree</th>
<th>Linear Octree</th>
<th>Three-dimensional SHSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>37.7GB</td>
<td>5GB</td>
<td>2GB</td>
</tr>
</tbody>
</table>


5.3 SHSF Construction Algorithm

Constructing the Octree data structures from a volumetric data set is time consuming. All the voxels must be visited. Usually, the Octree is built using a recursive algorithm[81] . It can also be built by visiting every voxel in the z-order with the aid of a stack[72, 83] . Both algorithms have $O(n^3)$ running time, where $n$ is the side length of the volume. The SHSF construction algorithm does not improve the running time; however, with the same running time, by using SHSF, additional functionalities and benefits can be obtained for free.

Due to the independence of the octants, the Octree is a good candidate for high performance parallel computing. The pointer-based Octree is easy to parallelize because different sub-volumes can be assigned to different processors and a global reorganization is unnecessary. By adjusting each task’s granularity, the parallelization can be well balanced. The pointer-based Octree, however, is only used in small scale applications. It must be built every time when the volume is loaded and must be linearized before saving onto the disk. This approach is not suitable for big data sets.

Parallelizing the construction of the linear Octree is more complicated than the pointer-based Octree. Because the octants must be recorded in z-order, a global reorganization step is unavoidable[78] . After all the tasks are done, one processor must collect all the resulting fragments and stitch them together in the z-order. This sequential portion of the parallel algorithm has a large impact on the speedup of the linear Octree construction.

The SHSF data structure overcomes its predecessors’ shortcomings. First of all, the SHSF uses the linear data structure and two layers of hierarchy to reduce the space complexity. Second, it does not require any global reorganization during the parallel construction, which significantly reduces the sequential computation time. I use a thread pool and the tasks are recursively dispatched. Each task forms a chunk using four steps. First, the thread waits for the I/O to read the sub-volume into the memory. Second, the thread builds the chunk. Third, the chunk is flushed back onto the disk. Fourth, the file offset and chunk size are
recorded in the upper level chunk if there is one. The upper level task is inactive until its lower level chunks are built and the upper level task is responsible for dispatching its lower level tasks. The lower level tasks are responsible for updating their dispatcher, the upper level task, with the file offset and chunk size. In shared-memory multiprocessing, the lower level tasks will write to the dispatcher’s memory space during the updating. The critical section is short because it only needs to report a few bytes of data. The resulting data file has all the chunks randomly recorded; within each chunk, the z-order is strictly preserved. Other than the disk I/O, there is no sequential portion in the parallel SHSF construction algorithm.

Besides parallelizing the computation, the disk I/O also plays an important role in the parallel algorithm. Due to a possibly huge amount of data, the I/O time is not negligible. To build a pointer-based Octree, the I/O time is:

\[ T_1 = T_{\text{Read Volume}} , \]  

(5.7)

To sequentially build a conventional linear Octree, the I/O time is:

\[ T_2 = T_{\text{Read Volume}} + T_{\text{Write Result}} , \]  

(5.8)

To build a conventional linear Octree using a parallel algorithm, the global reorganization step requires additional I/O time; therefore the total I/O time is:

\[ T_3 = T_{\text{Read Volume}} + T_{\text{Write Result Fragments}} + T_{\text{Read Result Fragments}} + T_{\text{Write Result}} , \]  

(5.9)

To build the SHSF, whether using a parallel or sequential algorithm, the total I/O time is:

\[ T_4 = T_{\text{Read Volume}} + T_{\text{Write SHSF Result}} , \]  

(5.10)
For the completeness of this discussion, I include the I/O time for the pointer-based Octree. However, it is not comparable to the other I/O time analysis because the pointer-based Octree does not consider saving the result onto the stable storage. If the writing time is taken into consideration, the I/O time for the pointer-based Octree will be the same as for the linear Octree. According to the comparison made in table 5.2, it is obvious that $T_{\text{Write SHSF Result}}$ is greater than $T_{\text{Write Result}}$. In addition to that, $T_3 > T_2$; therefore, the I/O time for building the SHSF, $T_4$, is always smaller than those ($T_2$ and $T_3$) for building the linear Octree.

The SHSF data structure provides a natural way to pipeline the I/O during the parallel construction. Because each task takes care of its own I/O, the I/O is well distributed among all the tasks and therefore it is possible to pipeline them. When the thread pool size is one, the execution is sequential. Let’s denote the computing time as $T_c$, I/O time as $T_{I/O}$, number of chunks as $n$. The execution time $T$ is:

$$T = n \times (T_{I/O} + T_c),$$  \hspace{1cm} (5.11)

When the thread pool size is greater than one, the I/O can be pipelined. On a single-processor system, when $T_c \geq T_{I/O}$, the execution time $T$ is:

$$T = n \times T_c + T_{I/O},$$  \hspace{1cm} (5.12)

When $T_c \leq T_{I/O}$, the execution time $T$ is:

$$T = n \times T_{I/O} + T_c,$$  \hspace{1cm} (5.13)

Since there is only one I/O unit and one processor, if the thread pool size is too big, the context switch overhead can slow down the execution. On a SMP system, if the thread pool
size is \( p \), when \( T_c \geq (p - 1) \times T_{I/O} \), the execution time \( T \) is:

\[
T = (T_c + T_{I/O}) \times \frac{n}{p} + (p - 1) \times T_{I/O},
\]

(5.14)

when \( T_c \leq (p - 1) \times T_{I/O} \), the execution time \( T \) is the same as that given by equation 5.13.

### 5.4 Results

I developed and tested the Brain Slicer under Microsoft Windows 2000 Professional edition. It has four major components, namely the volume builder, the SHSF construction program, the surface mesh generator and the virtual slicer. The volume builder takes a sequence of 2D parallel images and builds the 3D image (volumetric data). If there are gaps between two 2D images, the gaps will be filled by using linear interpolation. This component is written in Java. The SHSF construction program constructs the SHSF from the 3D image. It is written in C++ and uses the ORBAcus's JTC (Java-like Threads for C++) thread library for parallelization. The surface mesh generator is also written in C++. It generates the triangular surface mesh from the SHSF. The virtual slicer is a Java program that can slice the digital atlas (an SHSF file) in any oblique direction. One of the image sets that I work on is the rabbit brain image set consisting of 938 coronal section images. The size of each image is 844 by 679 pixels. The volume represented has \( 844 \times 679 \times 938 \) voxels.

#### 5.4.1 Pipelined Result

The SHSF construction program can run on either a single processor system or an SMP system. By pipelining the tasks that build the chunks, the program can yield good performance on both. Figure 5.4 shows timing results. The data for the single processor system was obtained from a Pentium IV 1.7GHz PC with 1G main memory. When the thread pool size is one, there is only one thread that can dispatch tasks, so execution is sequential.
When the thread pool size is greater than one, the disk I/O occurs concurrently with the cpu processing. The optimal thread pool size is two because there is only one cpu and the extra threads can only increase the context switch overhead. The data for the dual-processor system was obtained from a dual pentium III 550MHz PC with 1G main memory. When the thread pool size is one, execution is also sequential (data point 1 in figure 5.4 diagram B). By parallelizing, I was able to get significant speedup compared to the sequential execution. Due to the context switch overhead, the optimal thread pool size is the number of processors on the SMP system. (In the case described here, it is two)

Figure 5.4: SHSF construction time. Diagram A and diagram B are not comparable because they were obtained from different computers. The relative speedup is illustrated in the diagrams. (A) Single processor with single I/O unit, (B) Dual processors with single I/O unit

5.4.2 Slicing Result

Based on the SHSF data structure, I implemented the slicing algorithm[82] in Java. I am able to virtually slice the digital atlas in any oblique direction (see figure 5.5). The oblique section can be at different levels of detail. The deeper you go down the SHSF levels, the better the resulting image (see figure 5.6); however, as you go down the levels, the running time grows exponentially. Most of the time, it is unnecessary to go down to the very bottom
level of the SHSF data structure. As shown in figure 5.6, the image in diagram C is already good enough. For an extremely big volume, when you zoom in, the resulting image is likely to be bigger than the viewable area on the screen, and only a small portion of the image will be displayed. In this case, I only need to selectively visit limited chunks on limited levels to obtain the section image. This can help to reduce the slicing time. SHSF’s natural LOD support is extremely useful in an environment where the bandwidth and computation power is limited, such as the Internet or slow desktop computers. I tested the Brain Slicer on a web browser. It can provide coarse images with acceptable resolution through the Internet.

5.4.3 Triangular Surface Mesh

To generate the triangular surface mesh for the digital data, the atlas’s internal structure is useless. I only care if a voxel is inside the 3D object or outside the 3D object. The black and white color model is therefore more efficient in surface mesh generation. Figure 5.7 illustrates the idea. Obviously, the SHSF with only black and white is smaller than the SHSF with full grayscale details. The parallel section image sets went through a binarization process.

Figure 5.5: Images of the oblique sections (A) Oblique section with plane normal equals to $(0.577,-0.577,0.577)$, (B) Oblique section with plane normal equals to $(0,-0.707,0.707)$
Figure 5.6: Example sagittal section images generated with different levels of detail. From (A) through (D), the detail level increases one level at a time.
Figure 5.7: Image binarization (A) Original image, (B) Quadtree representation of the original image, (C) Binarized image, (D) Quadtree representation of the binarized image

Figure 5.8: Triangular surface mesh of a rabbit brain
prior to the SHSF construction. By traversing the SHSF data structure, a set of cubes that intersect the surface of the 3D object are recorded. Those cubes are of equal size; the size of the cube is determined by the desired surface mesh quality. The mesh generation algorithm only needs to go down to the SHSF level where the cube size is just satisfied. I use the well known marching cube[93] algorithm to build the triangular surface mesh. Figure 5.8 shows the result for a rabbit brain atlas.
Chapter 6
Volume Compression

The voxel-based digital atlases, volumetric data sets, can have raw data of up to a few hundred gigabytes. Given such big data sets, a data compression method is necessary. There are always arguments that the capacity of computer hardware is growing at a rapid pace and therefore people can always buy a bigger hard drive to accommodate the data. This statement is short sighted because the data set is also getting bigger and the new hardware offers the opportunity to study more species’ brain images. Good compression methods are always needed. When I say good compression methods, I am referring to a few important characteristics. Obviously, fidelity is number one. Compression rate\(^1\) is number two. Compression and decompression speed is number three. Compression speed sometimes is not as important as the decompression speed, because decompression is carried out more frequently. Increasing the compression/decompression speed is sometimes more important than increasing compression rate\(^2\). Other characteristics, such as dynamic access, progressive transmission, etc., are more application related. For digital atlases, dynamic access is crucial because decompressing a huge data set prior to each use seems to be impractical.

\(^1\)Compression rate reflects the amount of data size reduction. In this dissertation, I use bpp (bits per pixel) and bpv (bits per voxel) as the measure of compression rate.

\(^2\)The increased compression rate or the better compression rate refers to smaller bpp or bpv.
6.1 Background

General data compression methods can be put in four categories; namely, the transformation-based methods, the dictionary-based methods, statistical methods and heuristic methods. Transformation-based methods usually transform the data from one domain into another domain so that the data is more easy to directly compress or compress after quantization. This class of methods is often used in image compression. Dictionary-based methods form a dictionary of repetitive patterns while scanning through the data. The repetitive patterns are replaced by a much smaller dictionary word, a “token”, to reduce the size of the data set. LZ77, LZ78 and LZW [32] are a few popular algorithms. Some dictionary methods are adaptive, meaning the dictionary will change as the algorithm scans through the data file. Dictionary-based methods are often used for text data compression and general purpose data compression. Common compression software such as GZIP, COMPRESS, PKZIP, and WinZip all use some form of dictionary-based method. These compress computer files disregarding the type of the data source. Statistical methods refer to Huffman Coding [94, 95] and Arithmetic coding [96]. These two methods, along with their variants, examine the statistical information about the data source and assign variable code lengths to different symbols. The symbols with high probability of occurrence are assigned short codes while the symbols with low probability are assigned long codes. Statistical methods are not usually used by themselves, rather they are combined with other data compression methods such as the transformation-based methods. Heuristic methods are the methods that can’t be categorized into the previous three categories. Heuristic methods include run length encoding, gray code, scalar quantization, etc. These methods are not for general purpose and have limited usage in data compression applications.

Image compression has been extensively studied for more than two decades. Many compression methods, lossy or lossless, were developed for 2D images. Various standards such as JPEG, GIF, PNG have emerged. In spite of all this work, few volume compression methods
have been developed.

### 6.1.1 Lossless Compression

Lossless compression emphasizes the fidelity of the compression algorithm. Methods often employ Differential Pulse-Code Modulation (DPCM) to transform the highly correlated original data into a decorrelated form. The decorrelated form has a smaller entropy than the original data. After DPCM, the new data stream is compressed using statistical methods such as Huffman coding and Arithmetic coding. Lossless JPEG is an example of 2D image compression. Fowler [97] built a program called COMPVOX that applies differential pulse-code modulation (DPCM) followed by Huffman coding. This method uses linear voxel prediction and records the prediction errors, so that the source entropy is significantly reduced. However, COMPVOX assumes wide sense stationary and zero mean of the prediction error. These assumptions are usually not true and hence left room for further compression. General purpose compression methods can also be used for lossless compression. However, when they are used to compress image data, they often fail to efficiently decorrelate the spatial redundancy within the image source. In addition, general purpose compression methods can’t facilitate image processing applications that use the compressed image data.

### 6.1.2 Near-Lossless Compression

Near-lossless compression is a compromise between lossless and lossy compression. A few 2D near-lossless compression methods[98] have been developed. Near-lossless methods are designed to offer better compression fidelity as well as faster speed than the lossy ones. Compared to lossless methods, the near-lossless ones have better compression rates. I haven’t seen any near-lossless volume compression methods yet. After all, volume compression is a younger field than 2D image compression.
6.1.3 Lossy Compression

Lossy compression methods [99, 100, 101, 102, 103, 104, 105, 106, 66] are fighting the battle to maximize the compression rate while minimizing the image quality loss. Compression speed is also a major competing factor among the different methods. A few basic models are the Laplacian pyramid model [104, 105], fractal model [66], and spatial-to-frequency transformation model [106, 101, 100, 102, 103]. The first two models are transformations within the spatial domain, whereas the third one transforms the image from the spatial domain to the frequency domain. DCT, FFT and WT are among the top choices for spatial-to-frequency-domain transformations. For most lossy compression methods, the transformation is usually followed by a quantization step and then the quantized data is entropy coded. The quantization step introduces irreversible changes. Some transformation methods can also introduce such changes. Although the original image can’t be fully recovered and unwanted artifacts may be introduced [100], many scientific and biomedicine applications use lossy compression for purposes that require lower fidelity.

6.2 Lossless Compression

The lossless compression method discussed in this section was previously published in my paper [8, 9]. I consider a 3D image $D$ stored in raster order. $\widetilde{D}$ represents the compressed image data. $V(i,j,k)$ is a voxel of $D$, $f_c$ is the encoding function and $f_d$ is the decoding function. Data compression can be denoted as $\widetilde{D} = f_c(D)$ and decompression can be denoted as $D' = f_d(\widetilde{D})$ where $D'$ is the decompressed 3D image data. I say that $f_c$ is lossless iff $D = D'$.

6.2.1 Non-linear DPCM method

Similar to COMPVOX [97], I also use DPCM and Entropy coding. The difference between COMPVOX and my method resides in the DPCM and the unique data structure. The data
structure will be discussed in the following sections. Assuming the neighboring symbols in the data source are independent, according to information theory, on average, the minimum number of bits required to encode a symbol can be expressed as its entropy $H$, which is

$$H = - \sum_{i=1}^{n} P_i \log_2 P_i$$

(6.1)

where $n$ is the number of distinct symbols, $P_i$ is the $i^{th}$ symbol’s probability of occurrence in the data source. Huffman coding and arithmetic coding are the most popular entropy coding methods. I use Huffman coding due to its simplicity and the fact that it is in the public domain. Arithmetic coding has some marginal improvements over Huffman coding; however, research [95] showed that Huffman coding is still one of the top choices in entropy coding. Entropy coding can exploit the statistical redundancy in the data source. However, it does not reveal the spatial redundancy in an image data source. As with 2D images, neighboring voxels tend to be highly correlated in 3D images. Decorrelation of the image data can further reduce the redundant information.

I use a context-based non-linear lossless DPCM to process the image data before using Huffman coding to losslessly compress the data. In DPCM, a predictor is used to predict the current voxel’s value based on the voxels that have already been visited. Usually, the prediction error is clustered close to zero and the image of the prediction error has lower entropy than the original image. COMPVOX [97] uses a linear predictor that linearly combines the visited neighbor voxels’ values to predict the current voxel’s value. A linear predictor is simple and easy to implement. However, the prediction is not effective, especially when it encounters high frequency components, such as edges, in the image. Optimal linear predictions often rely on computing the optimal prediction coefficients. However, they fail to improve the prediction much further, because they are usually designed to minimize the mean square value of the prediction errors (MSPE). According to Memon [107], minimizing MSPE is useful for lossy compression but not lossless compression, because minimizing the
entropy of the prediction error is more desirable for lossless compression.

My non-linear DPCM is a heuristic method. In a nearly uniform area of an image, the value of a voxel is highly correlated to all of its neighbors and therefore linear predication works well. If the voxel belongs to a high frequency image component, the prediction error based on the linear combination of its neighbors can be big. A large number of bad predictions can lead to a large entropy of the prediction errors. In video compression, the pixel of the same location on the previous picture can be similar to the pixel on the current picture. Even though the pixel belongs to the high frequency component of the 2D image, in the third dimension, it can still belong to a low frequency component. Inspired by this, my heuristic is that a voxel that belongs to a high frequency image component in one direction can belong to a low frequency image component in another direction. By picking the right direction for prediction, I can effectively reduce the prediction error.

The volume is visited following a raster order. For voxel \( V(i, j, k) \), I want to predict its value from the visited voxels. Along the X-axis, I calculate the differences between four pairs of voxels and denote them as \( dx_0 \) through \( dx_3 \). The floor of their average is denoted as \( dx \). Floating point calculations are avoided for efficiency.

\[
\begin{align*}
    dx_0 &= V(i - 1, j + 1, k - 1) - V(i, j + 1, k - 1) \\
    dx_1 &= V(i - 1, j, k - 1) - V(i, j, k - 1) \\
    dx_2 &= V(i - 1, j - 1, k - 1) - V(i, j - 1, k - 1) \\
    dx_3 &= V(i - 1, j - 1, k) - V(i, j - 1, k) \\
    dx &= \left\lfloor \frac{1}{4} \sum_{i=0}^{3} dx_i \right\rfloor \quad (6.2)
\end{align*}
\]

Along the Y-axis and Z-axis, I also calculate the differences between four pairs of voxels and denote them as \( dy_0 \) through \( dy_3 \) and \( dz_0 \) through \( dz_3 \). Their average’s floors are \( dy \) and \( dz \).

\[
\begin{align*}
    dy_0 &= V(i - 1, j - 1, k - 1) - V(i - 1, j, k - 1) \\
\end{align*}
\]
For each direction, if the standard deviation of the difference between the voxel pairs is big, it is likely that in this direction the current voxel belongs to a high frequency image component. Therefore, I pick the direction with the smallest standard deviation of the differences between the voxel pairs. If more than one direction has the smallest standard deviation, I use a linear predictor to combine them and each direction is assigned the same weight. Let’s denote $\epsilon$ as the prediction error. The following is the pseudo code to calculate $\epsilon$.

**procedure** DPCM($V$)

**Require:** $V$ is initialized and contains the volumetric data

1: Compute $dx_0$ through $dx_3$ and $dx$

2: Compute the standard deviation $Sd_x$ of $dx_0$ through $dx_3$

3: Compute $dy_0$ through $dy_3$ and $dy$

4: Compute the standard deviation $Sd_y$ of $dy_0$ through $dy_3$

5: Compute $dz_0$ through $dz_3$ and $dz$

6: Compute the standard deviation $Sd_z$ of $dz_0$ through $dz_3$

7: if only one direction has the smallest $Sd$ then

\[
\begin{align*}
dy_1 &= V(i, j - 1, k - 1) - V(i, j, k - 1) \\
dy_2 &= V(i + 1, j - 1, k - 1) - V(i + 1, j, k - 1) \\
dy_3 &= V(i - 1, j - 1, k) - V(i - 1, j, k) \\
dy &= \left\lfloor \frac{1}{4} \sum_{i=0}^{3} dy_i \right\rfloor \\
dz_0 &= V(i - 1, j - 1, k - 1) - V(i - 1, j - 1, k) \\
dz_1 &= V(i, j - 1, k - 1) - V(i, j - 1, k) \\
dz_2 &= V(i + 1, j - 1, k - 1) - V(i + 1, j - 1, k) \\
dz_3 &= V(i - 1, j, k - 1) - V(i - 1, j, k) \\
dz &= \left\lfloor \frac{1}{4} \sum_{i=0}^{3} dz_i \right\rfloor
\end{align*}
\]
Figure 6.1: Prediction results with histogram: The prediction is applied to a volumetric data set. In order to show the prediction result, I extract 2D section images from the volume. (A) An image extracted from the original volume, (B) The error image for the linear prediction result, (C) The error image from the context-based non-linear DPCM prediction.

8: Use that direction to calculate $\epsilon$

9: \[ \text{if X-Axis has the smallest } Sd \text{ then} \]

10: \[ \epsilon = V(i, j, k) - V(i - 1, j, k) + dx \]

11: \[ \text{end if} \]

12: \[ \text{if Y-Axis has the smallest } Sd \text{ then} \]

13: \[ \epsilon = V(i, j, k) - V(i, j + 1, k) + dy \]

14: \[ \text{end if} \]

15: \[ \text{if Z-Axis has the smallest } Sd \text{ then} \]

16: \[ \epsilon = V(i, j, k) - V(i, j, k - 1) + dz \]

17: \[ \text{end if} \]

18: \[ \text{end if} \]

19: \[ \text{if two directions have the smallest } Sd \text{ then} \]

20: \[ \text{Use the combination of those directions to calculate } \epsilon \]
if X and Y-Axis have the smallest $Sd$ then
\[ \epsilon = V(i, j, k) + \frac{1}{2} \times [-V(i - 1, j, k) + dx - V(i, j + 1, k) + dy] \]
end if

if X and Z-Axis have the smallest $Sd$ then
\[ \epsilon = V(i, j, k) + \frac{1}{2} \times [-V(i - 1, j, k) + dx - V(i, j, k - 1) + dz] \]
end if

if Y and Z-Axis have the smallest $Sd$ then
\[ \epsilon = V(i, j, k) + \frac{1}{2} \times [-V(i, j + 1, k) + dy - V(i, j, k - 1) + dz] \]
end if

if all directions have the same standard deviation $Sd$ then
\[ \epsilon = V(i, j, k) + \frac{1}{3} \times [-V(i - 1, j, k) + dx - V(i, j + 1, k) + dy - V(i, j, k - 1) + dz] \]
end if

Figure 6.1 illustrates the results of my heuristic method. Because the 8-bit grayscale value must be between 0 and 255 while the prediction error is between -256 and 255, for the figure, I added 128 to the value and displayed the 0 to 255 range. From observation, I found the new DPCM method is also zero-mean and the error distribution is close to a Laplace distribution. In addition, my non-linear prediction method has smaller standard deviation and the mean is closer to zero. In the rest of the chapter, I assume those two properties. My heuristic method is not limited to the three orthogonal directions; however, based on the experimental results, adding on other directions yields limited improvement in the prediction accuracy and increases the running time unacceptably.

6.2.2 Different Types of Volume Blocks

DPCM as an encoding method is useful; however, it has a disadvantage that the entire volumetric data set must be decoded prior to accessing it. If the volume is extremely big
and people are only interested in a small portion of the data set, an auxiliary data structure
is desirable. I subdivide the volume into equal size blocks and use a Scalable Hyper-Space
File (SHSF) (see section 5.2 and [6, 7]) to index those blocks. Since the SHSF is essentially
a variant of the octree, in the rest of the chapter, I use the terms SHSF and scalable octree
alternatively for the reader’s convenience and also to ease the discussion. Since each block is
DPCM encoded separately, the intuitive approach is to build a Huffman tree for each block,
which can introduce excessive overhead. I developed a novel binning method to significantly
reduce the overhead. In this section, I start the discussion by differentiating the volume
blocks.

The volumetric data set is subdivided into blocks that can be classified into three types.
Those three types of blocks have different properties and should be treated differently.

The first type is the background block. It has a pure background color and therefore
doesn’t carry much useful information. If the background block is DPCM coded and then
fed to the Huffman coder, the compressed size is about $\frac{1}{8}$ of the original block size, assuming
I am using the 256-level grayscale color model. If I am using another color model and the
symbol’s code length is $n$ bits, the compressed size is $\frac{1}{n}$ of the original block size. This is
inefficient because the entire background block can be coded in only a few bytes. Efficiency
can be gained by using the auxiliary scalable octree, in which the entire background block
is reduced to a terminal node.

The second type of block is on the surface of the encoded 3D object. These blocks have
a big portion of background voxels of uniform color. Since the background voxels are always
grouped together, in the DPCM coded data, I can expect long runs of 0. The Huffman coder
and other entropy coders can only reduce the statistical redundancy. An ad hoc approach,
such as run length encoding combined with Hilbert order scanning [83, 61], can help to
further reduce the coding redundancy.

The third type is the internal block, which contains most of the volume’s internal detail.
Neither the auxiliary scalable octree nor the ad hoc approaches can help in this situation.
Therefore, for the internal blocks, I only use DPCM coding followed by Huffman coding to compress those blocks. In the actual object-oriented implementation, the second and third types of blocks are processed using the same volume compressor class.

6.2.3 Encoding with Multiple Huffman Trees

Huffman code [94] produced by a Huffman tree is a prefix code which is uniquely decipherable [108]. A Huffman tree is a positional tree. A general positional tree can be a $\sigma$-ary tree [108]. For clarity, all the following discussions and proofs only consider the binary case, namely $\sigma = 2$. Let’s denote the symbol distribution by $D_s$, the Huffman tree by $t$, each symbol $c_i$’s probability by $P_i$, and its prefix code’s length by $l_i$. The Huffman tree is a function of the symbol distribution, namely $t = t(D_s)$. The average code length $\bar{l}$ is a function of the symbol distribution $D_s$ and the Huffman tree $t$, therefore it can be represented as

$$\bar{l} = l(D_s, t(D_s)) = \sum_{i=1}^{n} P_i l_i \quad (6.5)$$

**Definition 6.2.1.** A prefix code is optimal if its average code length $\bar{l}$ is minimum

$$\bar{l} = \min \quad (6.6)$$

**Definition 6.2.2.** Let a symbol distribution $D_s$ have a set of symbols $S_{D_s} = \{c_1, c_2, ..., c_n\}$. Define $D_s \geq D'_s$ iff $S_{D_s} \supseteq S_{D'_s}$.

**Lemma 6.2.3.** A Huffman tree $t$ built from a symbol distribution $D_s$ can generate an optimum prefix code for $D_s$.

$$\bar{l} = l(D_s, t(D_s)) = \min \quad (6.7)$$

In other word, $\forall t' = t(D'_s)$ where $D'_s \geq D_s$. If $D_s$ is encoded using the prefix code generated
from $t'$ then

$$t = \ell(D_s, t(D'_s)) \geq 1$$  \hspace{1cm} (6.8)

Lemma 6.2.3 and its proof can be found on page 74 through 79 of *Graph Algorithms* [108]. Arbitrary decisions made in constructing the Huffman tree affect the individual codes but not the average size of the codes [32].

Assuming the Huffman coding overhead is ignorable, it is easy to prove that subdividing the volume into multiple blocks, building one Huffman tree for each block, and compressing the volume on a block basis can yield a better compression rate than treating the volume as a whole. Denote the size of the Huffman coded block by $s$ and the block by $b$. Let the number of voxels in $s$ be $n_v$. Let $s$ be a function of $b$ and a Huffman tree $t$, $s = s(b, t) = \ln n_v$.

Given a volume $b$, I can partition it into $n$ small blocks $b_1...b_n$ where $b = \sum_{i=1}^n b_i$ and for each block there is a corresponding Huffman tree $t_1...t_n$.

**Lemma 6.2.4.** For any volume $b$ subdivided into $n$ smaller blocks $b_i$ where $i = 1, ..., n$, then

$$s(b, t) \geq \sum_{i=1}^n s(b_i, t_i)$$  \hspace{1cm} (6.9)

Proof: According to lemma 6.2.3, the Huffman tree $t_a$ is a binary positional tree which generates optimum prefix code for a symbol distribution $D_s$. The Huffman tree $t'_a$ for a different symbol distribution $D'_s (D'_s \geq D_s)$ may not be optimal for $D_s$. Given the Huffman tree $t$ for volume $b$, for every Huffman tree $t_i$ and corresponding $b_i$, $s(b_i, t_i) \leq s(b_i, t)$, therefore

$$s(b, t) = \sum_{i=1}^n s(b_i, t) \geq \sum_{i=1}^n s(b_i, t_i). \text{ QED.}$$

**Lemma 6.2.5.** Disregarding the Huffman code book size, the best compression rate is obtained when the volume is divided into voxels.

I can prove lemma 6.2.5 by recursively applying lemma 6.2.4 to the volume. In real life, the size of the Huffman code book cannot be ignored. The Huffman code book is stored
as a serialized Huffman tree. It can be easily obtained by a tree traversal. The size of a Huffman code book is $S_{\text{codebook}} = C \times n_{\text{symbols}}$ where $C$ is a constant. Depending on the implementation, $C$ can be slightly different. When the subdivision is too fine, the Huffman coding overhead is going to exceed the original volume size because each block has its own Huffman code book attached to the block’s encoded data stream. It is therefore important to choose a suitable size for the subdivided volume blocks. From the discussion above, it is obvious that if I can efficiently avoid the coding overhead, a better compression rate can be achieved.

The internal blocks are encoded using my non-linear DPCM followed by Huffman coding of the prediction error. It is difficult to model the symbol distribution (histogram) of the original volume; however, after DPCM, the volume’s contents are replaced by the prediction error, which has a Laplacian distribution:

$$P(x) = \frac{1}{2b}e^{-|x-\mu|/b}$$  \hspace{1cm} (6.10)

where $\mu$ is the population mean; in this case $\mu \approx 0$. For two blocks, if $\Delta b \leq \xi$ (where $\xi \rightarrow 0$), then those two blocks are similar.

The similarity between different blocks’ error distributions offers me a way to reduce the Huffman coding overhead. I can group blocks with similar symbol distribution together and build one Huffman tree for each group of blocks. The Huffman tree is built based on the symbols’ probabilities $P_i$. A symbol’s Huffman code length $l_i$ can be approximated as

$$l_i \approx -\log_2 P_i$$  \hspace{1cm} (6.11)

The same symbol in different blocks may be assigned to a different Huffman code, even when its probabilities in the different blocks are close. This property is not my concern because I don’t care about the actual code assigned to the symbol; instead, I only care how many bits
Figure 6.2: Visualization of the binning method: (A) The three projected views of the three-dimensional binning space. (B) The three-dimensional view of the binning space.

are assigned to that code. As long as the code length doesn’t change, it is not going to affect the compression rate. Substituting equation 6.10 into equation 6.1 and setting $\mu = 0$, I get

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i$$

$$= -\sum_{x} \frac{1}{2b} e^{-|x-0|/b} \log_2 \frac{1}{2b} e^{-|x-0|/b}$$

$$\approx -\int_{-\infty}^{\infty} \frac{1}{2b} e^{-|x|/b} \log_2 \frac{1}{2b} e^{-|x|/b} dx$$

$$= \frac{\ln 2 + \ln b + 1}{\ln 2}$$

$$b = e^{H \ln 2 - \ln 2 - 1}$$

The entropy $H$ can be easily obtained by counting the symbols and then plugging into equation 6.1 and then $b$ can be calculated by substituting $H$ in equation 6.13. I label bins with different $b$ values and put the different blocks into the bins. In the actual implementation, I use a slightly different binning method that is more efficient and more suitable for real
data. In the Laplace distribution, most of the symbols are concentrated close to \( \mu \). I use the probability of symbols 0, \( \pm 1 \), \( \pm 2 \) to characterize the distribution. I build a three-dimensional space and use the probability of symbol 0 to label dimension 1 and use the sum of the probabilities of symbol \( \pm 1 \), the sum of the probabilities of symbol \( \pm 2 \) to label dimension 2 and dimension 3. For each dimension, I use a logarithmic scale to divide the space thus formed into a grid of rectangular units. Each unit is considered a bin. Every volume block is therefore abstracted as a point in three-space. All the blocks that fall in the same bin are considered to have a similar symbol distribution. I recalculate the symbol distribution and build a Huffman tree for each bin. Figure 6.2 is a visualization of the binning method. The size of the test volume is \( 1024 \times 1024 \times 1024 \) and the volume block size is \( 32 \times 32 \times 32 \). I was able to significantly reduce the Huffman coding overhead (figure 6.4A). To determine the size of the volume blocks, there are two factors to consider. If the volume block is too big, the compression potential is not fully explored. If the block is too small, it is not efficient in terms of disk I/O. I choose 32 as the volume block’s side length in order to balance those two factors.

### 6.2.4 RLE and Hilbert Volume Mapping

The basic version of RLE, run length encoding, explores long runs of repetitive symbols and records a tuple which consists of an escape symbol, the run length and the repeated symbol itself. When the source data contains long runs of symbols, RLE can be useful for compression. However, RLE also has its limitations; for example, it doesn’t address the statistical redundancy which entropy coding can handle. A combination of both can be used on the surface volume blocks mentioned in section 6.2.2. The conventional way to traverse the volume is raster scan. It is conceivable that when the traversal path moves from the background to the object’s surface, the current run of background voxels will be cut off. This effect results in many runs of background voxels not long enough to be useful for RLE coding. Using a Hilbert space-filling curve can help to obtain longer runs of background
voxels, because the volume traversal following the Hilbert curve has better locality. After DPCM coding, the volume block stores the prediction error, which is traversed and serialized following a Hilbert curve. The serialized data is fed to the Huffman coder with a built-in RLE coder. The resulting data from the Huffman coder is stored on the disk.

### 6.2.5 Scalable Octree

The octree and its variants [6, 7] have been investigated by many researchers throughout the years. It is quite useful in medical imaging. However its usage is limited for a number of reasons. The octree has a strict grid-like structure. All the voxels in the same octant must have the same color. An image of a natural scene can hardly meet this requirement and therefore results in a huge octree with many octants containing only one voxel. For each octant, there is a fixed coding overhead. If the octree is a pointer-based in-memory tree, the coding overhead mainly consists of 8 pointers. If it is a linear octree, the linear code associated with the octant adds on extra bits. The recursive subdivision structure of an octree allows the algorithm to extract partial volume data efficiently, because only a limited number of branches need to be visited. If the entire volume must be processed, there is little gain and sometimes it even complicates the processing. The octree’s worst case space complexity is $O(n^3)$, where $n$ is the side length of the volume. When $n$ is small, the tree is not going to take much space. I treat each volume block as the smallest unit (voxel) and build an octree of the volume blocks so that I can have easy access to a part of the volume. On the other hand, many pure background volume blocks can be reduced to octant representations that only occupy a few bytes. In some cases, many neighboring pure background volume blocks can be combined together to form bigger octants and therefore further benefit the volume access and compression. The conventional way of storing the volume data is sequential and a raster scan is the most common approach. In the data compression context, the sequential scan means compressing and decompressing the entire volume as a whole. When the volume data set grows beyond a certain limit, this is undesirable. An octree can be used as an index
structure for the volume data set which makes it possible to access only part of the volume. The volume data set therefore doesn’t have to be totally decompressed before use. Instead, it can be accessed dynamically when certain volume blocks are needed. In a system with limited main memory, by the same reasoning, the compression and decompression can be done out of core. In addition to the dynamic access, the compression and decompression can both be parallelized. During compression, the volume blocks can be compressed in parallel and the compressed volume blocks can be flushed to the disk out of order. The octree will keep track of the location and length of the compressed volume blocks. I use the SHSF data structure, an octree variant, in the actual implementation, so that the linear code length is fixed and the implementation is more scalable. The essential idea behind SHSF is the multiple levels of redirection in the octree structure. Chapter 5 has more details about SHSF.

6.2.6 Lossless Compression Algorithm Analysis

The compression has the following steps. The first step is to form the octree index structure. The recursion stops when the octant size is equal to the predefined volume block size. Let’s denote the side length of the volume block by $m$, the number of volume blocks on each side by $n$, and the side length of the entire volume by $N$, where $N = n \times m$. The number of recursions $r$ employed to build the octree is

$$r = \log_{2}n \approx \frac{8}{7}n^{3} \quad (6.14)$$

For each volume block, to determine whether the block has detail or homogeneous color requires $O(m^{3})$ time. The total running time $t$ for the first step is

$$t = \frac{1}{7}n^{3} + n^{3} \times m^{3} = \frac{1}{7}n^{3} + N^{3} \quad (6.15)$$
which is $O(N^3)$. The second step is the non-linear DPCM coding. The number of different error symbols are counted while visiting each voxel. Despite the complex prediction mechanism, each voxel is visited exactly once; the time complexity of the DPCM step is $O(N^3)$. In the third step, the prediction error is serialized following a 3D Hilbert curve. The run length information is recorded along the traversal. Similar to building an octree, the Hilbert traversal is also a recursive process. For each volume block, the number of recursions $r \approx \frac{2}{3}m^3$. In the worst case, I still need to process $n^3$ volume blocks and therefore the time complexity for the third step is

$$t = \frac{8}{7}m^3 \times n^3 = \frac{8}{7}N^3$$  \hspace{1cm} (6.16)

It is also $O(N^3)$. In the fourth step, I gather the symbol distribution information of all the volume blocks and bin the volume blocks into several groups. For each group, I combine the symbol distributions and form a new symbol distribution for the group. The new distributions are fed to a Huffman code generator to build multiple Huffman trees and then generate the code table. For step four, there are four sub-steps. Assuming the original volume data’s symbol width is $b$ bits, the DPCM step will yield error symbols $b + 1$ bits long. For a gray scale image (the case here), $b = 8$. Binning of all the group involves checking the probabilities of 0, ±1, ±2 and this takes $5n^3$ operations. Grouping the symbol distributions takes at most $2^{b+1}n^3$ operations. I use a heap to store the symbols and each heap operation, insertion or deletion, has a $O(\log_2 N_{heap})$ complexity. The heap size $N_{heap}$ is the number of symbols and therefore the worst case cost for each heap operation is $O(\log_2 2^{b+1}) = O(b)$ To build a Huffman tree requires $(2^{b+1} - 1) \times 3$ heap operations. The number of Huffman trees depends on the granularity of the binning algorithm. I can consider it as a relatively small constant $c_t$. Generating the code table is essentially a Huffman tree traversal taking $2^{b+1} \times C$ operations, where $C$ is a small constant varies slightly depending on the actual implementation. The total cost for step four is

$$Complexity_{step4} = 5n^3 + 2^{b+1}n^3 + ((2^{b+1} - 1) \times 3 \times O(b) + 2^{b+1} \times C) \times t_c$$

102
The fifth step compresses the volume blocks and flushes them onto the stable storage (disk). Each symbol requires a table lookup and memory copy. This step’s complexity is \( O(N^3) \). Except for step 4, all the steps have a \( O(N^3) \) complexity and, in this algorithm, \( 2^b < m^3 \). Apparently, step 4 is not dominant. The complexity of this compression algorithm is \( O(N^3) \), which is the same as the other compression algorithms that have been developed.

The decompression algorithm is not symmetrical to the compression algorithm. It only requires Huffman decoding, reverse Hilbert traversal, DPCM decoding and a partial visit of the octree. For each volume block, visiting the octree has a \( O(\log n) \) running time. DPCM and the reverse Hilbert traversal has a \( O(m^3) \) running time. The Huffman decoding takes \( O(H m^3) \) running time where \( H \) is the volume group’s entropy (see equation 6.1). Because \( H \leq b \), the running time to decode a block is \( O(m^3 + \log n) = O(m^3) \). On many occasions, only part of the volume is of interest. In case the entire volume needs to be decompressed, the worst case running time is still \( O(N^3) \). To preview the compressed volume, only the upper level octree index structure needs to be traversed and therefore the preview complexity is \( O(n^3) \).

Since every single step in the compression and decompression algorithm reserves at most twice as much memory as the volume block size, the space complexity of both compression and decompression algorithms is \( O(m^3) \).

### 6.2.7 Results for Lossless Compression

The prototype system was implemented using C++ on Microsoft Windows 2000. It would be trivial to parallelize the algorithm. A job dispatcher is responsible for building the upper level octree index and assigning the volume blocks to the workers. The workers encode the volume blocks using the DPCM encoder and then traverse the blocks following a Hilbert curve. The error symbol distributions are returned to the dispatcher. After building the Huffman coder
Figure 6.3: The parallelized compression algorithm
for each group of blocks, the dispatcher does another round of job dispatching. In the second round, the workers compress the volume blocks using the Huffman encoder. Figure 6.3 shows the detail. This parallelization has relatively low message overhead and therefore is a good candidate for a PC cluster using MPI.

Table 6.1: Comparison of different compression approaches. The compression performance (bits per voxel) is obtained by compressing a rabbit brain volume data set.

<table>
<thead>
<tr>
<th></th>
<th>GZIP</th>
<th>COMPRESS</th>
<th>COMPVOX</th>
<th>SPIHT</th>
<th>PNG image series</th>
<th>My lossless algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits per voxel</td>
<td>3.02</td>
<td>3.20</td>
<td>3.51</td>
<td>3.14</td>
<td>4.45</td>
<td>2.38</td>
</tr>
<tr>
<td>dynamic access</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>parallel compression</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>fast previewing</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6.1 shows the comparison results. The experiment was carried out on a rabbit brain data set. The original volume size is 1 GigaByte. The current implementation can handle a volume with 512 Gigabytes. It is simple to go beyond that if the hardware permits. My approach outperforms all the existing lossless compression methods and also offers other appealing benefits. In addition, as discussed above, the run time of my algorithm is the same order of complexity. (Note: The GZIP and COMPRESS are general purpose compression tools.) Different steps in my approach all contribute to the resulting compression rate. Subdividing the volume can improve Huffman coding’s compression performance. If the coding overhead is ignored, the smaller the block, the better the compression rate. When the block size becomes too small, the coding overhead must be considered. My binning algorithm can effectively reduce such overhead as shown in figure 6.4A. In the implementation, I didn’t use a too small block size because I want to have better I/O performance. Blocks of side length 16 or 32 are reasonable choices. Figure 6.4B demonstrates the compression rate
Figure 6.4: (A) Comparison of Huffman coding overhead: (I) every block has its own Huffman tree. (II) A group of blocks share the same Huffman tree. (III) Compressed data size without considering the Huffman coding overhead. (B) Compression gain from different steps: (i) original volume size (ii) Linear DPCM with Huffman coding (iii) Non-linear DPCM with Huffman coding (iv) Non-Linear DPCM with RLE and Huffman coding (v) Subdividing the volume into multiple blocks and applying non-linear DPCM, RLE, Huffman coding for each block

improvement from using non-linear DPCM coding, RLE and volume subdivision.

6.3 Near-Lossless Compression

There are two definitions of near-lossless compression. One definition is based on the percentage of altered pixels/voxels. For example, less than 5% of the pixels/voxels altered in the decompressed image is considered near-lossless. This definition is not convincing, because if the 5% altered pixels/voxels are the edges in the image and the intensity changes are significant, then the image can look totally different. A more reasonable and widely accepted definition concerns the visual appearance of the decompressed image. A compression method is near-lossless if no pixel/voxel is changed in magnitude by more than d gray levels. My near-lossless method is based on the second definition.
6.3.1 Outline of the Near-Lossless Compression Method

The compression method is outlined as follows. The 3D image is divided into sub-volume blocks and then indexed by an octree data structure. Each block is traversed following a Hilbert curve. While traversing the block, a near-lossless or lossless DPCM prediction is performed. The prediction error is recorded, as is the histogram of the prediction error. Each sub-volume block has its unique prediction error distribution. To form a Huffman code book of the prediction error for the entire volume is inefficient. On the other hand, to produce a Huffman code book for every sub-volume block also introduces heavy coding overhead. I characterize each block’s error distribution as a point in a high-dimensional space and then bin the points using a novel binning method. All the error distributions that fall into the same bin are summed together to form a summed error distribution. I build a Huffman code book for this distribution. The total number of Huffman code books is the number of bins. The coding overhead is therefore effectively reduced. All the sub-volume blocks’ prediction error is coded according to its own Huffman code book. Finally, the code books, the upper level index and encoded sub-volume blocks are flushed to disk. Figure 6.5 is the data stream flow chart of the compression method.

When the 3D image is accessed by digital brain atlas applications, based on the index structure, only those sub-volume blocks that are actually needed will be decoded. To decode
a block, its own Huffman code book is loaded. The decompressed data stream is the prediction error, which is translated back to the voxel intensity. The last step is to translate the Hilbert scan order back to normal raster scan order. It is faster to decompress a sub-volume block than to compress it.

### 6.3.2 Image Context Change

As discussed in section 4.3, a Hilbert curve is an open space filling curve (shown in figure 6.6A,B). It can be used in $n$-dimensional space where $n \geq 2$. The common image scan order is the raster scan. In compressing 2D images, the raster scan order is only slightly less efficient than the Hilbert scan order; therefore, the literature pays little attention to this technique. However, in 3D images, Hilbert traversal offers an appealing property. In a Hilbert curve, every pixel/voxel is the immediate neighbor of its subsequent pixel/voxel. There isn’t any abrupt image context change along a Hilbert traversal. In addition, the speed of context change is also much slower than that of raster scan. Let’s define an image context as a group of pixel/voxels that occupy a small area. In a 3D image, suppose there is a 3D image context of $N^3$ voxels. A raster scan will leave the context after visiting $N$ voxels. For a Hilbert scan, in the best case, it leaves the context after visiting all the $N^3$ voxels.

![Figure 6.6](image)

Figure 6.6: A) 3D Hilbert curve (Level 1); B) 3D Hilbert curve (Level 2); C) Voxels’ relative locations along a Hilbert curve; D) Voxels’ relative locations along a raster scan.

A DPCM prediction is used to reduce the data source’s entropy by predicting the current pixel/voxel’s intensity based on the previously visited pixel/voxels. The recorded prediction error can be used to reconstruct the data source and the prediction error’s entropy is usually
much smaller than the data source’s entropy. The greater the occurrence of big prediction errors, the bigger the prediction error’s entropy. Usually, a big prediction error occurs when the scan line passes through a high frequency image component such as an edge. Assume the same 3D image context defined in the previous paragraph is surrounded by a 3D image edge. Smooth regions are on both sides of the 3D edge. Using raster scan, the scan line passes through the 3D edge $2N^2$ times, yet in the best case, the Hilbert scan line pass through the 3D edge only twice. Depending on the location of the image context, the Hilbert scan line may pass through the context more than twice; but usually much less than $2N^2$ times.

Since the Hilbert scan line intersects the image edges less frequently and the image context change is also much slower than for the raster scan, a less complicated and faster DPCM can be used. Along the Hilbert scan line, suppose the current voxel is $V_n$ and prediction error $E$ is:

$$E = V_n - (V_{n-1} + V_{n-2})/2 \quad (6.18)$$

Figure 6.6C,D shows the voxels’ relative locations in Hilbert scan order and raster scan order. In both, the distance between $V_{n-1}$ and $V_n$ is 1. Along the Hilbert curve, the distance between $V_{n-2}$ and $V_n$ is $\sqrt{2}$. In raster order, the distance between $V_{n-2}$ and $V_n$ is 2. If there is a context jump, the prediction is invalid. Because $V_{n-2}$ is closer to $V_n$ in Hilbert scan order than in raster scan, the DPCM in Hilbert scan order tends to predict the intensity better than the same method in raster scan. This is observed through experiments.

### 6.3.3 Near-Lossless Quantization

Suppose $d$ levels of intensity change is allowed in the near-lossless quantization. Let $m$ be an integer. The quantization function is:

$$Q(E) = \begin{cases} 
(2d + 1)m & \text{if } E = (2d + 1)m + i \text{ for } i = 1, 2, \ldots, d \\
(2d + 1)(m + 1) & \text{otherwise} 
\end{cases} \quad (6.19)$$
Two examples are illustrated in figure 6.7A. Inter-dependency of the prediction error requires development of a complicated prediction quantization algorithm [98]. This problem can be avoided by looking at both the predicted voxel intensity and the decoded voxel intensity. I decode the visited voxel’s intensity while I encode the next voxel; therefore, I can guarantee that the quantization error will not accumulate. Let’s denote $V_n$ as the current voxel’s intensity, $V'_n$ as the decoded voxel’s intensity, $E$ as the prediction error, and $E'$ as the quantized prediction error. Assuming $V_{-2} = V'_{-2} = V_{-1} = V'_{-1} = 0$.

$$
E_0 = V_0 - (V'_{-2} + V'_{-1})/2
$$
$$
E'_0 = Q(E_0)
$$
$$
V'_0 = E'_0 + (V'_{-2} + V'_{-1})/2
$$

for the $n$th voxel, I have

$$
E_n = V_n - (V'_{n-2} + V'_{n-1})/2
$$
$$
E'_n = Q(E_n)
$$
$$
V'_n = E'_n + (V'_{n-2} + V'_{n-1})/2
$$

(6.20)

The prediction error has a Laplace distribution [109] (see equation 6.10). For prediction error $E$, the probability of occurrence is

$$
P(E) = \int_{E-0.5}^{E+0.5} \frac{1}{2b} e^{-|x|/b} dx = \begin{cases} 
\frac{0.5 - E}{b} - \frac{e^{-0.5-E}}{b} & \text{if } E > 0 \\
1 - e^{-\frac{1}{2b}} & \text{if } E = 0 \\
\frac{e^{0.5+E}}{b} - \frac{e^{-0.5-E}}{b} & \text{otherwise}
\end{cases}
$$

(6.22)

Using equation 6.1, the entropy $H$ of the prediction error can be computed. When the prediction error is quantized using equation 6.19, the probability of quantized prediction
error $E_q$ is

$$P(E_q) = \sum_{E=E_{q-d}}^{E_{q+d}} P(E)$$  \hspace{1cm} (6.23)

The entropy $H_q$ of the quantized prediction error is

$$H_q = \sum_{E_q} -P(E_q)\log_2 P(E_q)$$  \hspace{1cm} (6.24)

Therefore, my near-lossless quantization method can improve the compression rate by $r\%$ where

$$r = 100 - 100H_q/H$$  \hspace{1cm} (6.25)

Figure 6.7B,C are the plots that demonstrate the relationship between the maximum quantization error $d$, quantization error distribution’s $b$ value, the distribution’s entropy, and the percentage of compression improvement. Increasing the maximum quantization error $d$ can improve the compression rate; however, it degrades the decompressed image’s quality. On the other hand, increasing $d$ does not increase the compression rate linearly. Accurate intensity prediction yields a small $b$ value in the prediction error distribution, near-lossless quantization performs better when $b$ is small. Hilbert traversal helps to reduce $b$.

### 6.3.4 Results for Near-Lossless Compression

Table 6.2 shows the comparison between my near-lossless compression method versus other approaches.

### 6.4 Progressive Lossy-to-Lossless Compression

In section 6.2 and section 6.3, I discussed a lossless coder and a near-lossless coder. These two coders have limited usages. In this section, I am going to propose a progressive lossy-to-lossless coder. The progressive coder can be used as either a lossy image coder or a lossless
Figure 6.7: A) Near-lossless quantization schemes when \(d=1\) and \(d=2\). B) The compression rate improvement for different Laplace distributions (with different \(b\) values) where \(d\) is the maximum allowed quantization error. C) The entropy decrease when different quantization intervals are used. The quantization interval equals \(2d+1\).

coder because the lossy and lossless image information is embedded in the same data stream. Strictly speaking, the near-lossless coder is also a lossy coder; therefore, the progressive coder has a wide range of applicabilities.

### 6.4.1 Overview of the Coding Method

The major components of the coding method are shown in figure 6.8. The raw image is divided into cubic blocks. This is called tiling and it is widely accepted as a strategy to offer good scalability, error resistance and parallelism. The size of the blocks is a parameter depending on the particular data sets. Each block is traversed by a Hilbert curve so that the
Table 6.2: Comparison of different compression approaches. The compression performance (bits per voxel) is obtained by compressing a rabbit brain volume data set.

<table>
<thead>
<tr>
<th></th>
<th>GZIP</th>
<th>COMPRESS</th>
<th>COMP-VOX</th>
<th>SPIHT</th>
<th>PNG image series</th>
<th>My lossless algorithm</th>
<th>My near-lossless algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits per voxel</td>
<td>3.02</td>
<td>3.20</td>
<td>3.51</td>
<td>3.14</td>
<td>4.45</td>
<td>2.38</td>
<td>2.18</td>
</tr>
<tr>
<td>dynamic access</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>parallel compression</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fast previewing</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

2D or 3D signal is transformed into a 1D signal. This step generalizes this data compression method so it can be used for both 2D and 3D images. Hilbert traversal is not just for generalization purposes, rather it has a nice feature. Namely, its slow context change and locality property. Unlike the raster scan, which is adopted by most imaging systems, there is no context jump in the Hilbert traversal. Since neighbor pixel/voxels usually have similar or identical intensities, a 1D signal generated from Hilbert traversal of a multi-dimensional image preserves the spatial correlations in the original image. An integer wavelet transformation (IWT) is applied to the 1D signal to generate a stream of wavelet coefficients. I regard this data stream as a linearization of a wavelet coefficients tree based on breadth-first traversal. A linearization algorithm based on a spanning tree and priority queue rearranges the wavelet coefficients tree in a different order so that the result is suitable for progressive encoding/decoding and the overhead is small. The details are in section 6.4.4. Since I do not separate the bit-plane as in SPHIT [99] or zerotree [110], the linearization result can still be further condensed by an entropy coder such as a Huffman coder. The decoding process reverses the encoding process.
6.4.2 Wavelet Coefficients Tree

More details about IWT are discussed in section 4.4. After the wavelet transformation is applied to the 1D signal, the wavelet coefficients are stored in a data stream $S_w$ so that the lower frequency components appear before higher frequency components. Figure 6.9A is an illustration of $S_w$. Since $S_w$ is hierarchically produced, it can be regarded as a linearized wavelet coefficients tree $T$ (shown in figure 6.9B). $S_w$ is therefore a breadth-first traversal of $T$. 

Figure 6.9: Wavelet coefficients data stream can be considered as a linearized wavelet coefficients tree. A) Wavelet coefficients data stream. B) Wavelet coefficients tree.
In the MSE\(^3\) sense, wavelet coefficients with higher magnitudes carry more information than the coefficients with lower magnitude [99]. The linearization using breadth-first traversal does not assign higher priority to the more important information. In order to progressively compress a signal, the wavelet coefficients must be re-ordered so that the more significant coefficients gather in the front of the data stream and the less significant ones are moved towards the tail. The data compression problem is therefore transformed into a tree traversal problem.

6.4.3 Important Properties

To sort all the wavelet coefficients is not a good option. When the wavelet coefficients are sorted, their new order is different than their original order in \(S_w\). To reproduce \(S_w\), the magnitude of the wavelet coefficients as well as their locations in \(T\) must be preserved. Excessive overhead makes this sorting approach unacceptable.

To reduce the coding overhead, I propose a new linearization algorithm. The new algorithm is based on a few important facts and observations of \(S_w\).

1. Most of the images’ information is concentrated in the low frequency components.
2. Magnitude self-similarity exists between sub-bands. In wavelet coefficients tree \(T\), the parents and children of a node with high magnitude may also have high magnitudes.
3. Most of the wavelet coefficients are small.
4. Many tree nodes containing 0 and 1 are clustered together in \(T\).

Since most of the image’s information (or energy) is concentrated in the low frequency components, no matter the wavelet coefficients’ magnitude, the first few levels of \(T\) should be flushed to the front of the data stream. \(T\) grows exponentially; therefore, the first few levels of \(T\) form much less than 1% of the data stream. Magnitude self-similarity spans

---

\(^3\)mean square error
the branches in $T$. To encode a branch $B$ in $T$, only the location of $B$’s root needs to be encoded, the rest of $B$ can be simply breadth-first traversed. Branch traversal eliminates many location codes for the nodes in $T$. Again, the lower frequency components in $B$ should appear earlier than the higher frequency components in $B$, which also requires breadth-first traversal. Most of the wavelet coefficients are small. A magnitude of 0 can be safely discarded without encoding. Other small coefficients can be quantized in the lossy coder. In a lossless coder, all the information must be preserved. Since the tree nodes containing low magnitude coefficients are often clustered in $T$, and the clusters also appear in the form of tree branches, these coefficients can be encoded in the same way as the high magnitude coefficients. Namely, encode the root of the branch and traverse the branch afterwards. The wavelet coefficients tree breaks down into a spanning tree forest. Each branch is a spanning tree and the spanning tree nodes satisfy a common criterion. Details are in the next section.

### 6.4.4 Traverse the Wavelet Coefficients Tree

![Wavelet coefficients tree](image)

Figure 6.10: Wavelet coefficients tree with the important information highlighted.

The progressive coding problem is transformed into a tree traversal problem. This process is also called tree linearization. To produce a compact data stream, the goal is to eliminate unnecessary location codes as much as possible by following a hidden logic. The linearization algorithm has the following steps. In the first step, $2^n$ wavelet coefficients at the front of $S_w$ are copied from $S_w$ to the output data stream $S_o$. This step essentially breadth-
first traverses the first \( n \) levels of \( T \). Only a small percentage, \( x\% \) of \( T \) is visited, where \( 0 < x < 100 \) and \( x \) is usually smaller than 10. Assuming \( S_w \) contains \( 2^N \) coefficients, \( n \) satisfies \( n = N - 2 + \lfloor \log_2 x - 2 \log_2 5 \rfloor \). Step two reorders the remaining \( 2^N - 2^n \) wavelet coefficients. All the non-zero coefficients are inserted into a heap (priority queue) sequentially. The elements in the heap are the tuples \( t = (c, l) \), where \( c \) denotes the wavelet coefficient and \( l \) denotes \( c \)'s location in \( S_w \). The tuples are ranked by the coefficients. Due to the nice property of a heap, the biggest element is always at the top of the heap. Observation 2 in section 6.4.3 tells us that a large coefficient is likely to be in a tree branch in which all the branch nodes have high coefficient values. The heap offers an indirect way to locate such tree branches. Figure 6.10 is an illustration of the important versus less important information in a wavelet coefficients tree. In figure 6.10, the first few levels of the tree, shown surrounded by a rectangle, contain the most important, low frequency wavelet coefficients. Tree nodes indicated by solid circles contain high magnitude coefficients. The branches of high magnitude coefficients need to be output before the branches containing low magnitude coefficients. Denote \( c_{\text{max}} \) as the largest coefficient contained in a branch. For all non-leaf nodes \( C_i \) in the branch, the following condition must be satisfied to define a branch:

\[
|c_i| > 2^{|\log_2 |c_{\text{max}}| - r|}
\]  

(6.26)

where \( r \) is an integer parameter such that \( 0 < r < 8 \). \( r \) defines the magnitude range of the coefficients that can be grouped in the same branch. If \( r \) is too small, the branch contains only a few nodes; therefore, the coding overhead is high. If \( r \) is too big, there is no discrimination between the coefficients, so that the progressive coder can be impaired. I choose \( r = 2 \) in my implementation. For leaf nodes in the branch, this condition is lifted so that I can define the boundary of the branch without using extra bits. Since the leaf node does not satisfy equation 6.26, whenever such a node is encountered, the branch traversal
function will not go further down that branch. Every branch usually contains more than one node. Except for the largest node in the branch, all the other branch nodes are still recorded in the heap. Therefore, it is necessary to check each node taken from the heap to see if this node has already been processed. If not, a new branch is initialized, otherwise, the node is discarded and another node is fetched from the heap. In order to check if a node has already been processed, a decoding array must be maintained during the encoding process. Every encoded coefficient is recorded in the decoding array as soon as it is encoded. The pseudo code for the linearization algorithm is as follows.

procedure linearize($S_w, x, N, r$)

Require: $x < 1.0$ and $\text{sizeof}(S_w) = 2^N$ and $0 < r < 8$

1: initialize decoding array $S_d$
2: initialize queue $Q_o$
3: $n \leftarrow N - 2 + \lfloor \log_2 x - 2\log_2 5 \rfloor$
4: for $i = 0$ to $2^n$ do
5: append $S_w[i]$ to $Q_o$ {copy the low frequency components}
6: $S_d[i] \leftarrow S_w[i]$
7: end for
8: initialize a heap $H$
9: for $i = 2^n + 1$ to $2^N$ do
10: if $S_w[i] \neq 0$ then
11: insert tuple $t(c = S_w[i], i)$ into $H$
12: end if
13: end for
14: while $H$ is not empty do
15: $t_{temp} \leftarrow$ top element of $H$

A branch leaf node is not equivalent to the leaf node of a wavelet coefficients tree. A branch leaf node can be an internal node of a wavelet coefficients tree.

118
16: remove the top element of $H$
17: if $t_{\text{temp}}$ is in $S_d$ then
18:    continue
19: else
20:    $\min \leftarrow 2^{\left\lfloor \log_2 |t_{\text{temp}.c}| - r \right\rfloor}$
21:    $i \leftarrow \text{findBranchRoot}(t_{\text{temp}}, S_w, S_d, \min)$
22:    if $i \neq t_{\text{temp}.i}$ then
23:      append $-i$ to $Q_o$
24:      append $\min$ to $Q_o$
25:    else
26:      append $i$ to $Q_o$
27:      $\text{traverseBranch}(i, S_w, S_d, Q_o, \min)$
28:    end if
29: end if
30: end while
31: build a histogram $H_{\text{is}}$ of the integers appearing in $Q_o$
32: build a Huffman tree $T_h$ based on $H_{\text{is}}$
33: encode $Q_o$ using $T_h$ and produce the output data stream $S_o$

procedure $\text{findBranchRoot}(t, S_w, S_d, \min)$
1: $i \leftarrow t.i$
2: $j \leftarrow \lfloor i \div 2 \rfloor$
3: while $S_w[j]$ is not in $S_d$ and $S_w[j] \geq \min$ do
4:    $i \leftarrow j$
5:    $j \leftarrow \lfloor i \div 2 \rfloor$
6: end while
7: return $i$

procedure $\text{traverseBranch}(i, S_w, S_d, Q_o, \min)$
1: initialize queue $Q$
2: append tuple $t(c = S_w[i], i)$ to $Q$
3: while $Q$ is not empty do
4:   $t_{temp} \leftarrow$ the first element in $Q$
5:   remove the first element from $Q$
6:   append $t_{temp}.c$ to $Q_o$
7:   $S_d[t_{temp}.i] \leftarrow t_{temp}.c$
8:   if $t_{temp}.c \geq \text{min}$ then
9:       $j \leftarrow t_{temp}.i \times 2$
10:      if $S_w[j]$ is not in $S_d$ then
11:         append tuple $t(c = S_w[j], j)$ to $Q$
12:      end if
13:      $j \leftarrow j + 1$
14:      if $S_w[j]$ is not in $S_d$ then
15:         append tuple $t(c = S_w[j], j)$ to $Q$
16:      end if
17: end while

The running time of my linearization algorithm is dominated by the heap operations, and therefore the worst case running time is $O(m \log m)$, where $m = 2^N$. Besides the space allocated to store the original data stream $S_w$, I also allocated two extra buffers with similar size to store the decoding stream and the heap. Therefore, the space complexity is $O(Cm)$ with a constant factor $C = 3$. 
6.4.5 Comments on the Lossy Coder

In the previous section, the linearization algorithm and the progressive lossy-to-lossless coder is described. Similar to most lossless coders, a progressive lossy-to-lossless coder produces a fairly big output file. If a small output file is desired, a lossy-only coder is needed. The traditional way to produce a lossy coder is a spatial-to-frequency domain transformation \(^5\), followed by a uniform quantization of the transformation coefficients. Uniform quantization does not produce an optimized data stream for my linearization algorithm and therefore does not offer the best compression rate at the same peak signal-to-noise ratio (PSNR). Instead of the uniform quantization, I developed a spanning-tree-based quantization method. In my linearization algorithm, for every coefficient, the maximum coding overhead occurs when this coefficient is isolated from other non-zero coefficients by wavelet coefficients equal to zero. To retain the maximum amount of image information at the same bit rate, such overhead must be reduced.

The quantization algorithm takes two parameters, a spanning tree size \(j\)\(^6\) and a coefficient threshold \(c_t\). During the linearization, if a branch only has \(i\) nodes \((i < j)\) and the maximum coefficient \(c_{\text{max}} < c_t\), the entire branch is quantized to 0. If the quantization result still does not meet the bit rate defined by the imaging application, the output data stream can be chopped to the exact bit rate.

6.4.6 Bit-Error-Resilient Coding

In the decoder, extra error checking code can detect some bit errors. The Huffman coder uses variable length unique codes to encode different integers produced by the linearization algorithm. A bit error can be detected when a non-codeword occurs. An unexpected coefficient also indicates a bit error. If the decoder decides that two coefficients reside at the same location in the decoding stream based on the branch traversal order, there is an unexpected

---

\(^5\)DCT, DHT or Wavelet transformation are usually used.
\(^6\)In terms of the number of nodes in the spanning tree.
coefficient. If a high magnitude coefficient appears in a branch of low magnitude coefficients, this coefficient is unexpected. If a coefficient’s magnitude is out of the possible range, there must be an error. An invalid location code is also an indication of a bit error. Not all bit errors can be detected, but my linearization algorithm provides some hints to the decoder that rigorous error checking code can take advantage of.

To recover from a bit error, the simplest way is retransmission. A lazy approach is to retransmit only the current branch. If such retransmission causes more bit errors, the retransmission must be restarted from the beginning. In some extreme cases, when retransmission is impossible, some coefficients can be guessed based on their neighbors’ magnitude; however, this approach has very limited recovery power.

6.4.7 Results for Progressive Lossy-to-Lossless Compression

![Figure 6.11: JPEG2000 at 0.08 bpp, 0.32 bpp and 1.28 bpp.](image)

Table 6.3: Quantitative comparison (PSNR at the same bit rate)

<table>
<thead>
<tr>
<th>Format</th>
<th>0.08</th>
<th>0.32</th>
<th>1.28</th>
<th>5.12</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG 2000</td>
<td>78.42</td>
<td>80.65</td>
<td>86.35</td>
<td>90.89</td>
</tr>
<tr>
<td>My approach</td>
<td>79.13</td>
<td>84.06</td>
<td>92.27</td>
<td>98.12</td>
</tr>
</tbody>
</table>

My experiment compares the visual effect of the standard JPEG2000 [111] with my progressive coder at the same bit rate using the standard 512 × 512 Lena image. The
Figure 6.12: My new method at 0.08 bpp, 0.32 bpp and 1.28 bpp.

experiment shows slightly better visual appearance in figure 6.11, 6.12 and PSNR at the same bit rate in table 6.3.
Chapter 7
Conclusion

In this dissertation, I described a unique multi-disciplinary medical imaging project. A framework for building a state-of-art brain digital atlas is established. Several novel approaches are developed to address the issues in various parts of this project. While solving theoretically difficult problems, applications are always kept in my mind to make this project applicable. Many self-contained topics are discussed in this dissertation; however, they are not loosely coupled. Rather, those topics are indispensable components of digital brain atlas construction and application.

The most important problem of constructing the brain digital atlas is the image sequence alignment problem. Chapter 2 presented a global alignment metric, MEBP, which is the foundation of image sequence alignment (ISA). MEBP considers multiple image slices and uses a few previously aligned images to align the next float image. An imaginary ray passes through the image stack and predicts the value of the next pixel’s intensity. The prediction errors’ entropy is minimized when alignment occurs.

Chapter 3 addressed another fundamental problem of ISA, namely the optimization method for finding the best alignment. Finding the global optimum among many local optimums requires a global optimization method, yet the heavy computation cost prohibits the straight application of global optimization methods such as the genetic algorithm or simulated annealing. I used a multi-resolution approach to solve this problem. The upper layer of the optimization pyramid employs a genetic algorithm (GA). A cluster-assisted approach
is used to ensure the convergence of the GA. The bottom layers of the optimization pyramid are explored using the simplex method.

Chapter 4 studied an image post-processing problem. Swiss-cheese-type image defects (SCDs) are common in cryosectioned histology image sequences. To patch the SCDs, neighborhood resemblance and long range correlation are used. Hilbert traversal serves two purposes in the restoration method. If the defective pixels/voxels only occupy several consecutive pixels/voxels in the Hilbert traversal, these elements are interpolated based on the immediate neighborhood. If the defect spans a big area/volume, its surrounding good pixels/voxels are used to extract a multi-resolution signature of the defect and then the signature is searched in the image for a match. The content surrounded by the match signature is used to patch the SCD.

Chapter 5 presented the scalable hyper-space file, SHSF, data structure. This data structure is used to encode the high resolution digital brain atlas. It is an octree variant and differs from the traditional octree in its special linear code and multiple levels of indirection. The SHSF data structure is further explored in chapter 6 as an upper level image structure. It facilitates many digital atlas applications.

In chapter 6, three volumetric data compression methods were presented. A lossless compression method is based on non-linear DPCM. A near-lossless method sacrifices some data fidelity to improve the speed. The third method is a generalized coder that can be used to compress 2D as well as 3D images and incorporates lossy-to-lossless image information in the same data stream. This progressive image coder takes advantage of Hilbert traversal, integer wavelet transformation (IWT) and a new tree linearization method. Among the three methods discussed in chapter 6, the progressive coder is most versatile.

In the future, there are still a number of problems to be solved. Deformable image sequence alignment is an extremely challenging problem. Unlike the pairwise deformable image registration, the deformable image sequence alignment adds more degrees of freedom to the system. Consequently, it adds much more computation complexity. New alignment
objective functions and more efficient optimization algorithms need to be developed. One possible solution is to use a feature extraction algorithm followed by an N-body problem. The deformable alignment results in a grid and the grid is the basis for the image warping step. Deformable image sequence alignment shares some common aspects with deformable image registration. Yet the pairwise counter-problem tries to transform the float image to the same as the reference image. This will result in a cylinder if it is used in image sequence alignment.

The 2D-to-3D brain mapping problem remains an open issue. One expensive optimization problem embedded in another optimization problem is the nature of 2D-to-3D brain mapping. Using the existing cost function, the global optimum is often buried in a plateau in the search space. The global optimum may not even be the right answer. A more effective cost function is therefore necessary. The embedded optimization problem also needs a more efficient heuristic algorithm. The deformable 2D-to-3D brain mapping is the next level of this problem.

For the image defect concealment algorithm, I plan to work on a texture synthesis method based on global optimization. As an extension to my data compression research, I will apply my tree linearization method in region-of-interest, ROI, coding. With ROI coding, bits are assigned to the most important feature of the image. ROI coding can be useful for many potential digital atlas applications.

To build the state-of-art digital brain atlas and use it in bio-medical research, there is still a long way to go. The initial solutions may suffice for a period of time, yet, the needs are ever growing and expanding. I envision many potential new developments in this field. For old problems, research is still necessary to quantify the current qualitative results. For example, how do the interpolation methods interfere with image registration? Image registration constantly transforms the original image into different orientations and therefore image interpolation is used extensively. A fast yet inaccurate interpolation method can have negative influence on the optimization routine in image registration. An accurate
yet slower interpolation method may reduce the speed in finding the best registration. This problem has been ignored by many researchers, yet, problems of similar nature need to be studied so that in-depth understanding of the problems can be acquired.

To conclude this dissertation, I will investigate new research problems related to digital brain atlas construction and also study some interesting old problems to get more complete and in-depth understanding of them.
Appendix A

Brief Introduction to Mathematical Morphology

Mathematical morphology developed in set theory has received great attention in image processing. The morphological operators\(^1\) are particularly useful for the analysis of binary images. Common usages of those operators include edge detection, noise removal, image enhancement and image segmentation [46]. As demonstrated in figure A.1\(^2\), 2D morphological operators can be used to flag small image defects\(^3\) and remove small debris\(^4\). The two most basic operations in mathematical morphology are erosion and dilation. Dilation is also known as grow and expand; the effect of dilation is as if a binary image is growing itself. Erosion is also known as shrink and reduce; it reduces the size of a binary image. Dilation and erosion are complementary. A dilation of the foreground pixel is the erosion of the background pixel and vice versa. Both of these operators take two pieces of data as input: an image to be eroded or dilated, and a structuring element\(^5\). The structuring element is just a set of point coordinates (although it is often represented as a binary image).

It differs from the input image coordinate set in that it is normally much smaller, and its

---

\(^1\)Also known as morphological filters.

\(^2\)Grayscale on the pictures is only for demonstration purpose, the images are B/W. Except the light gray (representing the vanished white pixels) in D and G, all other gray or black pixels have a black color.

\(^3\)Image H and J in figure A.1 are the flagged image defects. Image H and J are the result of subtract the original image A from the morphological filtered image E and I respectively.

\(^4\)After erosion, the original image A becomes image D. Small debris and rough boundary is eroded away. Dilating image D yields image G. The smoothed boundary grows back, but the debris is permanently removed.

\(^5\)Also known as a kernel.
coordinate origin is often at the center of the kernel, so that some coordinate elements will have negative values. Erosion and dilation work by translating the structuring element to various points in the input image, and examining the intersection between the translated kernel coordinates and the input image coordinates. If the following condition is satisfied, the translated coordinate remains in the output set. Otherwise, that coordinate is removed from the input set

\[ k = T^{-1}(T(k) \cap I) \] (A.1)

where \( k \) is the kernel, \( I \) is the input set (input image) and \( T \) is the translation. Virtually all other mathematical morphology operators can be defined in terms of combinations of erosion and dilation along with set operators such as intersection and union.

2D morphological operators can be easily extended to 3D. The 2D kernel \( k_2 \) and 3D kernel \( k_3 \) that I use are

\[
\begin{align*}
k_2 &= \{-1, -1\}, \{0, -1\}, \{1, -1\}, \{-1, 0\}, \{0, 0\}, \{1, 0\}, \{-1, 1\}, \{0, 1\}, \{1, 1\} \\
k_3 &= \{-1, -1, -1\}, \{0, -1, -1\}, \{1, -1, -1\}, \{-1, 0, -1\}, \{0, 0, -1\}, \{1, 0, -1\}, \{-1, 1, -1\}, \{0, 1, -1\}, \{1, 1, -1\}, \{-1, 0, 0\}, \{0, 0, 0\}, \{1, 0, 0\}, \{-1, 1, 0\}, \{0, 1, 0\}, \{1, 1, 0\}, \{-1, -1, 0\}, \{0, -1, 0\}, \{1, -1, 0\}, \{-1, 0, -1\}, \{0, 0, -1\}, \{1, 0, -1\}, \{-1, 1, -1\}, \{0, 1, -1\}, \{1, 1, -1\}, \{-1, 0, 1\}, \{0, 0, 1\}, \{1, 0, 1\}, \{-1, 1, 1\}, \{0, 1, 1\}, \{1, 1, 1\} \end{align*}
\] (A.3)

\[ ^6 \text{Note that in many implementations of morphological operators, the structuring element is assumed to be a particular shape (e.g. a } 3 \times 3 \text{ square) and so is hardwired into the algorithm.} \]

\[ ^7 \text{In mathematical morphology, an image is considered as a set of point coordinates.} \]
Figure A.1: A. original binary image; B. one round of dilation; C. two rounds of dilation; D. one round of erosion; E. one round of erosion after one round of dilation; F. one round of erosion after two rounds of dilation; G. one round of dilation after one round of erosion; H. E subtract A; I. two rounds of erosion after two rounds of dilation; J. I subtract A
Appendix B

Image Pre-Processing

The raw images are obtained by photographing the cryosections. Many problems prevent the raw images from being directly used for image registration. Table B.1 shows some of the most common problems. I use thresholding in conjunction with Gaussian filtering to reduce the background noise. The background is set to pure black, corresponding to 0 in the 256-level grayscale. Small pieces of debris scattered in the background are removed in three steps; namely, thresholding, Gaussian filtering and morphological filtering\(^1\). After the thresholding and Gaussian filtering, some of the debris still exists, I binarize the image to create a binary image mask and then I erode the image mask so that the small debris is removed. In order not to reduce the area of the image, the mask is dilated. Multiplying the mask with the image, I can obtain an image that is free of small debris. For some relatively large pieces of debris, manual adjustment is my current solution.

Another unavoidable image defect is air bubbles. The bubbles are introduced to the image while mounting the slices onto the glass slides. Due to the bubble’s circular shape, it can be automatically located using an analytical method. The idea is to transform the image into Hough space [112] and search for clusters. A circle can be parameterized by

\[
(x - a)^2 + (y - b)^2 = r^2
\]  

\(^1\)More detail can be found in appendix A.
To search for circles, the Hough space is a 3D parameter space of \(a, b\) and \(r\). Given a predefined granularity of the Hough space, for each point in the image, find all the \((a, b, r)\) tuples that satisfy equation B.1. A point is therefore translated into a curve in the Hough space. After translating all the points into the Hough space, many clusters are formed. The center of a cluster is the location of a bubble. To reduce the amount of computation, the image needs to be preprocessed. Since the bubbles are highlights in the image, the image can be thresholded and then binarized to change the bubbles and other highlights into white pixels. All the black pixels can be safely ignored. A multi-resolution Hough transformation can then be employed to reduce the computation. A false positive result may exist when there is a group of highlighted pixels. The false positive can be verified by examining the extent of the highlight and then comparing with the radius.

In the skull, the brain is a connected entity. When the brain is sectioned, some of the brain tissues are no longer connected with each other and therefore many disconnected components are generated. I call them component images. Those component images must be registered separately. To extract the component images, I use a binary image mask in conjunction with component labelling. Only one scan is required to identify the disconnected component images. While scanning through the pixels, if the current pixel is a foreground pixel, this pixel’s previously visited neighbors will be examined to see if they are also foreground pixels. If any of the neighbors is a foreground pixel, the foreground neighbor pixel’s label will be inherited by the current pixel. If none of the neighbors is foreground pixel, the current pixel will be assigned a new label. After scanning through the image, the number of different labels is the number of disconnected component images.

Even when the images are taken in constant conditions, due to errors in the cryostat and tissue melting, uneven brightness of the images is a common problem. A histogram equalization will be necessary in the image postprocessing. Missing tissue restoration is another image postprocessing problem and is discussed in chapter 4. Overlapping tissue and broken tissue are complex problems and are out of the scope of this research; therefore
those image defects are manually removed. The problem of an image’s being out-of-focus is not addressed either. Image preprocessing helps to reduce the image problems. Not all problems can be solved automatically; human interaction or even manual adjustment may be necessary for some cases. Automatic preprocessing can reduce such work. A clean data set is helpful for further analysis, application and research.
Figure B.1: Raw image defects (A) Gaussian noise (B) Debris (C) Air bubble (D) Overlapping tissue. (E) Out of focus (F) Broken tissue
Table B.1: Summary of the raw image problems

<table>
<thead>
<tr>
<th>Image Problem</th>
<th>Cause</th>
<th>Solution</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noise</td>
<td>Imaging system noise</td>
<td>Thresholding, Gaussian filtering</td>
<td>Figure B.1A</td>
</tr>
<tr>
<td>Debris</td>
<td>Dirt in the imaging system or pieces of tissue debris</td>
<td>Thresholding, Gaussian filtering Morphological filtering or manual removing</td>
<td>Figure B.1B</td>
</tr>
<tr>
<td>Air bubble</td>
<td>Air is trapped while mounting slices onto the glass slides</td>
<td>Thresholding, Morphological filtering, Hough transformation and image defect concealment</td>
<td>Figure B.1C</td>
</tr>
<tr>
<td>Overlap tissue</td>
<td>Slice mounting problem</td>
<td>Manual adjustment</td>
<td>Figure B.1D</td>
</tr>
<tr>
<td>Out of focus</td>
<td>Human error</td>
<td>image restoration (not addressed here)</td>
<td>Figure B.1E</td>
</tr>
<tr>
<td>Broken tissue</td>
<td>Cryosection and slice mounting problem</td>
<td>Manual adjustment</td>
<td>Figure B.1F</td>
</tr>
<tr>
<td>Uneven lighting or brightness</td>
<td>Variation of slice thickness or tissue melting</td>
<td>Histogram alignment</td>
<td>Figure B.2A vs. Figure B.2B</td>
</tr>
<tr>
<td>Missing tissue</td>
<td>Cryosection and slice mounting problem</td>
<td>Image post-processing and 3D image defect concealment</td>
<td>Figure B.2C vs. Figure B.2D</td>
</tr>
<tr>
<td>Disconnected tissue</td>
<td>Naturally disconnected</td>
<td>Binarization and component image labelling</td>
<td>Figure B.2E</td>
</tr>
<tr>
<td>Image misalignment</td>
<td>Cryosection and slice mounting problem</td>
<td>Image registration (main focus of this dissertation)</td>
<td>Figure B.2E vs. Figure B.2F</td>
</tr>
</tbody>
</table>
Figure B.2: Raw image defects (A,B) Uneven lighting between neighboring slices (C,D) Missing tissue in one of the neighboring slices (E,F) Image misalignment
Index

2ⁿ-ary, 41
8-connectivity, 54

arithmetic coding, 86

binarization, 81
binning, 3
bit-error, 51
    resilient coding, 121
breadth-first traversal, 114

cluster, 105
    bounding box, 40
clustering
    algorithm, 38
    behavior, 40
    density-based, 45
    grid-based, 45
component
    image, 6, 132
    labeling, 54, 132
compression
    dictionary-based, 86
    heuristic method, 86
    lossless, 87
    lossy, 88
    near-lossless, 87
    statistical method, 86
    transformation-based, 86
conditional probability, 18, 21
connectivity search, 41
context change, 108
context-based non-linear prediction, 2
convolution filter, 50
cross correlation, 16
cryosection, 4, 131
decay
exponential, 27, 30
linear, 27
quadratic, 27
definition
    global, 10
    image
        alignment, 11
        registration, 11
    local, 11
    near-lossless, 106
deformable image sequence alignment, 125
density-based clustering, 42
digital brain atlas, 1, 4, 6, 37, 107
    vector-based, 70
    voxel-based, 85
dilation, 128
dissertation organization, 8
distribution
    Laplacian, 21, 97
    prediction error, 111
DPCM, 87
    context-based non-linear, 89
dynamic
    access, 85
    programming, 35
entropy, 17
    joint, 17
erosion, 128
exhaustive search, 5, 36, 38, 46
false positive result, 15
feature
    extraction, 126
    space, 14
fiducial mark, 15
filter
convolution, 50
Gaussian, 131
morphological, 128, 131
flood fill, 54
Fourier transformation, 16
fractal, 88
frequency domain, 16

Gauss-Newton method, 33
Gaussian filter, 131
gene, 40
genetic algorithm, 7, 35
global
alignment metric, 7, 10
definition, 10
optimization, 34, 126
heuristic, 7
gray code, 86
greedy algorithm, 35

Harr wavelet transformation, 50
Hessian matrix, 33
Hilbert
curve, 108
traversal, 54, 113
locality property, 55
histogram, 97, 107
equalization, 132
homogeneous coordinate, 13, 48
Hough
space, 131
transformation, 132
Huffman
code, 2
coding, 86
hybrid optimization, 32

image
error concealment, 51
interpolation, 5
modality, 17
postprocessing, 7
registration, 11
definition, 11
fundamental components, 14
sequence alignment, 10, 11, 20
definition, 11
deformable, 125
information theory, 17
integer wavelet transformation, 62
interpolation
linear, 56
multi-directional, 52
nearest neighbor, 6
joint entropy, 17

L-system, 56
Lagrange multiplier, 32
Laplacian
distribution, 21, 97
pyramid, 88
least square, 27
levels of detail, 74
Lindenmayer system, 56
linear
code, 2, 69
decay, 27
interpolation, 56
octree, 76
regression, 25
weighted, 27
linearization, 116
local
definition, 11
extremum, 37
optimization, 32
transformation, 14
long range correlation, 52, 63
lossless compression, 3, 87
LZ77, LZ78, LZW, 86

MEBP, 10, 20
GAME version, 22
PSM version, 22
morphological filter, 53, 128, 131
2D kernel, 129
3D kernel, 129
MSPE, 89
mutual information, 18
normalized, 18
n-body problem, 126
near-lossless, 106
quantization, 109
Nelder-Mead downhill simplex method, 33
Newton’s method, 33
objective function, 5
oblique section, 80
octree, 100
index, 103
variant, 2
optimization
calculus-based, 32
direct search, 33
dynamic programming, 35
genetic algorithm, 35
global, 34, 126
goal, 35
gradient-based, 33
greedy algorithm, 35
hybrid, 32
local, 32
multi-resolution, 36
pyramid, 46
simulated annealing, 34
orthogonal directions, 93
outlier intensity, 16
phase change, 16
plate stacking problem, 7, 23
POCS, 52
power spectrum, 17
prediction
error, 21
method, 25
prefix code, 95
progressive coder, 112
PSNR, 28
quantization, 121
near-lossless, 109
Quasi-Newton method, 33
raw image

acquisition, 4
problems, 4
region-of-interest, 126
run length encoding, 86, 94, 99
salient feature, 15
scalable
hyper-space file, 2
linear code, 71
scalar quantization, 86
scan order
Hilbert, 108, 109
raster, 55, 108
search space, 5, 19
discrete and finite, 14
SHSF, 2, 68, 94
signature extraction, 64
similarity metric
cross correlation, 16
entropy-based, 18
intensity-based, 16, 19
joint entropy, 17
mutual information, 18
normalized mutual information, 18
ratio image uniformity, 17
sum of
absolute intensity differences, 16
squared intensity differences, 16
variance of intensity ratios, 17
simplex method, 47
simulated annealing, 7, 34
SLC, 71, 74
SMP system, 79
spanning tree, 121
state diagram, 59
structuring element, 128
subgraph, 44
survey
clustering algorithm, 39
image
error concealment, 51
registration, 11
octree, 68
quadtree, 68

139
Swiss-cheese-type image defect, 50, 53

termination condition, 42
texture synthesis, 126
thresholding, 131
transformation, 12
  affine, 13
  global polynomial, 14
  local, 14
  perspective, 14
  primitive, 12
  reflection, 14
  rigid body, 12
  shear, 13
translation invariant, 17
triangular surface mesh, 84

uniquely decipherable, 95

volume compression, 85
volumetric data set, 1

wavelet
  coefficient, 114
  transformation
    Harr, 50, 59
    integer, 62, 113
weight function, 27

z-order, 71
References


Biography

Rongkai Zhao received his BS degree in chemistry from Fudan University in 1997. After worked for a renowned fortune 500 company, PRAXAIR, he decided to pursue post graduate education in University of Illinois. While in Illinois, he received MS degree in computer science from University of Illinois at Urbana-Champaign in 2001 and published the research results in a number of academic forums sponsored by IEEE, NSF, NIH, SFN and SPIE. His current research interest is neuro-informatics, bio-medical imaging, data compression and reconfigurable computing.