11Figure 3.18

CONSTRUCTION OF A THREE-DIMENSIONAL GEOMETRIC MODEL FOR SEGMENTATION AND VISUALIZATION OF CERVICAL SPINE IMAGES

A DISSERTATION
SUBMITTED TO THE PROGRAM IN MEDICAL INFORMATICS
AND TO THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Ramani Pichumani
January 1997

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Walter Murray, (Principal Advisor)
Department of Engineering-Economic Systems
and Operations Research

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John R. Adler, Department of Neurosurgery

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Russ B. Altman, Department of Medicine

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Michael G. Walker, Department of Medicine

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

To my parents and my sisters, for all the love,
inspiration, and motivation they have provided
so patiently and graciously.

CONSTRUCTION OF A THREE-DIMENSIONAL GEOMETRIC MODEL FOR SEGMENTATION AND VISUALIZATION OF CERVICAL SPINE IMAGES

Advisor: Walter Murray

The major goal of this dissertation is to develop a set of techniques for automatically extracting vertebral segments from three-dimensional computerized tomography (CT) and magnetic resonance (MR) images of the human cervical spine. An important motivation for this work is to provide accurate information for registration and for fusion of CT and MR images into a composite three-dimensional image. One of the major hurdles in performing image fusion is the difficulty of extracting and matching corresponding anatomical regions in an accurate, robust, and timely manner. The complementary properties of soft and bony tissues revealed in CT and MR imaging modalities makes it challenging to extract corresponding regions that can be correlated in an accurate and robust manner. Ambiguities in the images due to noise, distortion, limited resolution, and patient-specific structural variations also create additional challenges. Whereas fusion of CT and MR images of the cranium have already been performed, no one has yet developed an automated technique for fusing multimodality images of the spine. Unlike the head, which is relatively rigid, the spine is a complex, articulating object and is subject to structural deformation throughout the multimodal scanning process.

The hypothesis of my research is that by using finite-element techniques and model-based segmentation algorithms, it is possible to extract vertebral segments and determine anatomical landmarks robustly from three-dimensional CT image sets of the human cervical spine, to an accuracy of 95 percent or better. The accuracy is defined as the average sensitivity and specificity of the extracted vertebrae when compared against a gold-standard reference. The original contribution of this dissertation is to demonstrate the feasibility of a unified approach to visualization, modeling, and perception using finite-element techniques and optimization methods.

Although the techniques proposed here have been designed to work with both CT and MR images, *I have only performed a rigorous test of the validity of the hypothesis on CT images.* Whereas the performance of these techniques on MR images is of significant interest, I have chosen not to perform such an evaluation. A major contribution of this research is the construction of a generic finite element geometric model of the cervical spine.

Approved for publication:

By: ____________________________________

For Department of Medicine
Medical Information Sciences Program

Preface

The research presented in this dissertation has been driven by the need to enhance the diagnostic capabilities of Computerized Tomography (CT) and Magnetic Resonance Imaging (MRI). Having the capability to merge the complementary information from CT and MR images would be useful for patient diagnosis, as well as for treatment planning. I have written this dissertation for an audience consisting of physicians, computer scientists, and engineers. Only a basic understanding of the principles of these disciplines is assumed. Where a greater understanding of a particular field is required, I have included background material on the principles and techniques being used, to help the reader.

In Chapter 1 of this dissertation, I give an overview of my proposed research and how it fits into the context of the Stanford/VA SPIF Project. Chapter 2 is a review of image-understanding techniques, starting with simple threshold-based approaches and moving into more recent techniques, such as active contour models. In Chapter 3, I describe the methodology used to perform the model-based extraction of the vertebral segments. I also provide an overview of the Spinal Image Fusion project, and describe several techniques being investigated by the other researchers on the project. In Chapter 4, I present the results of various experiments to demonstrate the functionality of the extraction technique. In Chapter 5, I describe the evaluation method for validating the performance of the techniques presented in chapters 3 and 4, in particular, the technique for evaluating the accuracy of the model-based segmentation technique using a gold standard. I also present the results of this evaluation, and discuss the strengths and weaknesses of the techniques. Finally, in Chapter 6, I summarize the research techniques and describe the lessons learned in this dissertation. I also discuss directions for future research. In Appendix A, I introduce the essential spinal anatomy needed to understand the context and the language of this research. I do not, however, provide an exhaustive treatment of the subject. Readers interested in learning more about spinal anatomy will find an excellent source in Oliver and Middleditch [62]. The remaining appendices describe formulations of various computational techniques used in this dissertation and the software libraries that I created in order to implement these techniques.

Acknowledgments

One of the corollaries of writing a lengthy dissertation is that there are a vast number of individuals who helped make this work possible. I would like to start by thanking Dr. John Adler for providing the inspiration, motivation, and support for this research. Certainly without John's vision and energy, there would be no Spinal Image Fusion project at Stanford. Professor Walter Murray provided invaluable technical advice as well as support and encouragement to make this research possible. Walter has been an outstanding mentor and friend and has always been willing to take the time and effort to share his knowledge of applied math and optimization methods with his students. Dr. Russ Altman and Dr. Michael Walker have also provided invaluable technical and academic advice that shaped and solidified this dissertation. Russ and Mike have coached and guided me through all the academic hurdles that every doctoral candidate invariably confronts. I would also like to thank the entire Medical Information Sciences Executive committee, in particular, Dr. Ted Shortliffe, for providing me the opportunity to work with very talented people in an outstanding environment. The challenge, stimulation, and knowledge I have experienced here have been without equal. Professor Sandy Napel's and Professor Tom Binford's expertise in image understanding have enriched my knowledge of this field. With her expert scrutiny, Debby Fife provided exemplary editorial feedback that helped make this dissertation easier to read and understand. Dr. Gary Glover and the entire staff of the Lucas Research Center have also provided me with a rich environment for conducting this research. Dr. John Drace has provided much support for the Spinal Image Fusion project with his valuable knowledge of radiological imaging. Professor David Heeger deserves special thanks for chairing my oral defense committee and for taking an interest in my work. Despite the best efforts of the aforementioned, this dissertation has its share of flaws for which I deserve sole credit.

I would also like to thank my friends who went above and beyond the call of duty in helping with this dissertation. Ethiraj Venkatapathy provided friendship and support, as well as expertise in CFD techniques. Mary Washburn's legendary TEXpertise, editorial skills, systems knowledge, and her overall friendship and support have been indispensable to me. Dulce Ponceleon was always willing to help with a good laugh and is living proof that Stanford is a fun place to study. The imaging informatics "support group" (Maria Tovar, Smadar Shiffman, and Rhea Tombropolous) was always ready and willing to provide support and encouragement. All of my fellow students and colleagues in the Medical Information Sciences program helped with their probing questions, insights, and their overall enthusiasm. Darlene Vian and Rosalind Ravasio and the entire Section on Medical Informatics support staff helped maintain a friendly and productive environment that was simply a joy to work in. And most important, my parents and my sisters deserve credit for patiently providing love, inspiration, and motivation for my work. It is to my family that I dedicate this entire work.

This dissertation would not have been possible without the support of the Department of Neurologic Surgery at Stanford, the National Library of Medicine's training fellowship in Medical Informatics, the Department of Veterans Affairs in Palo Alto, the Section on Medical Informatics at Stanford, the Radiological Sciences Laboratory at Stanford, and the Richard Lucas Foundation. John Flynn of Silicon Graphics Incorporated donated computing equipment to the Spinal Image Fusion project. Additional computing facilities were provided by the Center for Advanced Medical Informatics at Stanford, which is supported by grant number LM05305 from the National Library of Medicine. The entire Symbolic Systems Support Group at Stanford deserves recognition for keeping the computing facilities running smoothly.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Spinal injuries and degenerative disorders of the spine represent an area of medicine for which current treatments remain inadequate. Most patients who experience spinal pain or disability are usually managed with a combination of pain-relieving drugs and physical therapy. For acute disorders or traumatic injuries, surgery is the preferred treatment. However, recent studies have shown that 25 to 40 percent of patients who undergo spinal surgery will experience unsatisfactory outcomes (e.g., persistent pain or disability) [37].

One component of uncertainty in the management of spinal disorders stems from the fact that current understanding of the physical and biomechanical properties of the spine is inadequate to predict the outcomes of various treatment plans. The spine is a complex, articulating, load-bearing structure that is responsible for supporting the head, absorbing vertical shock, maintaining correct posture, and protecting the spinal cord, while maintaining the flexibility needed for proper mobility. Because the vertebral column shields the spinal cord, many neurological disease processes are difficult to visualize and identify by routine examinations.

Computerized Tomography (CT) provides excellent views of bones and other dense structures inside the human body. Magnetic Resonance Imaging (MRI) provides excellent views of soft tissues. The most difficult spinal trauma cases, however, usually involve both these tissue types. Currently, physicians must merge CT and MR images mentally across many slice planes to form a complete three-dimensional picture of the patient's disorder. Although this merging can be a difficult and error-prone process for humans, it is ideally suited to computers.

Despite these limitations, modern imaging techniques help neurosurgeons and neurologists make more accurate diagnoses and plan more effective treatment strategies for spinal disorders than would otherwise be possible (especially in those cases involving structural abnormalities). In some cases, imaging studies play a critical role in properly evaluating a spinal disorder [32]. **Multimodality image fusion** is a technique whereby complementary images acquired using different modalities (e.g., CT and MRI) are combined to provide more meaningful information to a clinician. Neurosurgeons who treat patients with spinal disorders would like to be able to visualize the spatial relationships among the various soft and hard tissue types. Hence, multimodality image fusion would be useful for diagnosing and treating spinal pathologies where hard and soft tissues interact very closely. However, although much research has been published in the area of multimodality image fusion, no work has been reported on spinal image fusion (as of this date).

1.1 Project Goals

The context for the work described in this dissertation is the Stanford/Veterans Administration Spinal Image Fusion (**SPIF**) project, a joint research project involving the Department of Neurologic Surgery at the Stanford University Hospital, the Palo Alto Veterans Administration Hospital, the Section on Medical Informatics at the Stanford University School of Medicine, and the Radiological Sciences Laboratory at the Stanford University Medical Center. The goal of the Spinal Image Fusion Project is to generate fused images of the cervical spine *automatically* by using a retrospective registration technique applied to CT and MR images acquired with a predetermined imaging protocol. The work reported in this dissertation deals with the automated feature-extraction techniques needed to perform image registration.

Fused images will provide physicians with a surgical planning and simulation tool that is presently unavailable in any research or clinical setting. Physicians will be able to examine, manipulate, and perform quantitative measurements on the merged image sets using high-resolution graphical workstations. Although there are many approaches to the problem of **image fusion**, the goal of this project is to develop *automated* fusion techniques to reduce turnaround times, to aid in diagnosis, and to improve the success rate for treatment of spinal disorders.

Although spinal image fusion has clinical value in itself, the benefits of this project will extend to other research areas as well. In fact, the *long-term goal* of the SPIF project is to apply the knowledge gained from fusing spinal images to construct accurate **biomechanical models** of the human spine. By borrowing from structural analysis techniques used in the engineering disciplines (e.g., finite-element analysis and continuous solid modeling), physicians should be able with these models to simulate both the static and the dynamic behavior of the spine under various operating conditions, and to predict the outcomes of various interventional techniques such as vertebral fusion, disk resection, weight loss, bracing, and muscle strengthening.

Several researchers have constructed structural models of the spine using both finite-element (FEM) techniques and continuous models [59] [13] [36]. However, no one has developed a method for constructing such models on a patient-specific basis. Such models can only be constructed by incorporating structural information gathered from CT and MRI studies of individual patients. Simulations of interventional techniques could then be done on a patient-specific basis, and physicians could tailor their interventions so as to maximize the stability of the spine and minimize the pain and disability experienced by the patient.

1.1.1 Overview of Image Fusion

In general, image fusion is a process whereby images from various detectors (at possibly different orientations) are merged to provide a more complete picture of a scene or object under identification. *Multimodality* image fusion means that the various detectors use different imaging principles to measure different physical properties. For example, CT imaging relies on the X-ray attenuation properties of various materials, while MR imaging relies on the net nuclear magnetic spin of certain atoms, in particular, hydrogen [45]. Some researchers have loosened the definition of image fusion to include any **visualization** of registered image sets [38]. Visualization refers to the presentation of large or complex data sets in a graphical manner. For example, using the prototype SPIF system, a clinician can view a CT image slice side by side with an MR image reformatted at the same slice orientation. The clinician can select a point on the CT image and the corresponding point on the MR image will be highlighted. Other forms of image fusion include *picture-within-a-picture* (a specified region from one modality is pasted onto the other modality) and *window shading* (the user can move a horizontal or vertical bar that reveals one modality underneath another). Strictly speaking, multimodality image fusion can be thought of as a subset of multimodality image visualization.

Before images from different modalities can be fused, they must first be *registered*. The basic principle behind multimodality **image registration** is to find the optimal geometric transformation that will map image pixels from one modality onto corresponding pixels in a different modality. Image registration is a subset of image fusion because registered images typically do not undergo anything other than geometric transformations. On the other hand, image fusion typically involves classifying various image regions and applying further operations such as rendering and morphological and algebraic set operations (e.g., union, intersection, exclusion).

Different categories of registration techniques can be distinguished by characteristics such as rigid-body versus non-rigid-body transformation, local versus global transformation, two-dimensional versus three-dimensional representation, and pixel-based versus structure-based (object) matching. An exhaustive review of these techniques is beyond the scope of this dissertation and interested readers should consult [81] for an excellent treatment of this topic.

1.1.2 The SPIF Approach

The image-fusion methodology proposed by the Stanford/VA SPIF project can be categorized as **voxel substitution** using a **local**, **rigid-body**, three-dimensional, **model-based**, **landmark-matching** registration technique. In this dissertation, a *landmark* is defined as a two- or three-dimensional anatomical structure that is used to establish a reference for orientation of structures within an image. Although landmarks can be made up of points, curves, or surfaces, they will typically have unique locations (such as origin or center). Complex landmarks can have other related parameters such as length, area, volume, curvature, axes of orientation, etc. The landmarks used for spinal image fusion include disk/bone boundaries, vertebral bodies, facets, and foramina (these landmarks are discussed in greater detail in Chapter 3).

The landmark-extraction technique developed here is called *model-based* because it relies on a high-level model of the vertebrae to be extracted. Several researchers have claimed that machine recognition of objects without the use of a high-level model is an ill-posed problem (i.e., it cannot have a robust or accurate solution) [58]. A model-driven system can reduce the size of the solution search space by pruning out input combinations that are improbable given the constraints of the model [7] (Section 2.2.3 discusses some of these aspects in greater detail). Levitt [51] argues that an explicit model can be used to document assumptions made about structures to be imaged, and to assign algorithmic parameters in a semantically meaningful way. Furthermore, Levitt claims that it is easier to modify the behavior of an explicit model-driven system than it is to modify an implicit collection of algorithms that may perform the same function.

Statistical modeling can be a useful technique for dealing with the ambiguity that is inherent in the process of image formation and interpretation. In addition, researchers have applied statistical-modeling techniques to analyze the natural variations seen in biological structures [16]. A statistical model that captures the geometric characteristics of interest can be used to set up and guide the operation of low-level image-processing algorithms. Hence, a high-level geometric model can provide a context in which the lower-level processing modules would operate. In addition, such a model can synthesize a multitude of information provided by low-level modules (some of which may be conflicting) into a small set of manageable hypotheses that could be ranked in order of maximum likelihood.

At present, the SPIF system has the ability to display registered CT and MR images of the head in a side-by-side or a windowpane format. Registration is performed by matching three-dimensional surface triangles representing the skull in the CT images with corresponding skin-surface points in the MR images. An optimal rigid-body transformation matrix is determined by minimizing the average displacements of the MR points from the CT surface triangles. Hence, this system uses a *global, rigid-body* transformation to perform image registration of the head. The skull boundaries are extracted from the CT and MR images using an interactive edge-editing program whose edges are extracted by a Canny edge-detection algorithm. The CT surface triangles and MR surface points are fed into an optimization program, which determines the six rotation and translation parameters in R^3 that minimize the total least-squares distances between the points and their neighboring surfaces.

The SPIF system currently performs two-dimensional visualization of fused images using a *voxel-substitution* technique, which takes voxels that correspond to bone in MR images and recolors them (i.e., substitutes them) with voxels from the CT image. In this manner, the clinician can use the system to visualize bony structures overlayed on top of soft tissues in a single composite image (see Figure 1.1).

Although the SPIF system has performed adequately on cranial images, there are several limitations that need to be addressed in order for the system to work on spinal images. First, because the skull is treated as a single monolithic entity, the current system can only register one object at a time. In

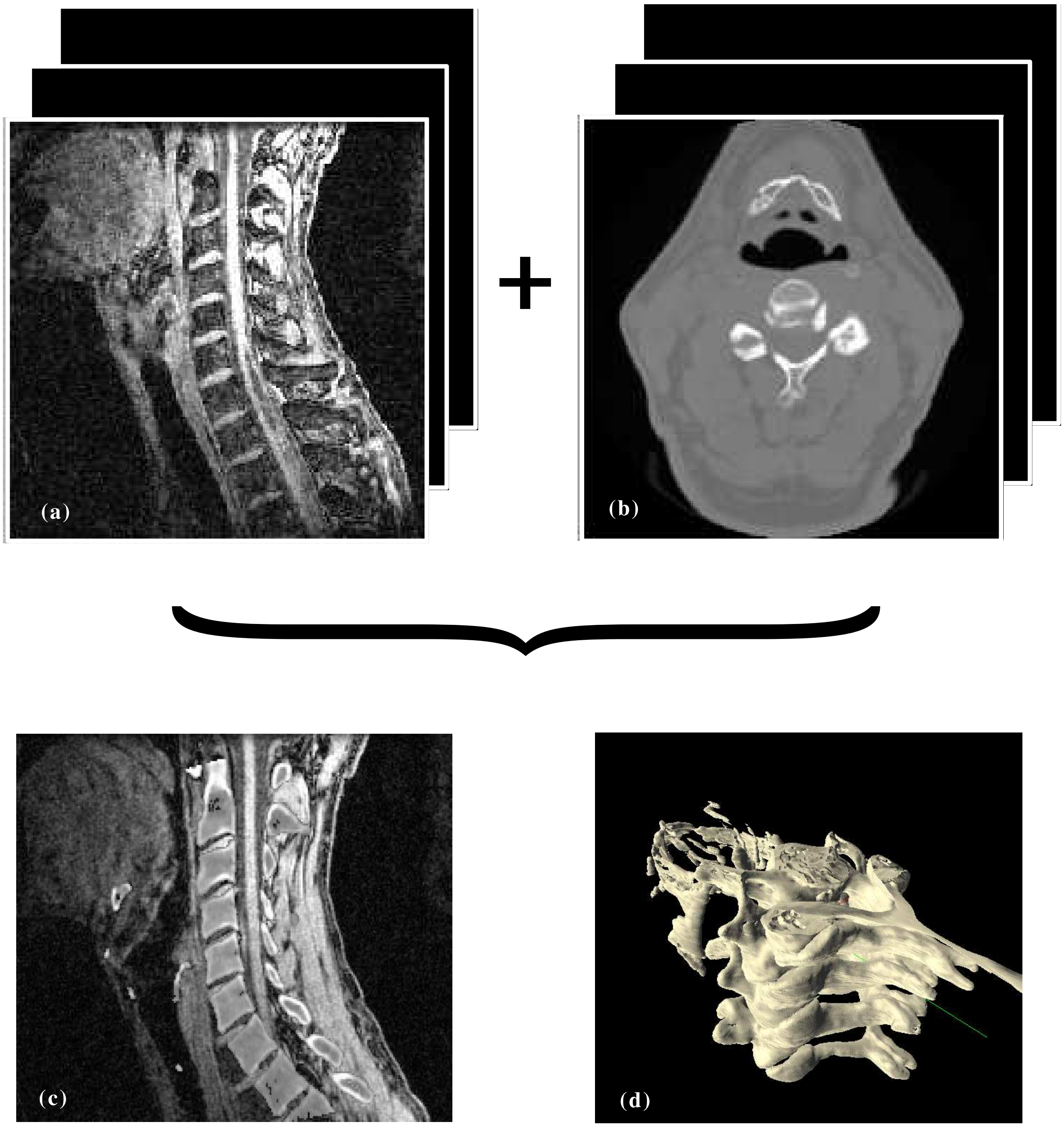


Figure 1.1: The SPIF system can be used to combine two-dimensional CT and MR images (shown in figures a and b, respectively) into either two-dimensional registered image sets (shown in figure c), or into three-dimensional volumes (shown in figure d).

contrast, the spine consists of several vertebral bodies, which can move relative to one another in some constrained manner. Second, the interactive boundary-extraction process is very labor intensive and requires a significant degree of user judgment. As a result, this step is prone to errors and biases that can affect the accuracy and repeatability of the registration process. Third, the smaller size and greater geometric complexity of spinal disks and vertebral bodies make it more difficult to perform accurate registration. Although it is possible to reduce the *field of view* of the images to accommodate the smaller sizes of individual vertebral segments, the curved shape of the overall cervical spine requires a greater field of view than would be desirable for each individual spinal segment. Finally, the skull is a simply connected geometric object. That is, any two points within the skull can be connected by a path that remains within the same volumetric region. On the other hand, the vertebral bodies are not simply connected domains and require more complex surface and region descriptions. To make matters worse, each individual vertebral body is simply connected in some slice planes and multiply connected in other slice planes. As a result, any surface extraction and reconstruction algorithm must take into account such dichotomies. Until now, the SPIF system has lacked an anatomical model for dealing with such situations.

1.2 Dissertation Goals

One of the major hurdles in performing image fusion is the difficulty of extracting and correlating anatomical landmarks in an accurate, robust, and timely manner. Although several researchers have successfully performed cranial image fusion [39] [82] [42], there are no techniques for performing fusion of cervical spine images at this time.

Current techniques for registering CT and MR images are not adequate for spinal images. These techniques make rigid-body assumptions that are valid for the skull, or elastic-body assumptions that are valid for the brain. Neither of these assumptions are valid for the spine, which consists of vertebral segments that are rigid bodies, yet movable. In addition, *none of the existing registration techniques were designed with the ultimate goal of building and using biomechanical models*.

Although dozens of techniques have been developed to perform segmentation of *two*-dimensional CT and MR images, none of these techniques has been shown to work on complex *three*-dimensional images of the cervical spine. The goal of this dissertation was to develop *automated* techniques for extracting boundaries of vertebral segments from three-dimensional CT and MR images of the cervical spine. This was a crucial requirement for the SPIF system, because the complementary tissue characteristics of CT and MR imaging modalities makes it difficult to extract matching landmarks that can be correlated in an accurate, robust, and timely manner. With this in mind, the hypothesis of my research can be summarized in the following statement:

By using finite-element models, optimization techniques, and model-driven segmentation algorithms, it is possible to extract vertebral segments robustly from three-dimensional CT image sets of the human cervical spine (C3 through C7), to an accuracy of 95 percent or better. The accuracy is defined as the average *sensitivity and specificity of the segmented vertebrae compared with reference gold standards created by expert human segmentation.*

The results of the quantitative evaluation have shown this hypothesis to be valid. Although the techniques designed were intended to work with both CT and MR images, I have performed a rigorous test of the validity of the hypothesis only on CT images. Whereas the performance of these techniques on MR images is of significant interest, due to resource limitations they will not be evaluated for this dissertation. The first and second cervical segments (C1 and C2) are of less interest to neurosurgeons and have been omitted in the SPIF project. A major contribution of this research will be to improve the overall speed, accuracy, and robustness of the image-registration process by automating the vertebra-extraction task. Chapters 2 and 3 provide more detailed descriptions of the algorithms developed and/or investigated during my research.

In the process of validating the hypothesis of this research, I also developed an appropriate technique for measuring the accuracy of the extracted features. However, a measure of accuracy is useful only in the context of an evaluation methodology. Therefore, I devised an evaluation plan that uses a gold standard both for training and for testing the extraction algorithms. Section 1.4 and Chapter 5 provide more detailed descriptions of the evaluation plan.

1.3 Summary of Methodology

The first step of the extraction method is to construct a three-dimensional finite-element surface model of the vertebral elements to be extracted using a modified polyhedral tessellation algorithm (see Chapter 3). The volumetric CT image data are first edited to remove any ambiguities or artifacts before being fed to the tessellation algorithm. The output of the tessellation algorithm is a set of triangular surface patches comprising each individual vertebra. A smoothing and triangle-decimation step is then applied to reduce the number of triangular surface elements used in the model. The CT data set applied to construct the model was selected from a sample of patients and was chosen on the basis of having geometric properties that are closest to the mean of the sample data (i.e., the average data set). This data set, along with a set of parameterizations to account for geometric variations seen in the sample data, will constitute a *generic* spine model. Because there is no generic individual, it is unlikely that a single generic model can suffice for guiding the extraction algorithm. Instead, several generic models are used to account for the expected variations seen in the patient population.

Once the generic models have been constructed, *patient-specific* models can be constructed using a three-step process. The first step is to perform a crude match of the generic model to the patient data by applying a homogeneous transformation matrix to the vertices of the generic model. Figure 1.2 shows a sample generic model used as a starting point in this technique. The transformation matrix scales, rotates, and positions the generic model so that it matches the vertebra in the patient data as closely as

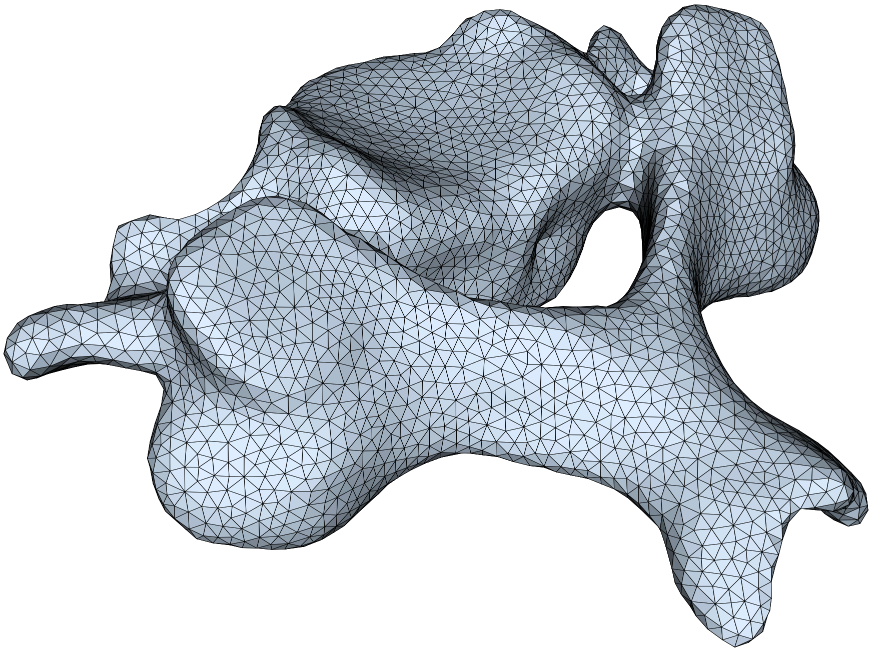


Figure 1.2: A generic finite-element vertebral model used by the extraction algorithm. This particular model has 9672 triangular surface elements.

possible. The matrix is computed by locating as many of the predetermined landmarks as possible from the patient data. A minimum of four non-coplanar landmark points must be located in order to construct this matrix. In the second step, the locations of the triangle vertices of the model are adjusted so as to minimize their distances to the surface of the vertebra in the CT data set. A *figure of merit* is computed, which is based on the total number of vertices that match the data and the degree to which they match. Using this figure of merit, the third step is to adjust the transformation matrix to yield a better match by using the NPSOL subroutine, a sequential quadratic programming (SQP) algorithm developed by Gill, Murray, Saunders, and Wright [34]. Steps 2 and 3 are repeated until an optimal fit has been achieved, at which point the patient-specific model has been determined. Figure 1.3a shows the generic model after it has undergone this process. The surfaces shown in blue correspond to regions where there is no boundary information needed to unambiguously segment the vertebral data. Adjacent to the blue regions are large spikes that are caused by false negative boundaries. This illustrates the problems encountered when using thresholding or edge-detection methods on data that have missing or ambiguous boundary information. To fix these problems, I have developed a kinematic relaxation algorithm that smoothly merges information from the generic model with the modified generic model in those regions where the data is incomplete or ambiguous. Figure 1.3b shows the generic model model after the relaxation process. This model has now become a patient-specific model and can be used to extract and fuse the bony vertebra with the MR images, or to generate the biomechanical model for simulation or visualization purposes.

1.4 Evaluation

I recruited several radiologists to generate a gold standard reference model for evaluating the results of the extraction algorithm. Because radiologists are trained to work with two-dimensional images, it is straightforward to generate a gold standard model by outlining the boundaries of vertebrae from cross-sectional slices. Using these boundaries, three-dimensional vertebrae are extracted to create isolated volumetric representations as well as tessellated surface models. The volumetric vertebrae are used to perform a voxel-by-voxel comparison with the output of the extraction algorithm. The false positive and false negative voxels are then tallied to compute the overall sensitivity and specificity of the proposed solutions. Using this method, I found the average sensitivity and specificity of the machine segmented vertebrae to be 94 percent and 99.9 percent respectively. In Chapter 5, I describe in detail the methodology used to compute these results.

1.5 Review of Previous Work

A review of the literature having to do with MR and CT image matching, segmentation and tissue-classification algorithms, and modeling methods for soft and bony tissues reveals little that is directly applicable to spinal images.

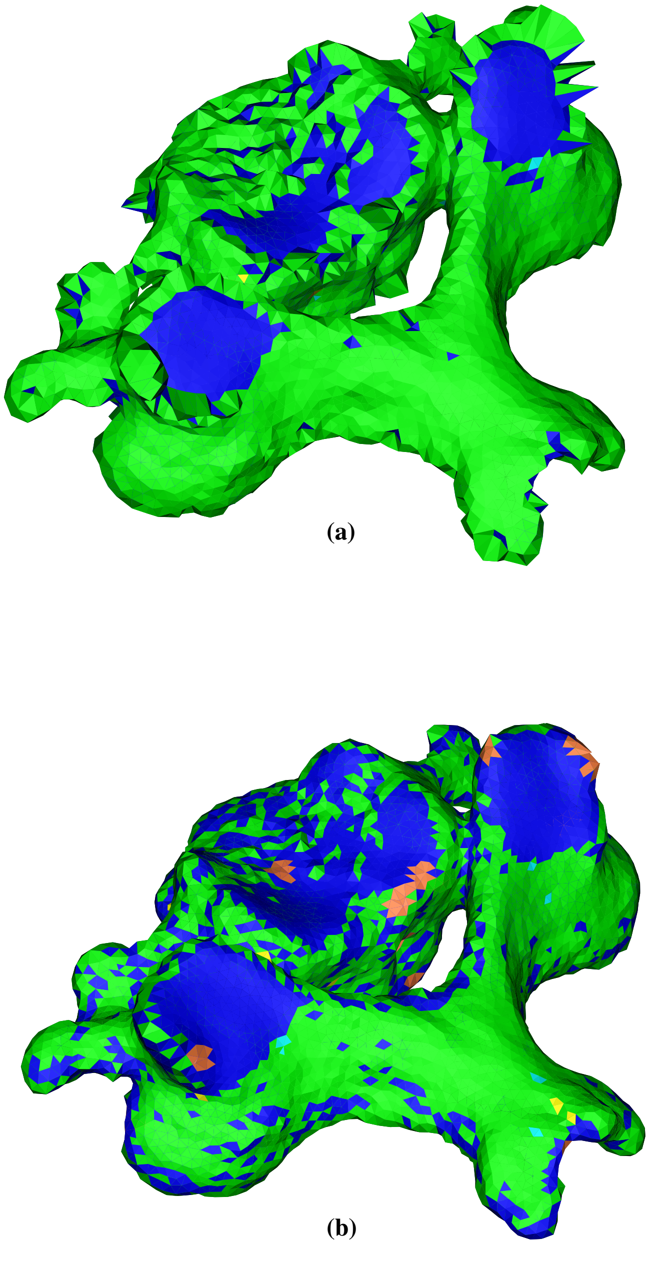


Figure 1.3: The surface elements of the generic model have been modified to match the local variations of the patient data as depicted in (a). Due to ambiguities in the patient data, some boundary information is either missing (as depicted by the blue surfaces) or is incorrect (as depicted by the surface spikes). After the kinematic relaxation process has been performed (b), the surface information from the original generic model has been smoothly merged into the adapted model in those regions where the model is incomplete or incorrect. The orange regions correspond to the fixed surface spikes.

Segmentation and Matching Techniques

To date, no one has developed a system for fusing automatically CT and MR images of the spine, using either manual or automated methods, although several research groups have developed automated techniques for registering CT, MR, and/or PET images of the head (see Appendix B for more detailed descriptions). Furthermore, no one has developed a technique for automatically extracting vertebrae from either CT or MR images. Because model-based segmentation of images can be viewed as a matching process, I have summarized the results of various image-matching techniques to provide a background review of current segmentation and fusion techniques for medical images. If one of the images is derived from a known reference model (e.g., an atlas), matching and segmentation can be viewed as being equivalent tasks.

Gamboa-Aldeco and Fellingham [33] developed one of the earliest retrospective surface-based methods for registering CT and MR cranial images. Their method determines the optimal global affine transformation matrix by matching the zeroth-, first-, and second-order moments of the two data sets. They also developed a system for manually displaying outlined anatomical features from one data set in the context of the second data set. They can also display composite images showing the merged data with the outlines superimposed onto their respective features. Unfortunately, the authors did not present any data on the absolute accuracy of their matching technique using actual image data. However, using a plastic model of a vertebra, they found the misregistration error of their technique was 2.3 mm $\pm$ 0.7 mm in the worst case. They also measured the effects of changing various parameters such as slice spacing, range of the scans (i.e., total scan height), and the thresholds used to determine the surfaces. They found that the biggest error contribution was due to the threshold parameter used in the segmentation process. The number of slices and the range of the slices did not affect the accuracy as much since the overall symmetry and volumes of the object were still similar.

Hill and his colleagues [41] [40] developed an automated rigid-body correlation-based technique for registering CT and MR cranial images (see Section B.8). Their technique precomputes the cumulative variance of the ratios of intensity values as a function of the degree of misregistration. Using this empirical information, they can quickly register CT and MR images accurately to within 2 mm. Although their technique is useful for the head, it is not clear how well it would perform on the spine where the rigid-body assumption does not hold.

One of the earliest automated CT/MR/PET cranial image-registration techniques was developed by Pelizzari and his colleagues [66] [50]. Their technique forms a three-dimensional model of the outer surface of the head and tries to find the optimal rigid-body transformation parameters that minimize the total distance between points on one model to the surface of the other model. Their technique is noteworthy not only because of its intrinsic accuracy, but also because of the sophistication of the methods used to evaluate the accuracy (less than 1 mm for CT/CT registration, 1 mm for CT to MR, 2 mm for MR to PET, and 2 mm for CT to PET). Although this technique performs adequately for cranial images, it cannot be applied directly to spinal images for two reasons: (1) the rigid-body assumption does not hold in the spine, and (2) the technique relies on accurate segmentation of the two objects that are to be registered.

Van den Elsen developed an automated retrospective technique for matching two- and three-dimensional CT and MR images of the head [81] [82] (see section B.7). She determines the optimal rigid-body transformation matrix that translates and rotates by maximizing a correlation between geometric features computed from both image sets. Her published results indicate that this technique can register two-dimensional image sets with a typical mismatch error of 0.75 pixels. Because her technique applies a global rigid-body transformation, it requires an extended control mechanism to be applied to spinal images on a per segment basis.

Szeliski and Lavallee developed a nonrigid-body (elastic) technique for matching three-dimensional facial surfaces acquired from laser range scanners [76] (see Section B.5). They use a three-dimensional warping function to provide a smooth deformation for matching the two data sets. Although they claim that their elastic technique produces more accurate results than a purely rigid-body technique, they require that the volumes to be registered can be modeled using parametric splines, and that the volumes have reasonably similar morphologies. Hence, their technique is not well suited for multimodality image sets, or where the imaged anatomies are not easily represented by spline functions.

Among the various types of registration techniques, van den Elsen's correlation-based approach appears to be the easiest to automate. Her technique requires minimal intervention on the part of the user and is based only on intrinsic image features, namely ridges, that can be found using a low-level mathematical operator. However, her technique still requires segmentation of the imaged anatomy to generate fused images.

Modeling Techniques

The spinal models developed to date can be categorized as either *static*, *strictly kinematic*, or *fully biomechanical*. The static models are used primarily for visualization and do not have a high degree of clinical significance. The strictly kinematic models deal with the dynamic behavior of body parts without considering the effects of masses and forces and the material properties of those parts [24]. These models are mainly used for animation and cinematographic purposes. The fully biomechanical models incorporate not only kinematic information, but also the material properties of the body tissues (e.g., mass, density, elasticity). From the standpoint of clinical relevance, the biomechanical models are the most useful, but they are also the most difficult to construct.

Although none of the biomechanical models are comprehensive enough to incorporate all of the hard and soft tissues involved in spinal mechanics, several researchers have made important initial strides in building experimental models with potential clinical significance. Dietrich, Kedzior, and Zagrajek [26] developed an experimental finite-element model of the spine that captures the cervical, thoracic, and lumbar sections as well as the rib cage. Their model incorporates vertebrae, disks, ligaments, and muscles acting directly and indirectly on the spinal column. With this model, they report they are able to determine the static and dynamic forces acting on the intervertebral disks and abdominal cavity as a function of external loads and muscle excitations. They also investigated the distribution of forces and displacements of the spinal system as a function of its overall shape. Unfortunately, the authors do not present any data to verify the accuracy of their simulations. Furthermore, their model was built using measurements from a single specimen of a nondegenerate human spine. Hence, it is not clear how to adapt their model to individual patients, nor is it clear that such a model can be used for image-understanding purposes. Breau et al. [13] constructed a three-dimensional finite-element grid for a ligamentous lumbar spine (L1 through S1) from 2 mm CT cross sections. Although the bony structures of their model were reconstructed from the CT images, the ligaments and intervertebral disks were generated analytically based on direct geometric measurements of the bone boundaries. Unfortunately, Breau did not perform any biomechanical simulations using the spine model. Despite the limitations of both these biomechanical models, they do provide a basis for further research into computer-based biomechanical simulation of the spinal system.

From the perspective of the SPIF project, image registration is not the end goal in itself. Instead, image registration will ultimately allow us to develop biomechanical models of the spine and to simulate the outcome of various treatment plans. For this reason, correlation-based techniques alone are not adequate. In fact, none of the current registration techniques are useful for modeling purposes. After having investigated all of these techniques, I believe that the most promising approach will be to develop a model-based registration technique that combines certain features of correlation-based methods with landmark-based methods.

All the matching techniques developed so far rely upon properties of the head that do not necessarily hold for the spine. First, the head is treated as single monolithic object. This implies that a single rigid global transformation, possibly combined with local corrections, is sufficient to register the data sets. Second, any interscan patient motion will not change the global or local geometry of the head. Hence, patient orientation should have little effect on the accuracy of the registration process. Third, none of the existing techniques uses a weighting scheme to trade off errors between critical anatomical regions and not-so-critical regions. Such a trade-off would require some knowledge of patient anatomy so that accuracy in unimportant regions can be sacrificed in favor of higher accuracy in more important regions. Such trade-offs would be difficult to perform without having some sort of model of the imaged anatomy.

More significantly, despite the varied degrees of success in performing image fusion, *no one has yet developed a unified approach that solves the combined problems of feature extraction, biomechanical modeling, and three-dimensional visualization*. All the previous methods described here have dealt with solving each problem in an isolated fashion, without sharing potentially useful information among the various methodologies.

1.6 Research Questions

During the course of this dissertation, I was guided by several questions that helped determine the feasibility of the original hypothesis:

1. What types of geometric modeling techniques are best suited for the cervical spine?
2. How reliably and robustly can the image features be detected?
3. What are the repeatable landmarks for which I should search?
4. How will various segmentation algorithms work on actual patient data?
5. Can the feature-detection process be fully automated? How would the accuracy of a fully automated process compare to that of a semiautomated process?
6. How well will the optimization algorithms achieve a best fit between the generic model and the patient-specific data?

These questions have influenced the design trade-offs that have been made in the segmentation algorithm. For example, landmarks that were easier to detect were chosen in favor of those that might provide somewhat more accurate information. Such landmarks will be easier for trained technicians to locate without having to rely on input from radiologists. In addition, these landmarks should also be easier for automated algorithms to detect.

1.6.1 Model development

I investigated various domain decomposition and modeling techniques and I decided to use a finite-element model (FEM) to represent the generic model of the cervical spine. This model will be used to provide a context for the feature-detection algorithms, to set up initial conditions, and to combine low-level information into higher-level hypotheses.

1.6.2 Repeatable landmarks

I consulted with a group of anatomists, radiologists, and surgeons to determine which landmarks they would find useful in identifying features in CT and MR images. These landmarks are described in more detail in Section 3.2.3. I also performed statistical analyses on CT data sets to determine which object landmarks appear to be most reliably located.

1.6.3 Boundary and region tracking

My initial examination of the output of several commonly used two-dimensional boundary detectors (Roberts [69], Prewitt [68], and Sobel [72]) has revealed that gradient-based edge detectors can produce edges that are not properly connected. Furthermore, none of these edge detectors handles ambiguous or missing edges in the data. They do not employ any higher-level knowledge about the objects or the edges within the objects to be detected. As a result, I developed a direction-weighted threshold/gradient boundary detector for locating the surfaces of the vertebrae.

1.7 Overview of Research Conducted

The goal of this research is to extract, robustly and accurately, vertebral elements from CT images. These elements can then be used to form the basis of patient-specific spinal models for performing biomechanical simulation. To achieve this goal, I conducted the following studies:

- Determined a set of useful object landmarks from expert radiologists and neurosurgeons
- Determined which of the experts' landmarks are repeatably detectable by the computer on CT and MRI data sets
- Used statistical methods to discover other landmarks and image features that are easily and accurately identified by the computer
- Developed a generic spine model and a high-level model-driven methodology for guiding the operation of lower-level segmentation algorithms
- Developed a technique for performing a crude match of a generic spine model to patient-specific data
- Used a non-linear sequential optimization subroutine (NPSOL) to refine the crude match to achieve an optimal fit of the spine model to the patient data
- Developed a system for demonstrating the performance of the model-based extraction algorithms using actual patient image data

One of the contributions of my research will be to reduce the amount of time needed to perform registration of CT and MR images when compared to semiautomated techniques. Existing techniques for multimodality image registration are inadequate for spinal images because they rely on either rigid-body assumptions or elastic-body assumptions. Neither of these assumptions are valid for the spine because the vertebral bodies are movable, rigid bodies. Furthermore, none of the existing techniques was designed to be used for building biomechanical models.

Another contribution of my research will be to develop a unified model that captures the structural properties of the cervical spine in a framework that can be used in several application areas. Not only will it be useful for image fusion and visualization, this framework can also be useful for building models for understanding the biomechanical properties of an individual's spine and predicting the static and dynamic behavior of the spine under different operating conditions.

Chapter 2

Survey of Current Techniques

Since the advent of computed tomography, methods for automatically detecting anatomical objects in CT and MR images have been active areas of research in the medical imaging community. Most of the published work has dealt with the application of such techniques to cranial images, and very little has dealt with the area of spinal images. The reasons for this are manifold. The anatomical scale of spinal elements is small compared with the field of view of the images. As a result, the spatial contrast of the features of interest is very poor and the features become very noisy and ambiguous. In addition, the topological characteristics of various spinal elements, such as the vertebral bodies and facets, are more complex than many of those found in the cranium. Finally, relative motion among the vertebral bodies can invalidate certain techniques based on rigid-body assumptions that are applicable to the cranium.

This chapter is intended to provide a review of existing image understanding methods and provide a motivation for the design of the feature-recognition techniques used in my research. I will start by reviewing some of the common approaches used in image understanding and explain the strengths and limitations of each approach.

2.1 Segmentation

If an image has been preprocessed appropriately to remove noise and artifacts, **segmentation** is often the key step in interpreting the image. Image segmentation is a process in which regions or features sharing similar characteristics are identified and grouped together. Image segmentation may use statistical classification, thresholding, edge detection, region detection, or any combination of these techniques. The output of the segmentation step is usually a set of classified elements, such as tissue regions or tissue boundaries. In the image matching process, those features that are easiest to identify accurately and repeatably on both image sets are candidates for *registration landmarks*.

Most segmentation techniques are either *region-based* or *edge-based*. Region-based techniques rely on common patterns in intensity values within a cluster of neighboring pixels. The cluster is referred to as the region, and the goal of the segmentation algorithm is to group regions according to their anatomical or functional roles. Edge-based techniques rely on discontinuities in image values between distinct regions, and the goal of the segmentation algorithm is to accurately demarcate the boundary separating these regions.

2.1.1 Region-based segmentation

Region-based segmentation methods attempt to partition or group regions according to common image properties. These image properties consist of

1. Intensity values from original images, or computed values based on an image operator
2. Textures or patterns that are unique to each type of region
3. Spectral profiles that provide multidimensional image data

Elaborate systems may use a combination of these properties to segment images, while simpler systems may be restricted to a minimal set on properties depending of the type of data available.

Thresholding

Thresholding is the simplest way to perform segmentation, and it is used extensively in many image processing applications. Thresholding is based on the notion that regions corresponding to different tissue types can be classified by using a range function applied to the intensity values of image pixels. The assumption is that different tissue types will have a distinct frequency distribution and can be discriminated on the basis of the mean and standard deviation of each distribution (see Figure 2.1). For example, given a two-dimensional image $I(x, y)$, we can define a *simple* threshold rule to classify bony tissue or a *compound* threshold rule to classify fatty tissue:

- Simple: (if $I(x, y) > I_0 \Rightarrow$ Bony tissue)
- Compound: (if $I_0 < I(x, y) < I_1 \Rightarrow$ Fat)

Thresholding has been used by Chow and Kaneko to segment ventricles from cineangiograms of the human heart [18]. They chose this technique because of the strong bimodal distribution of image values corresponding to regions that are interior and exterior to the ventricles.

2.1.2 Limitations of thresholding

The major drawback to threshold-based approaches is that they often lack the *sensitivity* and *specificity* needed for accurate classification. Sensitivity is defined as the *true positive rate* (**TPR**) for a function or a test that must detect the presence or absence of some intrinsic property (for example, tissue type) [74].

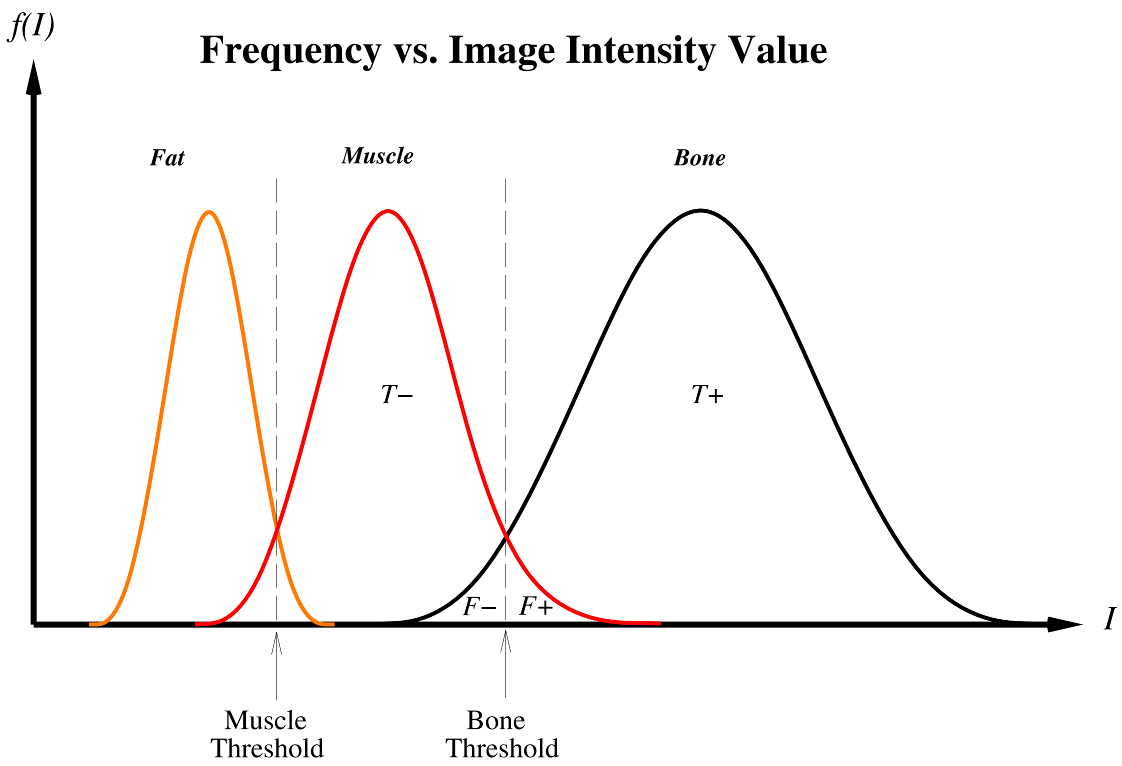


Figure 2.1: A hypothetical frequency distribution $f(I)$ of intensity values $I(x, y)$ for fat, muscle and bone, in a CT image. Low intensity values correspond to fat tissues, whereas high intensity values correspond to bone. Intermediate intensity values correspond to muscle tissue. $F+$ and $F-$ refer to the false positives and false negatives; $T+$ and $T-$ refer to the true positives and true negatives.

Hence, the purpose of the test is to determine, as accurately as possible, the presence or absence of this intrinsic property. Formally, the sensitivity of a binary test is defined as follows:

$$Sensitivity = TPR = \frac{\sum True+}{\sum Intrinsic+}, \tag{2.1}$$

where *True+* is defined as the number of samples that have the intrinsic property *and* were categorized by the test as positive, and *Intrinsic+* is defined as the *total* number of elements that have the intrinsic property (regardless of the outcome of the test).

In the previous example, the simple threshold rule for classifying bone will read the intensity values of each data element and categorize it as *true* if the value is above the threshold I_0 and *false* if it is not above the threshold. Hence, this test tries to determine an intrinsic property of an image pixel (i.e., bony = *true* or bony = *false*). In Figure 2.1, the area under the Bone curve and to the *right* of the Bone Threshold line is classified as *T+* because this test will correctly categorize such elements as being bone. The area under the Bone curve and to the *left* of the Bone Threshold line is classified as *F-* because this test will incorrectly classify such elements as not being bone. In this example, the sensitivity is a measure of how well this test can categorize as bone those tissues that are truly bone.

On the other hand, specificity is defined as the complement of the false positive rate (**FPR**):

$$Specificity = 1 - FPR = 1 - \frac{\sum False-}{\sum Intrinsic-}, \tag{2.2}$$

where *False-* is defined as the total number of samples that do not have the intrinsic value but were categorized incorrectly as *true*, and *Intrinsic-* is defined as the *total* number of elements that do not have the intrinsic property [74].

In Figure 2.1, the area under the Muscle curve that lies to the *right* of the Bone Threshold line is classified as *F+* because this test will incorrectly categorize such elements as being bone. Conversely, the area under the Bone curve that lies to the *left* of the Bone Threshold line is classified as *F-* because this test will correctly categorize such elements as not being bone. The specificity of this test is a measure of how well this test will reject from the bone category those elements that truly are not bone. Because their denominators are different, the sensitivity and specificity measures are not complements of one another. In fact, they measure two distinct aspects of any binary test: the ability to correctly reject false properties and the ability to correctly accept true properties.

It is important to realize that the definitions for sensitivity and specificity are valid for *any* test that performs binary categorization. No matter how sophisticated a test may be, it is still valid to ask what its sensitivity and specificity are.

As a rule, increasing the sensitivity of a binary test will reduce the specificity of the test, and vice versa. For example, by increasing the Bone Threshold in Figure 2.1, we can increase the specificity of the test because there will be fewer false positives. As a result, this will decrease the sensitivity because there will be fewer true positives. By plotting specificity versus sensitivity for a given test, we can construct what is known as a *Receiver Operator Characteristic* (ROC) Curve [74]. The ROC curve is a graphical representation of the trade-off between specificity and sensitivity for a given test (see Figure 2.2). We can see by examining Figure 2.2 that increasing the sensitivity will cause the specificity to decrease (more false positives).

In addition to depicting the trade-off between specificity and sensitivity for a given test, the ROC curve is also useful for comparing the performance of different tests on the same categorization tasks. For instance, suppose we had three tests (test 1, 2, and 3) that used three different techniques for categorizing bone. By overlaying the ROC curves for each of the tests on the same graph, we can visually determine which test performs better under various conditions. For example, in Figure 2.3, the test depicted by curve C_1 lies above and to the left of curve C_3. Hence, we conclude that test 3 is better than test 1 because for any given value of sensitivity, test 3 has a lower false positive rate (i.e., higher specificity). In practice, test 3 may be undesirable for reasons other than its power of discrimination. For example, test 1 may be a low-cost screening test for tuberculosis that will be given to a large population, whereas test 3 may be a more expensive chest X-ray exam. For a screening test, it is generally better to trade specificity for increased sensitivity so that fewer cases go undetected. The comparisons between test 1 and test 2 are less clear. Although test 1 is better than test 2 for high values of specificity, test 2 is better than test 1 for high values of sensitivity. In the absence of any other considerations, test 2 would be preferred in those situations where high sensitivity is preferred (for example, screening tests), whereas test 1 would be preferred in those situations where high specificity is preferred (for example, for ruling out a diagnosis for a risky surgical procedure).

2.1.3 Enhanced thresholding

There are many techniques for improving the ROC performance of threshold-based segmentation:

- *Seeding and region growing.* Simple thresholding makes no use of the geometric connectivity information present in an image. Using a seeding and region-growing technique, a segmented pixel set is created by initially selecting one or more pixels from the image (called the seed pixels). The seeds will usually correspond to some anatomical region or landmark and are often specified interactively by the user. The region-growing algorithm will then add to the segmented pixel set all the pixels that are *r-connected* to the initial seed pixels and also fall within the threshold limits. To be *r-connected* to one another, two pixels must share at least r corner points ($r \in \{0, 1, 2\}$). The algorithm recursively adds to the segmented pixel set all the pixels that are connected to the current members of the pixel set. The process stops when no more pixels can be added to the segmented set. This method tends to reduce the total number of false positives because only those pixels that can be r-connected to the original seed are classified. Isolated pockets of pixels that are not part of the object being classified but fall within the range are eliminated.

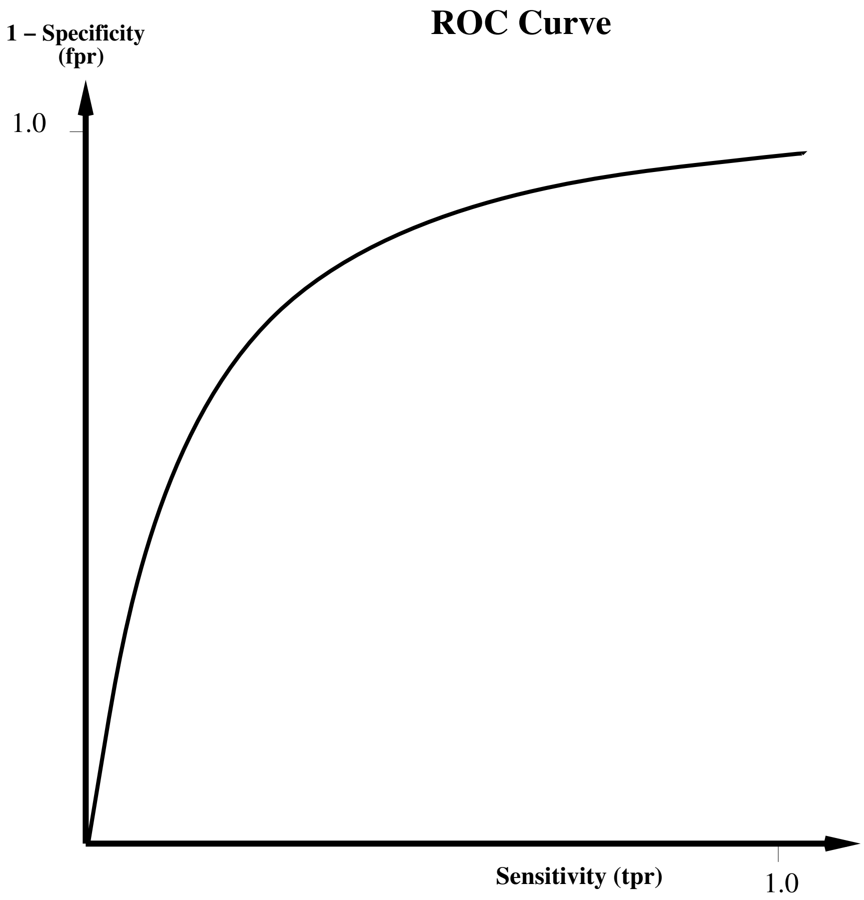


Figure 2.2: A Receiver Operator Characteristic (ROC) curve for the simple bone classification test constructed by varying the bone discriminating threshold. Points near the top right region of the curve correspond to lower bone threshold values, whereas points near the bottom left region of the curve correspond to higher bone threshold values. Note that the vertical axis represents $(1 - Specificity)$, which is also known as the false positive rate (fpr).

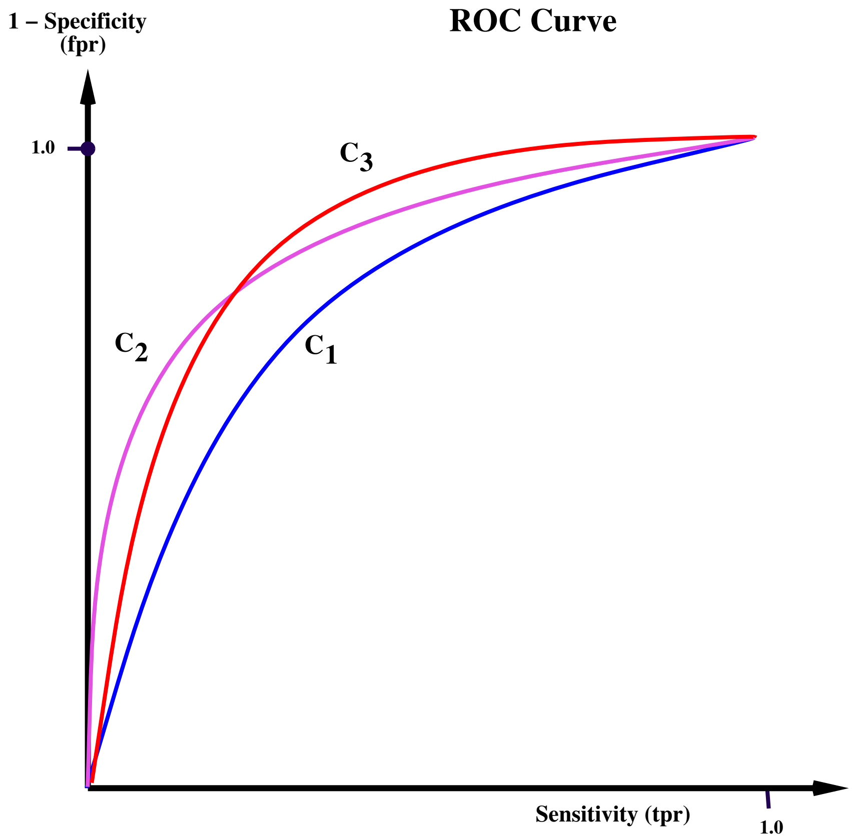


Figure 2.3: A family of ROC curves for a hypothetical set of classifiers. Test 1 (represented by C_1) is a better classifier than tests 2 and 3 because for all values of sensitivity, test 1 has a lower false positive rate. Test 2 is better than test 3 only at high values of sensitivity.

- *Hysteresis.* In this technique, a different threshold rule is used to grow a region than is used to select the seeds. Typically, the growing threshold is set to a lower value than the seed-selection threshold. This technique tends to produce segmented pixel sets with more clearly defined borders. Because the seed pixels have a higher threshold value, the seeds are less likely to correspond to false positive regions. This also tends to reduce the number of isolated pockets of false positive regions and produces sharper object borders.

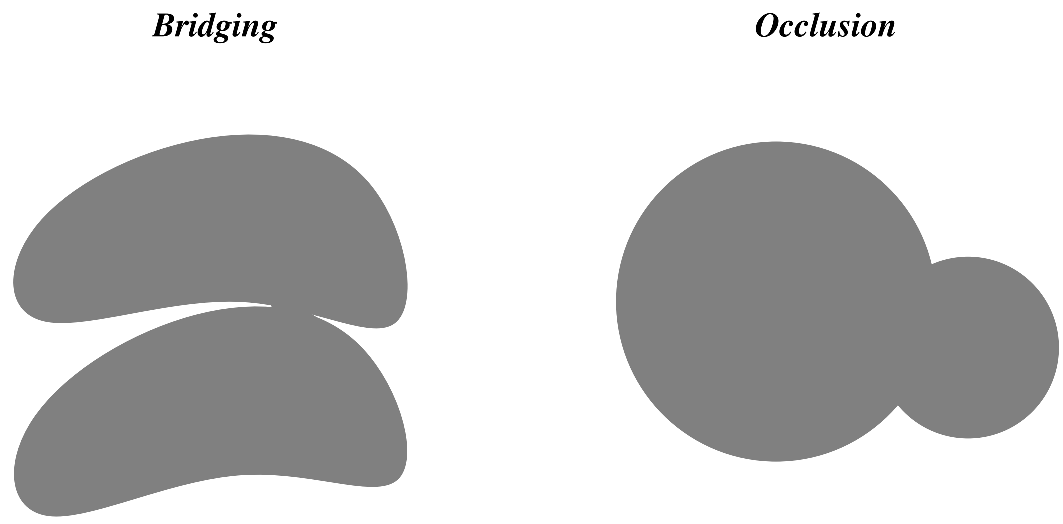
- *Subdivision.* Rather than use a single value throughout the image, the threshold can be made into a spatial function of the region of interests [18]. This implies that the region of interest can be viewed with a standard orientation and scale.

- *Pattern recognition and texture analysis.* Many tissues, such as bone marrow, have a regular pattern or texture that can be detected using techniques such as statistical clustering [29]. Often, regions can be grouped together according to image intensity values, and measurements regarding the distribution and concentration of various groups can be used to classify tissue types.

- *Spectral analysis.* This technique is useful when images can be acquired in different bands (for example, color or temperature components), with multiple sensors, or with multiple imaging parameters (for example, T1- and T2-weighted MR images) [60].

2.1.4 Assessment of region-based methods

Most region-based methods are simple, easy to implement, and execute fairly quickly. They tend to perform well in tasks such as area and volume calculations, and classification of easily distinguishable tissues. However, they do have certain drawbacks that can limit their usefulness. First, they provide no structural information about the regions they classify, because it is generally difficult to extract geometric information from a collections of pixels. Second, they can be confused by image anomalies such as false bridges between close, but separate regions, or by occlusions such as one region blocking the view of another region (see Figure 2.4). More sophisticated region-based methods include Fourier pattern analysis [77] and neural networks models [48]. In Fourier analysis, regions are decomposed into a regular grid pattern and an n-dimensional Fourier transform is applied to each subregion. The frequency and intensity distribution of transformed data can be compared with the expected distribution from training samples. Neural networks are similar to statistical image processing methods with the exception of the presence of hidden nodes that can regroup and transform the input data patterns. Although neural networks themselves suffer from many drawbacks, such as sensitivity to geometric scale, orientation, and intensity distribution, they have nonetheless been used with varying degrees of success in thousands of applications.



(a) **(b)**

Figure 2.4: Illustration of two potential limitations of region-based classification methods. (a) Due to the limited spatial resolution of CT and MR images, bridging errors can occur between distinct anatomical structures. (b) Occlusion errors occur when one region overlaps another region and partially blocks the perceived boundary of one or both of these regions.

2.1.5 Boundary-Based Segmentation

In order to overcome some of the limitations of region-based methods for classification and segmentation, boundary-based methods are often used to look for explicit or implicit boundaries between regions corresponding to different tissue types. The two most commonly used boundary based methods are known as *ridge detection* and *edge-detection*:

- Ridge detection follows the peaks (local maxima) in the original image. For example, one way to outline the shape of a crater in a two-dimensional surface plot is to "walk along" the crest of the crater's rim. The circular path that would be traced out would be a much more compact representation of the crater than say, the set of all points inside the crater.

- When objects don't have large enough ridges at the region boundaries for ridge detection, the *gradient operator* can be used to enhance the boundaries (i.e., edges) between distinct regions. Edge detection is identical to ridge detection except that peaks are tracked in the gradient space of the image instead of the original image space. The gradient space can be computed by simply applying the gradient operator to the entire image. The two-dimensional gradient-operator is defined as,

$$\nabla I(x,y) \triangleq \vec{\imath}\frac{\partial}{\partial x}I(x,y) + \vec{\jmath}\frac{\partial}{\partial y}I(x,y).$$

When it is necessary to distinguish the inside versus the outside of the edge, the Laplacian operator can be used. In two dimensions, the Laplacian operator is defined as,

$$L[I(x,y)] \triangleq \frac{\partial^2}{\partial x} I(x,y) + \frac{\partial^2}{\partial y} I(x,y).$$

The drawbacks of ridge-and edge-detection methods are that they can produce spurious, missing, or discontinuous edges. Hence, they can suffer from inadequate sensitivity and specificity because the image in the gradient space must be thresholded or otherwise classified according to edge or non-edge membership. Also, the problem of tracking an edge that bifurcates into two or more edges is one that cannot be adequated resolved using these low-level image operators alone. Furthermore, the derivative operator is inherently noisy and will exacerbate any noise already present in the image. Hence, the images must be smoothed before applying the gradient or Laplacian operators. Unfortunately, smoothing an image can hide or blur fine structures and other subtle features.

2.1.6 Summary of low-level operators

Although region and boundary techniques have been used for a wide variety of image understanding tasks, they rarely can be applied without a high-level agent to guide their operation. Consider for example, the generation of spurious boundaries in locations with steep gradients. These false boundaries can occur in locations where undersampling produces a stair-step or aliased image. Even if aliasing can be corrected, two- and three-dimensional segmenters are still susceptible to boundary tracking problems (boundary tracking is necessary in order to aggregate primitive elements, such as lines and surfaces, into complete anatomical structures or landmarks). Boundary tracking errors can occur at locations where edges intersect or where edges break off from one another. Another major limitation of these methods is that they fail to utilize or extract any shape information from the image objects (Section 2.2.1 will describe a technique for understanding shape information). In general, some sort of high-level interaction is required to (1) provide a context for the operators, (2) bound the search space, (3) disambiguate conflicting results, and (4) account for missing information. Section 2.3 will describe some approaches to deal with these limitations.

2.2 Shape Recognition

In order to utilize or to extract the shape information of objects in an image, a method for *representing* shape is needed. One such representation, known as the *medial axis*, was introduced by Blum to describe the shapes of various biological structures [10]. A recent refinement of the medial axis method, the *multi-scale medial axis* (MMA) technique, was proposed by Coggins and Pizer [20] [21]. Both methods represent the shape of an object by applying a transform that skeletonizes the object. In formal terms, the skeleton (or medial axis) is defined as the set of all points in an object that have at least two points

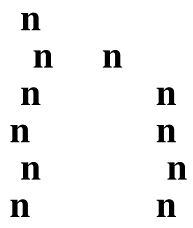


Figure 2.5: An example of object recognition at different scales. The number of "**n**"'s recognized in this figure depends on the resolution of the segmentation algorithm (see text).

that are equidistant to the object's boundary. The medial axis provides a compact way of representing shape because regions become reduced to curves that follow the general shape of an object. This is useful in applications such as character recognition, in which shape information is more important than area or volume information.

2.2.1 Multiscale medial axis

Coggins and Pizer were motivated by the desire to analyze images over a continuum of spatial scales. The classic approaches, which relied solely on edge and region information, were unable to handle many simple problems in image understanding. For example, how would you extract all the letter n's in Figure 2.5?

Although the correct answer is 12, a simple classification algorithm would report either 11 or 0, depending on its scale of operation. If the scale of the algorithm was the same as the scale of the smaller letter n's, it would probably report 11. If the algorithm was operating on the scale of the larger character, it would probably report zero because it would not find any edges or contiguous regions that resemble the letter n.

This led Coggins to the following conclusions:

- Edges can be misleading because they often contain too much noise and yield information of poor quality.
- Regions can be misleading because they often appear to be discontiguous.
- The human vision system seems to preserve features and their related ambiguities at various spatial resolutions; competing hypotheses about an image can be preserved and resolved at the level of cognition rather than at the level of perception.

As a result, Coggins reasoned that it is important to be able to interpret features on a continuous spatial-resolution scale.

2.2.2 Origin of scale

If the intensity values of an image are reinterpreted as a distribution of molecular concentration in a liquid or as a distribution of temperature in a solid, then techniques from the field of continuum mechanics can provide certain insights into the shape properties of objects in an image. For example, consider the two-dimensional, isotropic diffusion equation:

$$\frac{1}{k}\frac{\partial}{\partial t}I(x,y) = \frac{\partial^2}{\partial y^2}I(x,y) + \frac{\partial^2}{\partial y^2}I(x,y). \tag{2.3}$$

The temporal parameter, t, in equation 2.3 converts the problem into a dynamic system. The solution of the two-dimensional diffusion equation is

$$I(x,y,t) = K\int_I e^{-k(\frac{u^2+v^2}{\sigma^2})}I(u-x,v-x)dudv, \tag{2.4}$$

but equation 2.4 can also be viewed as the convolution of a Gaussian function with mean zero and variance σ^2,

$$I(x,y,t) = G(\sigma,x,y) * I(x,y). \tag{2.5}$$

Viewed as a convolution process, the time varying problem becomes converted into a purely static problem,

$$I(x,y,t) \Longrightarrow I(x,y,\sigma).$$

Other forms of the diffusion equation are also possible. For example, if the diffusion constant differs in the x and y directions, the material can be described by the anisotropic diffusion equation,

$$\frac{\partial}{\partial t}I(x,y) = D_x\frac{\partial^2}{\partial y^2}I(x,y) + D_y\frac{\partial^2}{\partial y^2}I(x,y). \tag{2.6}$$

If the diffusion constant varies depending on location, the material can be described by the amorphous diffusion equation

$$\frac{\partial}{\partial t}I(x,y) = D_x(x,y)\frac{\partial^2}{\partial y^2}I(x,y) + D_y(x,y)\frac{\partial^2}{\partial y^2}I(x,y). \tag{2.7}$$

The significance of the MMA is that it allows for a hierarchical approach to image understanding through the use of scale parameter σ. Using this hierarchical approach, the first step is to compute medial axes at largest scale of interest. Once the gross features of an object have been recognized, the smaller features can be found by reducing the scale parameter depending on the information obtained at the larger scale. This process is repeated until all the structures and features of interest have been recognized (see Figure 2.6). The hierarchical MMA approach is capable of recognizing large-scale features without sacrificing information contained at the finer resolutions. There are, however, several drawbacks to the MMA approach: (1) skeletal shape cannot recover the original size of the objects, (2) edge and boundary

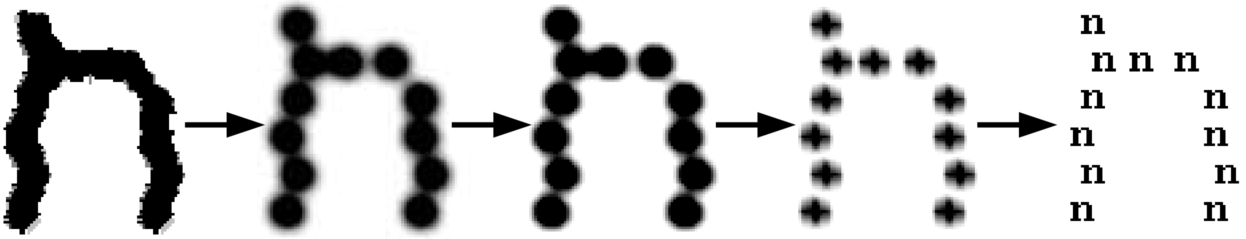


Figure 2.6: The multi-scale medial axis approach performs feature recognition by starting a the largest image scale of interest and then repeatedly reduces the scale until finer objects can be recognized.

information is lost, therefore the original shape is not uniquely represented, and (3) texture and pattern information is lost. Despite these drawbacks, the MMA can provide a structural road map of objects in an image. Although the MMA can't capture all the important features of an object, it can be used to provide some high-level information that provides the context for low-level image operators.

2.2.3 Limitations of non model-based vision

Moses et al, have constructed a theoretical argument for why non-model–based vision systems cannot correctly recognize objects in a consistent manner [58]. The authors offer a mathematical proof for their arguments based on a definition of *consistent recognition functions*. The details of this proof are beyond the scope of this dissertation. However, the basic result of the proof is that because different objects can produce similar looking images or image features, it is not possible to distinguish these objects without prior knowledge of how the images were formed. For example, consider the cylinder and the truncated cone in Figure 2.7. Both the top and bottom as well as the left and right sides of the cylinder are parallel in view (a), and the left and right sides of the cone are antiparallel. However, when viewed from orientation (b), the left and right sides of the cylinder have become antiparallel, making the cylinder indistinguishable from the cone. In order to recognize these two objects, a non-model–based vision system would have to be trained on all possible perspective transformations of these two objects. Furthermore, it would be difficult to take into account a known viewing orientation to improve the discrimination capability of the non-model–based system. On the other hand, a model-based system would not have to be trained on all possible view orientations. Furthermore, the model-based system could use the viewing orientation (either known or hypothesized) to discriminate between the two objects. For example, the ratio of the radii of the top and bottom disks spans a smaller interval for the cylinder than for the cone. Also, the degree of antiparallelism of the left and right sides is less pronounced for the cylinder than for the cone. Given the viewing orientation, the model-based system can predict *a priori* the degree of antiparallelism and the ratio of the radii that it will expect to see.

A less-contrived example of the limitations of non-model–based vision is the correction of blurred photographic images. Image blurring can be caused by camera motion, object motion, lens defocusing, chromatic separation, and image sensor diffusion. Without a model of the image formation process, it is difficult to reduce the degree of blurring. However, if we know the properties of the lens, or the velocity of the objects, or the physics of the image sensing device, we can optimally correct for the blurring. Of course, the correction will still be limited by factors such as the signal-to-noise ratio of the image sensors and the quality of the lens. Furthermore, some blurring processes such as intensity saturation and image bleeding are mathematically or physically irreversible. The point is that for any given set of constraints, a model-based vision system will have a better object discriminating capability than a non-model–based system. Indeed, many current research projects in computer vision deal with the construction of better geometric and computational models for recognizing objects and scenes.

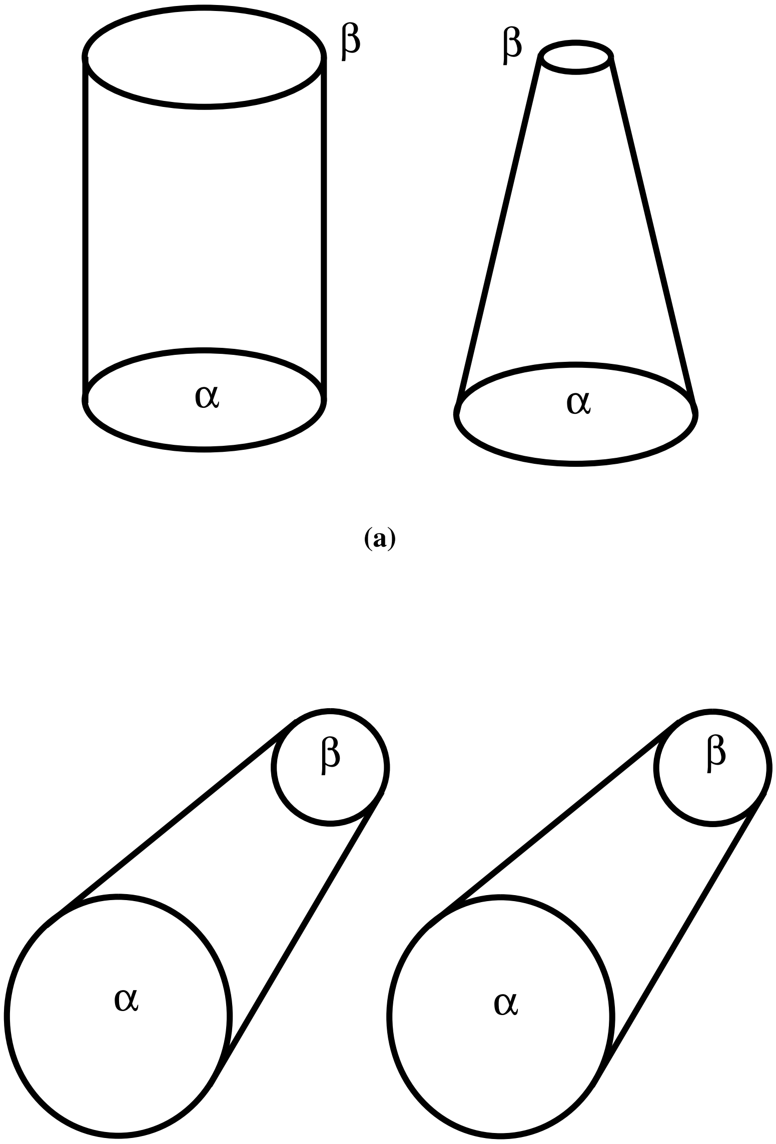
2.3 Model-based Approaches

All of the image analysis techniques reviewed so far are classified as low-level because they can operate directly on the data without any knowledge about the structure of the data or what the data represents (i.e., higher-level knowledge). This is not to say that low-level algorithms always *exclude* higher-level knowledge, but rather, that they can operate *with or without* this knowledge (with different levels of performance in each case). It is up to the implementer to decide (1) what types of high-level knowledge the low-level techniques will use and (2) how and when this knowledge will be used. In this section I will review several model-driven image analysis techniques. These techniques either use higher-level information in a fundamentally essential way or use an implicit representation of this higher-level information. I will describe three different model-driven techniques that have been applied successfully in three different application domains: (1) the active contour model, (2) the Kalman filter model, and (3) the Bayesian network model.

2.3.1 Snakes: an active model

Kass developed a novel technique for image segmentation, which was able to solve a large class of segmentation problems that had eluded more conventional techniques. He was interested in developing a model-based technique that could recognize familiar objects in the presence of noise and other ambiguities. Kass [47] proposed the concept of a *snake*, which was an active contour model using "an energy minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges." Snakes have been successful in performing tasks such as edge detection (both actual as well as subjective), corner detection, motion tracking, and stereo matching.

A spline is nothing more than a polynomial or set of polynomials used to describe or approximate curves and surfaces. Although the polynomials that make up the spline can be of arbitrary degree, the most commonly used are cubic polynomials. For example, a simple two-dimensional curve can be approximated by the following pair of cubic equations:



(b)

Figure 2.7: A cone and a cylinder (a) can have identical perspective transformations when viewed at a particular orientation (b).

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x, \quad y(u) = a_y u^3 + b_y u^2 + c_y u + d_y. \tag{2.8}$$

Higher-order polynomials can have undesirable non-local properties. Complex shapes can be decomposed into smaller regions having fewer inflection points by the introduction of appropriately placed *knots* (i.e., control points). In equation 2.8, the variable u is called the spline parameter and typically varies over the interval $[0,1]$. Because cubic polynomials can have a maximum of two inflection points, complex shapes need to be decomposed into simpler segments or patches having at most two inflections. Although the coefficients a, b, c, d uniquely determine the shape of the spline, they are not usually specified directly. Instead, they are computed from other constraints such as continuity of zeroth, first- and second-order derivatives at boundary points between neighboring polynomial segments.

Snakes fall into the category of *active contour models* because they dynamically alter their shape and position while trying to seek a minimal energy state. A two-dimensional dynamic contour called v can be defined in terms of its x and y coordinates, which in turn are parameterized by s, the linear parameter, and t, the time parameter,

$$v(s,t) = (x(s,t), y(s,t)),$$

where $s \in \omega$, usually defined as the closed interval [0,1], and $t \in T$, usually defined as the half-open interval [0,∞). The coefficients that minimize the energy of the spline can be found using either optimization techniques or differential calculus.

In Kass's original model, the total energy of a snake is made up of three subterms:

$$E_{total} = \frac{1}{2} \int_\omega [E_{internal} + E_{image} + E_{constraint}]$$

$E_{internal}$ is the internal energy of the spline and depends solely on the shape of the spline. E_{image} is the image energy and depends solely on the image intensity values along the path of the spline. $E_{constraint}$ is the constraint energy and is created by artificial energy fields imposed by the user or the high-level control agent. Other energy terms can be defined, but for the purpose of this review section, I will limit the discussion to these three terms.

The internal energy is defined as

$$E_{internal} = \alpha \left| \frac{\partial v}{\partial s} \right|^2 + \beta \left| \frac{\partial^2 v}{\partial s^2} \right|,$$

where α controls the amount of stretching the snake is willing to undergo and β controls the amount of flexing it will allow. Large values of α will increase the internal energy of the snake as it stretches more and more, whereas small values of α will make the energy function insensitive to the amount of stretch. Similarly, large values of β will increase the internal energy of the snake as it develops more curves, whereas small values of β will make the energy function insensitive to curves in the snake. Smaller values of both α and β will place fewer constraints on the size and shape of the snake.

The image energy of the snake is defined as

$$E_image = w_1 I(x,y) + w_2 \left|\nabla I(x,y)\right|^2 + \cdots,$$

where w_1 is called the line coefficient and w_2 is called the edge coefficient. Large positive values of w_1 tend to make the snake align itself with dark regions in the image, $I(x,y)$, whereas large negative values of w_1 tend to make the snake align itself with bright regions in the image. Small absolute values of w_1 make the snake more indifferent to intensity variations in the image. Similarly, large positive values of w_2 tend to make the snake align itself with sharp edges in the image whereas large negative values of w_2 make the snake avoid these edges. Small absolute values of w_2 make the snake indifferent to edges in the image.

The external constraint energy is defined as

$$E_{external} = -\sum_i k_i(x_{1i} - x_{2i}) + \sum_j max(peak_j, \frac{1}{r_j}),$$

where the k_i terms are external spring factors and the r_i terms are called volcano factors. A large value of k_i makes the snake behave as if there was a powerful spring connected between a point on the image at (x_{1i}) and a point on the snake (x_{2i}). The larger the value of k_i, the more powerful the force of the spring. The r_i terms are called volcanoes because the plot of $max(peak_j, \frac{i}{r_j})$ resembles the profile of a symmetric volcano. Computationally, these volcano terms act as a repulsion force between a point on the image at a distance r_j from a point on the snake; the larger the value of $peak_j$, the stronger the force of repulsion. As a result, springs and volcanoes are a way of capturing high-level knowledge about images and the features contained within these images.

Using the internal and image energy forces, a snake will find desired image features in an autonomous fashion. The spring forces can be set up interactively by the user to confine the region in which a snake will operate. The volcano forces can be set up by the user to define regions that the snake should avoid.

Snakes have several advantages over classical feature extraction techniques:

- They can be controlled interactively by using appropriately placed springs and volcanoes.
- They are easy to manipulate because the external image forces behave in an intuitive manner.
- They are autonomous and self-adapting in their search for a minimal energy state.
- They can be made sensitive to image scale by incorporating Gaussian smoothing in the image energy function.
- They are relatively insensitive to noise and other ambiguities in the images because the integral operator is an inherent noise filter.
- They can be used to track dynamic objects in temporal as well as the spatial dimensions.

Snakes are not without their drawbacks, chief among which are the following:

- They can often get stuck in local minima states; this may be overcome by using simulated annealing techniques at the expense of longer computation times.
- They often overlook minute features in the process of minimizing the energy over the entire path of their contours.
- Their accuracy is governed by the convergence criteria used in the energy minimization technique; higher accuracies require tighter convergence criteria and hence, longer computation times.

In summary, snakes are a model-driven approach for solving many image understanding problems that are difficult, if not impossible, to tackle using classical approaches. Just like human vision, snakes start with an *a priori* model of what an object should look like. By using the smoothness constraints of the splines, they are able to fill in for missing and noisy boundary information. As a result, they are more robust than non-model based methods, which make little use of image structure. Much of the current research in active contour models deals with generalizing the form of the contours and overcoming the convergence and stability problems encountered during the energy minimization process [6] [25] [52] [53] [57].

2.3.2 The Kalman-filter model

Often times we would like to estimate the state of a system given a set of measurements taken over an interval of time. The *state* of the system refers to a set of variables that describe the inherent properties of the system at a *specific instant of time*. **Kalman filtering** is a useful technique for estimating, or updating the previous estimate of, a system's state by (1) using indirect measurements of the state variables and (2) using the covariance information of both the state variables and the indirect measurements. Intuitively, the idea is to use information about how measurements of a particular aspect of a system are correlated to the actual state of the system.

To illustrate this concept, suppose we know that the average height and weight for males in the U.S. is 178 cm and 71 kg, respectively. That means, that in the absence of any other information, a randomly selected group of males will most likely have a mean height of 178 cm and a mean weight of 71 kg. But suppose we are told that the height of one of the males is 150 cm. How would the estimate of his weight change? Suppose we also know that the length of his forearm is 25 cm. Would that provide an even better estimate of his weight? The answer is "yes" if we know how the various measurements are correlated to one another. In human anatomy, we can exploit the correlations among various anthropometric quantities to make estimates using limited data. Kalman filtering is one such technique for using the correlation information to derive better estimates of unknown quantities.

The linear dynamic model

The Kalman filter was orginally developed to estimate time-varying quantities (state variables) from a set of noisy measurements (signals). The set of state variables is usually represented by a vector denoted as $\mathbf{x}(t)$. For example, we could define the state of a satellite to be its position in Cartesian coordinates relative to the center of the earth. In this case, the state vector would look like the following:

$$\mathbf{z}(t) = \begin{bmatrix} x & y & z \end{bmatrix}^T, \tag{2.9}$$

where the (xyz) triplet represents the time-varying coordinates of the satellite's position (note the distinction between $\mathbf{x}$, the state *vector*, and x, the *x-component* of the state vector). [1] In general, it is not possible to determine through direct measurements the precise state of such a system. Instead, we must rely on measurements that provide information about the state variables. In this example, we may have to rely on external radar signals or telemetry signals sent by the satellite itself. In other words, we must infer the state of the system based on the set of measurements we have about the system. In an ideal world, the state of the system can be determined precisely given an appropriate set of measurements. In practice, this is difficult because these measurements are corrupted by noise. In this example, if we had an onboard instrument that could measure the satellite's (x,y,z) coordinates in Cartesian using a gyroscopic system, the relationship between the state vector $\mathbf{x}$ and the measurement vector $\mathbf{z}$ can be written as

$$\mathbf{z}(t) = H\mathbf{x} + \mathbf{v}, \tag{2.10}$$

where $\mathbf{z}$ is the measured position vector and $\mathbf{v}$ is the noise vector associated with the measurement. The matrix H is referred to as the *transfer* function because it transfers the input, which is the actual state, to the output, which is the set of measurements. Equation 2.10 expresses the measurement vector as a function of the state vector. What we really need is the state vector expressed as a function of the measurement vector. If we know the value of $\mathbf{v}$, this would reduce to a simple matrix inversion ($\mathbf{x} = H^{-1}(z - v)$). However, the addition of the *unknown* $\mathbf{v}$ term implies that we can no longer determine $\mathbf{z}$ precisely. Instead, we must find an *estimate* of $\mathbf{z}$ based on our knowledge of how the state variables are correlated to one another, and on the prior distribution of $\mathbf{v}$. A Kalman filter is a function that provides an *optimal* estimate of the state vector $\mathbf{x}$ given such knowledge. This is an example of a *linear* dynamic system because the input and output are described by a linear equation.

The non-linear dynamic model

Suppose the position of the satellite was measured by a ground-based tracking station, which outputs the position in spherical coordinates relative to the station's tracking antenna (spherical coordinates are specified by a distance ρ from an origin and two angles of rotation θ and ϕ). This would be an example of a *non-linear* dynamic system because the satellite's state vector (which is represented in Cartesian coordinates) is no longer a linear function of the measurement vector. This relationship can now be expressed as

[1] Because of the signal processing origins of the Kalman filter, the confusion between $\mathbf{X}$ and x is somewhat unavoidable when used in a geometric context. Given the pervasive use of $\mathbf{X}$ as a state vector in the literature, the reader will have to keep the distinction clear throughout the remainder of this text.

$$\mathbf{z}(t) = H(\mathbf{x}) + \mathbf{v}, \tag{2.11}$$

where

$$H(\mathbf{x}) = \begin{bmatrix} \arccos z/\sqrt{x^2+y^2+z^2} \\ \arctan y/x \\ \sqrt{x^2+y^2+z^2} \end{bmatrix}. \tag{2.12}$$

H is now a general vector-valued transfer function of $\mathbf{x}$. In both the linear and non-linear cases, the noise is assumed to be a property of the *measurement* process and not of the system's state vector.

The linear static model

Although the Kalman filter was designed in the early 1960s to estimate the state of *dynamic* or *time-varying* systems, it can also be used on *static* systems as well. Altman [2] has used this technique to determine the structure of certain protein molecules using geometric information provided by nuclear magnetic resonance (NMR) studies. In this application, the inputs consist of geometric measurements such as distances and angles among various atoms (along with the variances of these measurements), and the output of the filter is a vector that lists all the coordinates of the atoms within the molecule.

Altman and Brinkley have also used a linear Kalman filter to extract the boundaries of the kidney and the spleen from cross-sectional CT images [3]. In this case, the output state vector (which represents the boundary points of the organ) is updated by boundary measurements that are introduced in a point-wise sequential fashion. As more measurement points are introduced, the state vector estimate converges to the true boundary. Hence, the Kalman filter provides a *Bayesian* updating mechanism for sequentially refining the estimate of the overall boundary vector. [2] In other words, it is a model-driven image segmentation technique. In both cases, Altman is using the Kalman filter as a computational mechanism for solving a geometric contraint-satisfaction problem. In the protein structure problem, the constraints are the measured distances and angles among atoms and the problem is to find the optimal geometric configuration of these atoms that satisfies all the constraints. In the organ segmentation problem, the constraints are interactively supplied boundary points.

To illustrate how the Kalman filter can be used to solve a geometric-constraint problem, I will discuss in greater detail the kidney segmentation technique developed by Altman and Brinkley. In this technique, the kidney cross-section is represented by a two-dimensional **radial contour model (RCM)** in which the organ boundary points are specified in polar coordinates relative to a predefined kidney center [15] [16]. These points are sampled at uniform angular intervals of 15 degrees (although in theory, non-uniform angular intervals will work just as well). An RCM with uniform angular displacements can be thought of as a deformable bicycle wheel whose outer rim can be arbitrarily stretched or compressed to match

[2] Although some authors use the term *Bayesian* to refer to techniques that use Baye's rule to calculate the probabilities of discrete events, the definition I use here refers to the subjective notion of probability as opposed to the frequency-based notion.

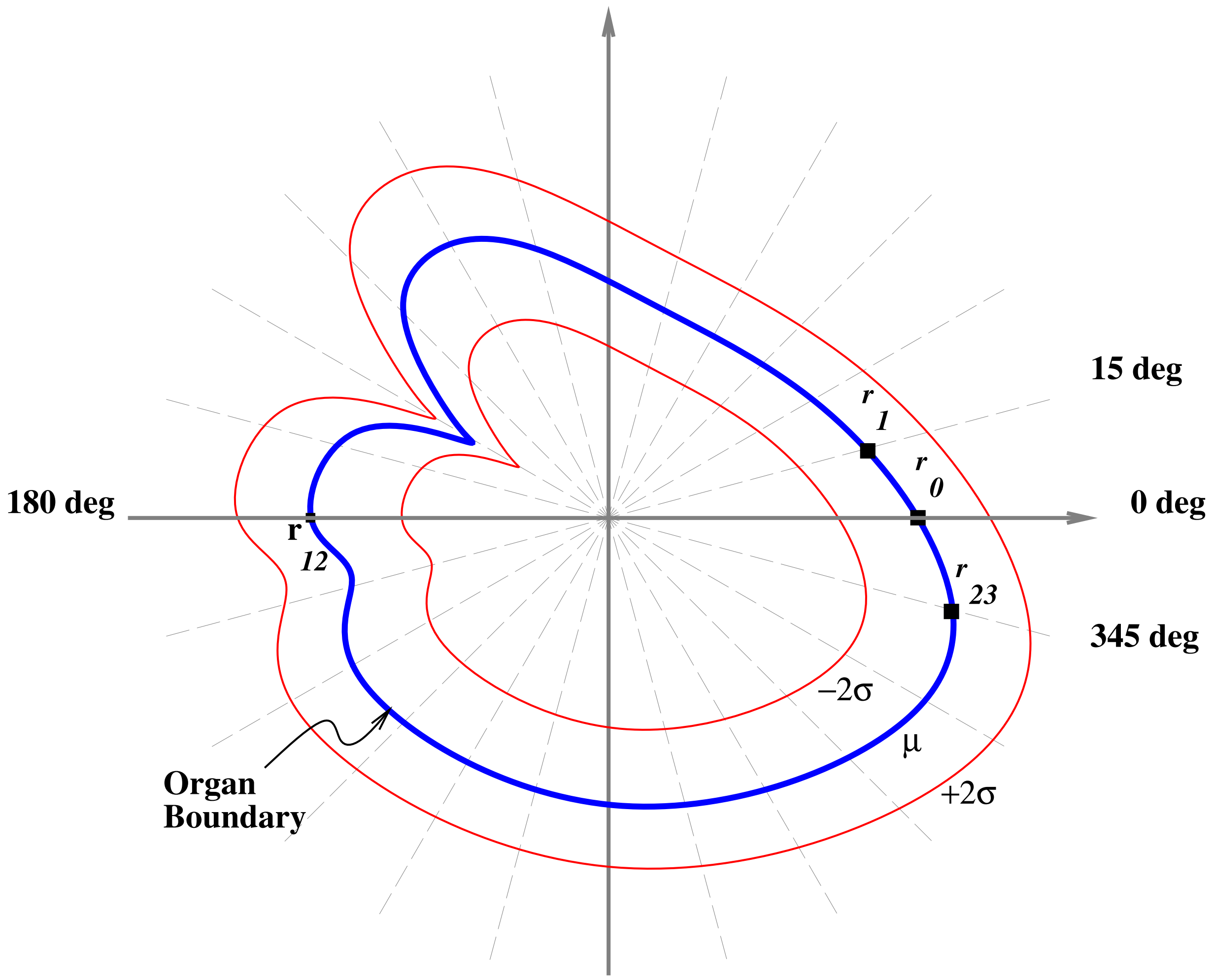


Figure 2.8: A representation of a kidney cross section using the Radial Contour Model (RCM).

the boundary of the object it is representing (see Figure 2.8). Here, the RCM is initialized to look like an average kidney cross section (i.e., the spokes are initialized to the mean boundary displacements computed over a set of training images for each spoke). Starting from this initial RCM (which also serves as a generic kidney model), the Kalman filter incrementally refines *all* the displacements as boundary points are introduced in a sequential fashion.

The state vector is made up of the 24 radial displacement values at each angular value ranging from 0 to 345 degrees:

$$\mathbf{x} = \left[\begin{array}{cccccc} r_0 & r_1 & r_2 & \cdots & r_{22} & r_{23} \end{array} \right]^T . \tag{2.13}$$

The measurement vector is a subset of the state vector in which only the displacements that are measured are present. For example, if the measurement vector $\mathbf{z}$ consists of two components, the first at zero degrees and the second at 180 degrees, the measurement model would look like the following:

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \left[\begin{array}{c|c|c|c} \begin{matrix} 1 \\ 0 \end{matrix} & \varnothing & \begin{matrix} 0 \\ 1 \end{matrix} & \varnothing \end{array} \right] \mathbf{x} + v, \tag{2.14}$$

where the two non-zero columns represent the contributions from r_0 and r_{12}. Because the measurement vector is a subsample of the state vector, the matrix H will consist entirely of zeros and ones. The equation for generating a new estimate of the state vector (denoted as $\mathbf{x}_{new}$), based on the previous state estimate (denoted as $\mathbf{x}_{old}$) and a new measurement $\mathbf{z}$, was derived by Kalman as the following [46]:

$$\mathbf{x}_{new} = \mathbf{x}_{old} + K\{\mathbf{z} - H\mathbf{x}_{old}\}. \tag{2.15}$$

Equation 2.15 states that the magnitude of the update is a weighted difference between the predicted measurement $H\mathbf{x}_{old}$ and the actual measurement $\mathbf{z}$. The weighting matrix K is the Kalman filter that produces an optimal estimate of the new state and is defined as follows:

$$K = C_{old}H^T\{HC_{old}H^T + v\}^{-1}. \tag{2.16}$$

The matrix C_{old} is the state covariance matrix and is also updated after each measurement according to the following equation:

$$C_{new} = C_{old} - KHC_{old}. \tag{2.17}$$

This covariance matrix C captures the correlations among all the state variables and permits the Kalman filter to make estimates about the system's state using a relatively small set of measurements. Indeed, before any measurements are made, the value of C_{old} is initialized to the covariance matrix computed on a set of training data:

$$C_{initial} = \begin{bmatrix} \sigma^2_{r_0} & \sigma_{r_0 r_1} & \cdots & \sigma_{r_0 r_{23}} \\ \sigma_{r_1 r_0} & \sigma^2_{r_1} & \cdots & \sigma_{r_1 r_{23}} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{r_{23} r_0} & \sigma_{r_{23} r_1} & \cdots & \sigma^2_{r_{23}} \end{bmatrix}, \tag{2.18}$$

where the variance and covariance of the elements are defined in terms of the expected value of their cross-products:

$$\sigma_{r_i r_j} = E[r_i r_j] - E[r_i]E[r_j], \tag{2.19}$$

and

$$\sigma^2_{r_i} = \sigma_{r_i r_i}. \tag{2.20}$$

The diagonal elements of C represent the variances of each of the 24 radial displacements from the set of training data. [3] Equivalently, the value of $\mathbf{x}$ is initialized to the mean displacements computed from the training data:

$$\mathbf{x}_{initial} = \left[\begin{array}{cccc} E[r_1] & E[r_2] & \cdots & E[r_{23}] \end{array} \right]^T, \tag{2.21}$$

The mean state vector $\mathbf{x}_{initial}$ can be thought of as the *a priori* or *generic* model of the organ in the absence of any measurements from an organ. Figure 2.8 illustrates how the mean and the variances of the generic model can be used to define a region of high probability for finding the organ boundary. This region is determined by bracketing the elements of the mean state vector (labeled μ) by $+/-2\sigma_j$ (note that σ_j is unique for each of the 24 radial displacement elements). As radial displacement measurements are made sequentially on an organ to be segmented, the state vector and the $+/-\sigma$ brackets converge toward the true boundary of the organ.

Altman and Brinkley verified the performance of their technique by demonstrating that it satisfied the following three properties:

1. The generic model of the organ along with its $+/-\sigma$ brackets adequately describes the variations seen in actual test cases. This was verified by comparing 15 test kidneys against the generic model (which was generated using a different set of training data). The authors found that all of the test cases had radial displacements that were within two standard deviations from the generic model.

2. As more measurements are introduced, the estimated state vector $\mathbf{x}$ converges toward the actual organ boundary. This was verified by computing an RMS error difference between the estimated $\mathbf{x}$ and the actual boundary. The authors noted that not only was the error reduced for those radial elements that corresponded to actual measurements, but the error was also reduced for radial elements that were not measured. In other words, providing a single new measurement reduces the error of each of the elements. This effect is due to the information provided by the covariance matrix.

3. As more measurements are introduced, the values of $+/-\sigma_j$ become smaller and smaller. Although this property may seem intuitive, it is not readily apparent from equation 2.17. The authors verified this property by inspecting the values along the diagonal of C after each update.

As as result of their experiments with kidney cross sections, the authors concluded that the linear Kalman filter is capable of estimating the complete boundaries of kidneys from CT images using a limited set of boundary measurements. Furthermore, they demonstrated that the covariance matrix and the mean state vector can model both the geometry of the kidney and spleen cross sections as well as the variability seen in these organs. Hence, the Kalman filter technique can provide a *statistical* framework for modeling and extracting the boundaries of kidney sections. This statistical framework could behave more robustly certain in situations where deterministic algorithms fail.

[3]The authors used a training set consisting of 15 kidney cross sections.

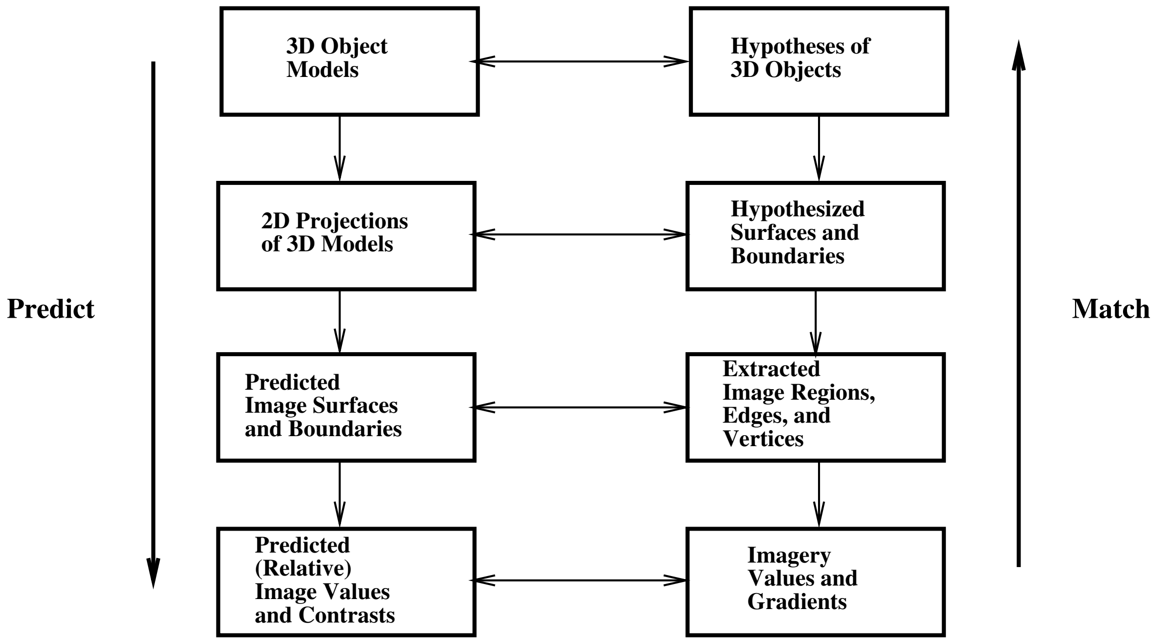


Figure 2.9: The conceptual diagram of the hierarchical image interpretation model developed by Binford.

2.3.3 Model-based vision

Binford has presented a report on the status of research in computer-based vision and image understanding currently under way at the Stanford Robotics Laboratory [8]. In particular, he focuses on a system he has developed called SUCCESSOR, which is designed to interpret images from multiple input sensors (including stereo images) and perform segmentation, enhancement, object identification, and visualization. I will focus more on his efforts in model-based image segmentation in this review.

Binford proposed a dual hierarchical image understanding technique in which image values, structures, and shape descriptors are combined to form hypotheses about objects to be recognized [7]. At each level of this hierarchical process, information from the image is matched against a prediction model (see Figure 2.9 for a conceptual diagram that illustrates this approach).

Starting from the top left side of the diagram, the process is driven by a three-dimensional model of the objects to be detected and their geometric relationships to one another. Using an estimated viewing perspective of the imaging device, the first step is to predict which objects are most likely to be visible, and project those objects onto the imaging planes based on the known geometric characteristics of the image sensors. The knowledge of the imaging modality is then used to predict the contrasts in the image based the material properties of the objects. The predicted contrasts are used to match the predicted edges, boundaries, regions, and textures against those found in the actual images using various low-level operators. When a reasonable (but possibly incomplete) match is found at the lowest level, the process can proceed to extract and match higher-level features (shown on the right side of the diagram). The regions, edges, and vertices extracted from the image operators can now be reliably combined into surfaces and boundaries that can then be combined into the three-dimensional objects to be detected. The advantage of the dual hierarchical approach is that it reduces the search space of hypotheses regarding how low-level image features are combined into higher-level abstractions. The prediction hierarchy (left side) prunes the search space so that the extraction hierarchy (right side) can perform more refined image processing operations. As a result, Binford's technique potentially can provide more accurate image interpretations than would be possible using a single-sided extraction hierarchy.

One of the basic principles behind Binford's approach to segmentation is to utilize as much information as possible about the geometries of the objects being modeled and the spatial relationships among these objects. Using this information, objects can be recognized more accurately from a wide variety of input sources. Furthermore, he claims that this recognition technique is more robust in the presence of noise and other ambiguities than existing algorithms that rely solely upon imaging properties. An added benefit of his model-based segmentation approach is that it can incorporate knowledge about the viewing parameters of the images and relate that knowledge to the geometric representations of the objects. This enables his system to utilize a rich set of models for understanding images and recognizing the objects contained within. These models have successfully been used to segment medical images, aerial photographs, indoor scenes for mobile robots, and recognition of industrial parts.

Binford also proposed models based on computational solid geometry (CSG) techniques and representations of straight homogeneous generalized cylinders (SHGC) and curved surfaces of revolution (CSR). These models can represent a wide variety of objects and are especially suited for objects that exhibit some degree of symmetry (e.g., spinal segments and disks). Once a primary object axis has been identified, this modeling technique can represent a wide variety of topologies by sweeping out specific functions of the form f(r,theta) along a straight, three-dimensional path, or by sweeping a constant function along a curved three-dimensional path (see Figure 2.10). These geometrical representations can be combined with basic set operations (such as intersection and union) to construct more complicated composite objects. Directed acyclical graphs (DAGs) are used to represent hierarchical relationships among the various subobjects that form these composite objects. This representation has also been extended to incorporate Bayesian models of uncertainty to perform evidential reasoning under uncertainty.

In the SUCCESSOR system, once the objects to be modeled have been parameterized and represented using DAGs, image segmentation can be viewed as a process for fitting the contours of the modeled object to data points in the image. Several approaches can be used to perform the fitting. In one approach, an implicit equation can be derived to directly fit the predicted contour with the observed data. Another approach is to construct a wire-frame model of the composite object, apply an error bound to the frame, and direct the search for contours in the image data. The wire-frame model can be adjusted to account for object-to-object variation as well as variations in the viewing parameters (e.g., position, scale, and orientation). Regrettably though, Binford provides no quantitative results of any of the image understanding projects. Instead, he refers interested readers to more detailed papers written by various

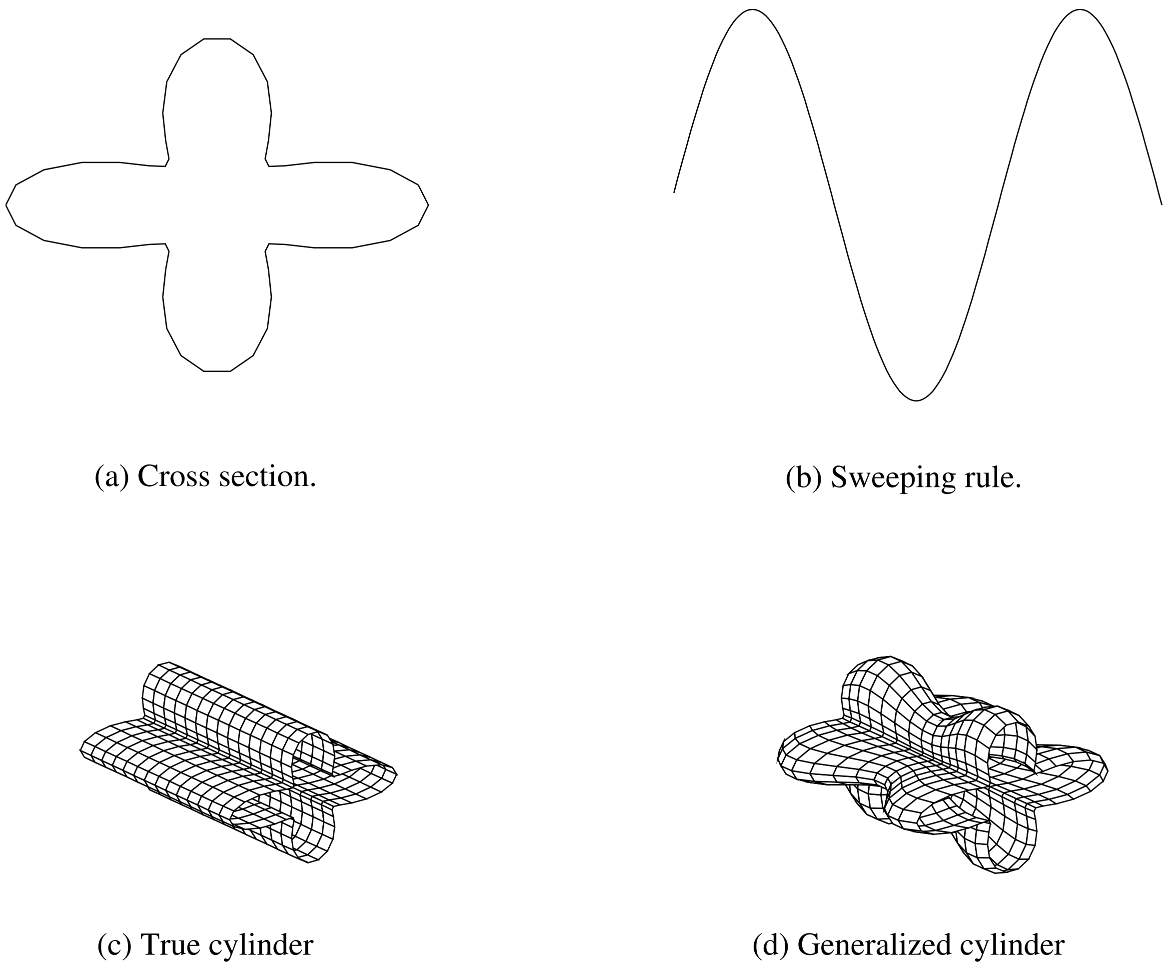


Figure 2.10: A straight homogeneous generalized cylinder (SHGC) is defined by two functions. The first function defines the cross-section (a) of the cylinder. The second function is the sweeping rule (b), which modulates the width of the cylinder (c) through the transverse axis to create the final SHGC (d).

graduate students in the lab.

2.3.4 Computational vision

Ponce and Kriegman have essentially extended the SUCCESSOR model to include algebraic surface patches and their intersecting curves [67]. They describe a new approach for recognizing curved three-dimensional objects from two-dimensional image contours derived from edge maps of gray-scale information. They claim that their technique is powerful enough to handle most objects created by existing CAD tools. Their basic their approach is to construct a closed-form expression for the error estimate between the predicted contour and the observed data using a technique known as elimination theory. They do not explain the details of elimination theory but merely state that it is a technique for removing independent variables from a set of parametric equations. This allows complex equations to be reparameterized into a consistent set of independent variables (in their case, three rotation angles and three displacement variables). Ponce and Kriegman have successfully used this approach to fit image contours to models of curved three-dimensional objects using the SUCCESSOR system. They have achieved surface recognition accuracies of 1.5 pixels for toroidal shaped objects containing many bumps and irregularities.

While the exact mathematical details in their paper are very skeletal, they basically use the following expression to represent the theoretical contours of each object:

$$F(x, y, dx, dy, dz, a, b, c) = 0, \tag{2.22}$$

where x and y represent the position of the object on the image; dx, dy, and dz represent the displacement of the object from the viewing origin and a, b, and c represent the orientation angles of the object. With this equation, they use the Levenberg-Marquardt algorithm to solve the non-linear least-squares minimization of the error function (in this case, the sum of F^2 over all the contour data points). A more accurate method is to define the error function as the distance between the image data points and the theoretical points. They argue that under the assumption that the distance between the actual and theoretical data points is relatively small, the error function can be approximated as

$$F(x + nx, y + ny, dx, dy, dz, a, b, c) = 0, \tag{2.23}$$

where nx and ny are the scaled normals of the contours at each data point. This equation can be solved using a simple Newton-Raphson method. Using this latter technique, they were able to detect the position and orientation of various toroidal objects and differentiate these objects based on various properties such as surface smoothness, color, and geometric aspect ratio.

The advantage of their system is that it is one of the few working systems that can recognize simple three-dimensional objects independent of the view angles and the presence of occluding objects and features. However, their system has many signficant drawbacks as well. For instance, each object model must be reparameterized into polynomial expressions involving the six variables described earlier. While they have done this for toroidal objects, they admit that this step is difficult to generalize and may require a great deal of CSG analysis to extend to other objects. Furthermore, they only work with two-dimensional edge maps (i.e., two-dimensional images representing only the edges of a gray-scale image). Since edge maps are susceptible to shadow effects from occluding surfaces, all the shadows had to be removed manually. In general, while their technique is very accurate, they require very "clean" images, which means a certain amount of manual preprocessing must be performed on each image. Both of these limitations make their approach unfeasible for use in general medical image processing. Most systems in the medical domain require robust, automated algorithms to deal with large studies involving dozens of slices of axial images per study. Furthermore, medical images contain very complicted geometries for which no adequate models have been developed (let alone subjected to the rigors of elimination theory).

Despite these drawbacks, Ponce and Kriegman have taken an important first step in recognizing simple three-dimensional objects from two-dimensional contour data. For future research, they propose to extend their elimination technique to include so-called superquadratics used in computer graphics. They would also like to couple the image segmentation and object recognition steps in order to eliminate the error-prone manual preprocessing of the two-dimensional gray-scale data. As a result, their object recognition approach shows great promise for becoming a useful image understanding tool in the near future.

2.3.5 The Bayesian model

The active contour model described in Section 2.3.1 is an example of a model-driven segmentation technique. The Kalman filter technique described in Section 2.3.2 adds the notion of statistical variation to the model-driven segmentation process. In this section, I will introduce the **Bayesian network** (BN) approach to segmentation. Bayesian networks are also referred to as *belief networks*, *probabilistic networks*, *probabilistic belief networks* (PBN), and *probabilistic causal networks*. Since all of these terms can be used interchangeably, I will refer to them as *belief networks* throughout this dissertation. Regardless of the name, this approach adds the dual notions of *probability-* and *utility-based* decision making to the repertoire of segmentation techniques. The belief network model makes decisions about how to interpret *probabilistic evidence* (i.e., non-deterministic information) to support or reject a *hypothesis* or *outcome*. Outcomes that yield the highest *expected utility* are chosen as the optimal solutions. With the belief network model, the definition of expected utility incorporates all the probabilistic uncertainty associated with the outcome, as well as the inherent utility of the outcome. Utility can be defined in dimensions such as monetary cost, entropy, or energy. [4]

Many researchers have been using belief networks as a convenient mechanism for managing uncertainty in expert systems. Most of these expert systems to date have dealt with the tasks of classification or diagnosis in a temporally static problem domain. To a lesser extent, researchers have also been able

[4]Some authors argue that *all* utility can be converted into monetary units [43, 44]. While this may be true in theory, in practice it is often simpler to deal directly with the units that are most appropriate for the problem at hand, rather than devise a scheme for converting utility to monetary units.

to model dynamic properties using belief networks in an effort to simulate and predict the behavior of time-varying systems.

The idea of incorporating belief networks into this model-driven computer vision system was first proposed by Levitt and Binford [8]. The motivation for using belief networks was to make image interpretation insensitive to variations in structure, viewpoint, sensor type, shading, illumination, and obscuration [8]. Furthermore, real world image interpretation requires the ability to handle millions of features in a statistically efficient manner, using an accurate and rigorous mathematical model of uncertainty. According to Binford, relating image features such as step, delta, and slope discontinuities (i.e., boundaries and edges) to image structures and object models is a difficult problem.

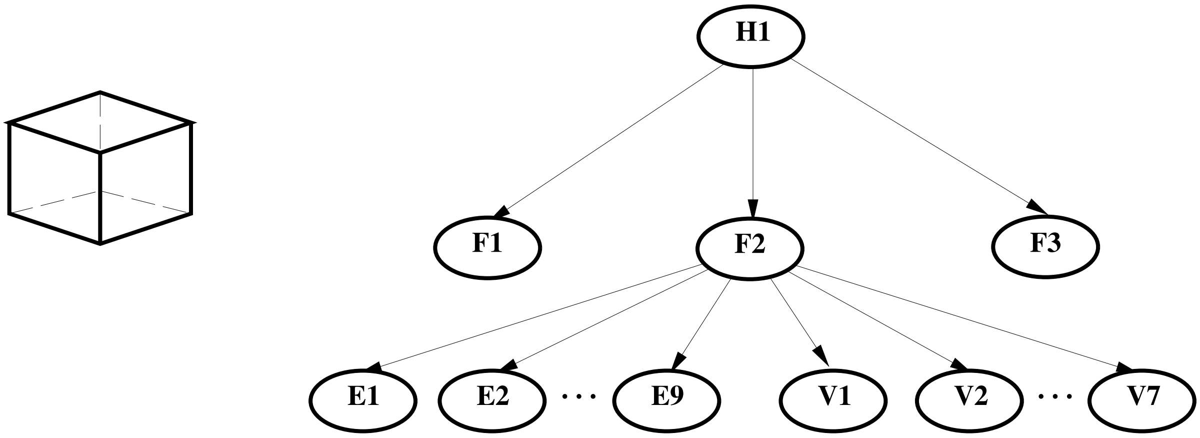
Levitt and Binford argue that the belief network fits naturally into the hierarchical image understanding model, where objects are composed of parts and joints. Joints specify the relationships among parts, which in turn are composed of subparts and joints. This recursive relationship can be expressed in a directed acyclic graph (see Section 3.5.2 for a more detailed description of directed graphs). Bayesian inference is used to accrue *evidence* (i.e., observations) about the image in a mathematically coherent framework. In this manner, a sufficient set of probabilistic evidence, even if it is incomplete or ambiguous, can be amalgamated to support or deny *hypotheses* about the objects in the image. Competing hypotheses can be rank ordered by their overall probability, or likelihood of occurrence.

To briefly illustrate how a belief network can be used in model-based vision, suppose we have two three-dimensional objects that we would like to recognize from a photographic image: (a) a solid cube and (b) a solid pyramid. The two DAGs shown in Figure 2.11 represent the belief networks for these two objects. Let H_1 represent the hypothesis "object is a cube," and H_2 represent the hypothesis "object is a pyramid."

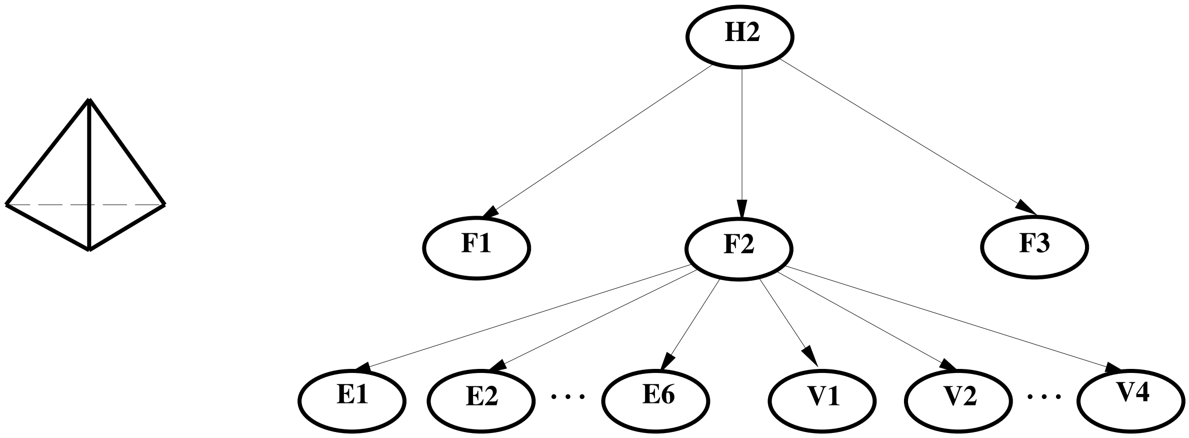
Given a photograph of a cube, we would expect to see at most three faces, nine edges and seven vertices. For a three-sided pyramid, we would expect to see at most three faces, six edges, and four vertices. The DAG for the cube shows that H_1 gives rise to three faces (F_1, F_2, F_3). In turn, each face gives rise to four edges (E_1 through E_4) and four vertices (V_1 through V_4). By performing statistical experiments, we can determine the probability of seeing the faces under various lighting and viewing conditions. That is to say, we can determine $P(F|H)$ the probability of detecting a face, given the type of object. Likewise, we can collect statistics on the probabilities of seeing the edges and vertices $P(E|F)$ and $P(V|F)$ given that we have seen a face. These statistics are called the *observed* probabilities because they are based on what we can observe from the image. What we would really like to know is the *inferred* probability, that is, the probability that the face was created a certain object. By using Baye's rule (also referred to as Kolmogorov's theorem by statisticians), we can compute the inferred probabilities. For example, the probability that a face was created by a cube can be computed as

$$P(H_1|F) = \frac{P(F|H_1)P(H)}{P(F)} = \frac{P(F|H_1)P(H)}{P(F|H_1) + P(F|\neg H_1)}. \tag{2.24}$$

The inferred probability is now expressed in terms of the observed probabilities $P(H_1)$, $P(F|H_1)$, and



(a) A three−dimension cube and its corresponding DAG.



(a) A three−dimension pyramid and its corresponding DAG.

Figure 2.11: A simple belief network representation of two objects as directed acyclic graphs (DAGs). The cube hypothesis (a) is represented as an object with three visible faces, nine visible edges, and seven visible vertices. The pyramid hypothesis (b) is represented as an object with at most three visible faces, six visible edges, and four visible vertices.

$P(F|\neg H_1)$ (where $\neg H_1$ means "not" H_1, that is, the probability of seeing a face given that the object is a pyramid). H_1 is also called the *prior probability* of the cube hypothesis. In other words, it is the overall probability that the object is a cube in the absence of any other information. Statistically speaking, $P(H_1)$ is simply the ratio of the cube population to the total population of objects.

In a similar manner, we can compute the probability that an edge was created by a face,

$$P(F|E) = \frac{P(E|F)P(F)}{P(E)} = \frac{P(E|F)P(F)}{P(E|F)+P(E|\neg F)} = \frac{P(E|F)(P(F|H_1)+P(F|\neg H_1))}{P(E|F)+P(E|\neg F)}. \quad (2.25)$$

Strictly speaking, the probability of observing a face given an edge will vary depending on the number of edges that were detected. That is,

$$P(E_1|F) \neq P(E_1 and E_2|F) \neq P(E_1 and E_2 and E_3|F). \quad (2.26)$$

However, to reduce the complexity of the computations and the size of the statistical data gathering task, the assumption of *conditional independence* can be introduced. This assumption states that the probability of seeing one edge does not increase the probability of seeing another edge. As a result, the subscripts on the edges can be dropped,

$$P(E|F) = P(E_1|F) = P(E_1 and E_2|F) = P(E_1 and E_2 and E_3|F) = \cdots \quad (2.27)$$

Although it might be tempting to do so, it is not possible to compute the probability of a hypothesis given an edge observation, $P(H_1|E)$, by simply multiplying $P(H|F)$ by $P(F|H)$. The computation of $P(H_1|E)$ is a non-trivial problem and was first solved by Pearl (interested readers should consult [65] for a rigorous derivation). There are many commercial software packages that will compute inferred probabilities for belief networks using observed statistical data.

In addition to observations about the presence of edges and vertices, Binford and Levitt have proposed using relationships such as parallelism, connectivity, and angular displacements as evidence in the belief network. Furthermore, they have also proposed parameterizing the observed probabilities as a function of the viewing orientation. In this case, the observed probabilities would be represented by probability distributions rather than by scalar constants.[5] In general, the computation of such probability distributions requires quasi-invariant transformations to map random variables from the measurement domain to the computation domain.

Several prototype systems have been built using the model-based belief networks. In section 2.3.6 I will describe a system for segmenting two dimensional radiographic images. In section 2.3.7 I will describe a complete vision system for interpreting monocular greyscale images based on this technique.

[5] The authors have collected statistical data for these probability distributions for various geometric objects.

2.3.6 Image segmentation using belief networks

To illustrate how belief networks can be used to segment medical images, I will describe a prototype system developed by Levitt, Hedgcock et al, to automatically analyze symptoms of arthridities in hand radiographs [51]. This system was developed at the San Francisco VA Medical Center with the goal of diagnosing and tracking radiological studies involving arthridities. It demonstrates the feasibility of combining Binford's model-based image understanding approach with probabilistic inference techniques. There are three major processing activities in this system: (1) prediction of image features based on three-dimensional anatomical models of the hand (parameterized by patient population statistics), (2) extraction of local image features from regions with a high probability of pathology, and (3) accumulation of evidence from lower-level image processing and feature matching procedures to support or deny hypotheses regarding anatomical structures (the likelihood of each hypothesis is computed using a Bayesian calculus).

The authors claim that their system accurately segments "normal" and "somewhat degenerated hand anatomy." Furthermore, the system has the ability to "soft fail," that is, to recognize when it has inaccurately segmented an image set. This feature allows clinicians to take into account the accuracy of the automated results when making a diagnosis. The authors are currently investigating the quantification of rheumatoid arthritis, osteoarthritis, and other degenerative joint diseases using their system.

The Bayesian reasoning technique collects *evidence* to support or deny *hypotheses* about the image being analyzed. In this case, evidence refers to features such as edges, curves, vertices, and regions that are extracted by low-level image processing operators. Also, properties of features such as size, intensity variations, straightness, parallelism, convexity, and closure are also measured. The prior probability distributions of various features are derived from the sensor-anatomy physics and are combined with the estimated viewing orientation to generate the predicted image values and contrasts (see Figure 2.9).

Hypotheses are generated by matching the evidence extracted from the images with geometric models of the hand. These three-dimensional geometric models were built using published population statistics on the lengths, widths, and relative sizes of the phalanges (bones inside the digits) and metacarpals (bones inside the palm). The generated hypotheses provide possible interpretations about the size, location, and orientations of these anatomical surfaces and boundaries. The process is repeated as newly generated hypotheses are used to update the generic three-dimensional model, which is then used to predict more features.

The system's control loop (shown in Figure 2.12) chooses the processing actions that will rapidly converge toward a highly probable interpretation of the image anatomy (that is, a greedy algorithm based on highest expected probability). As each hypothesis is generated, a Bayesian network is created and the collected evidence is used to update the probabilities in this network. The accumulation and computation of the network probabilities is performed by an external software package called Quanta, which is a commercial implementation of Schacter's belief network algorithm [71].

Because of the concentric layering of bone and soft tissues in the phalanges and metacarpals, the geometric model of the hand was built using non-deformable straight homogeneous generalized cylinders.

```
Initialize Bayesian Hand Network
   Run Hand-Finder
   If (Probability(Hand) < 0.5)
      Then Return ( Error: "Not a hand radiograph" )
      Else Set PossibleProcesses = sub-part-actions of
           hand-node

Loop Until (Number of PossibleProcesses = 0)
   Set NextProcess =  Highest-Utility-Process(PossibleProcesses)
   Run NextProcess
   Update Bayesian Network with NextProcess result
        For each (New Node in Bayesian Network)
   Set PossibleProcesses =
      Union(PossibleProcesses, sub-part-actions of node)
End Loop
```

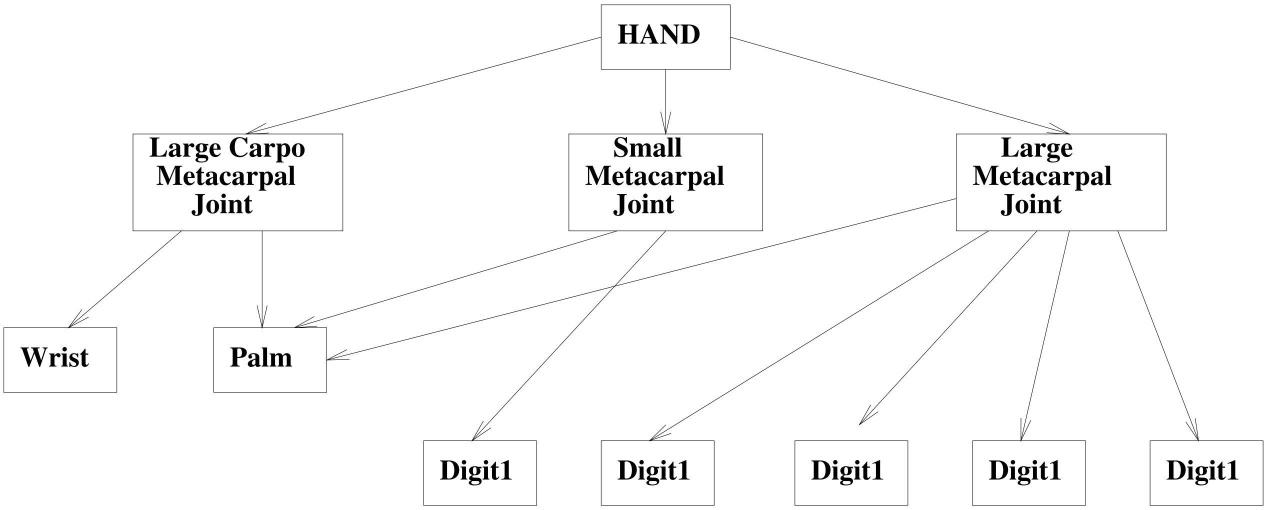
Figure 2.12: Top-level control loop of Levitt's hand radiograph analyzer.

At the top level, the hand is modeled as a collection of digits, metacarpal joints, and the palm (see Figure 2.13(b)). The digits, in turn, are modeled as collections of phalanges and joints (see Figure 2.13(b)).

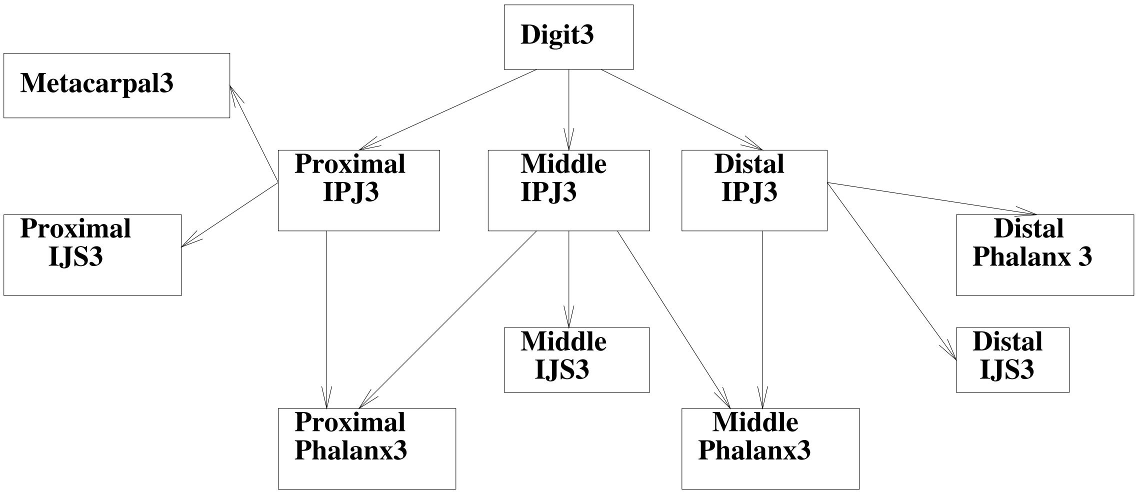
In total, there are 58 modeled anatomical parts, each with its corresponding Bayesian network. As each new piece of evidence is obtained, a new node is attached to the appropriate networks and the belief values are updated. For example, when a phalanx segmentation operation is performed, a new evidence node is created and is given a probability based on the prior distribution of P(phalanx | segmentation evidence value). By propagating the probability of the evidence node through the Bayesian network, the probability of each active hypothesis can be recomputed.

To summarize, the geometric and statistical model is used to predict surfaces and boundaries in the two-dimensional image projections. The initial model is determined from prior statistical distributions as well as from initial features extracted by low-level algorithms. The predicted structures are used to direct the search for more features on a finer scale. As new features are extracted and more surfaces and boundaries are matched, the probability distributions of each hypothesis are evaluated using a Bayesian calculus. This cycle is repeated until a hypothesis with a high probability is generated, or until the system determines that no hypothesis can be confirmed.

Levitt's system cannot segment phalanges that are subluxed (a common condition in advanced cases of arthridites whereby the soft tissues separating the articulating bones have loosened or degenerated). However, because the system evaluates the accuracy of its segmentation process, it will reject anomalies such as subluxed joints, rather than provide incorrect measurements. One of the signficant contributions of this Bayesian model-based approach is that it can evaluate hypotheses based on ambiguous, incomplete, or obscured evidence. Furthermore, the Bayesian method combines prior probability distributions from population statistics with evidence from image processing operations using a mathematically sound



(a)



(b)

Figure 2.13: The Levitt model of the hand consists of two hierarchical levels. The first level (a) expresses the relationships between the digits and the metacarpals. The second level (b) expresses the relationships among the phalanges, the inter-phalangeal joints (IPJ), and the inter-joint spaces (IJS). The arrows indicate a *consists-of* relationship.

decision theoretic approach. Unfortunately, the authors have deferred, for various reasons, any formal clinical evaluations of the system's performance, so the validity of their approach has yet to be established.

2.3.7 Three-dimensional Bayesian analysis

A system for performing recognition of three-dimensional objects from photographic images using belief networks has been demonstrated by Mann [54]. Mann has augmented the SUCCESSOR system by adding the capability to create belief networks and to solve them using a software package called HUGIN [5]. This package accepts a belief network as input and generates probabilities of various hypotheses in the network. To demonstrate the feasibility of his approach, Mann has developed a network to recognize the shape and pose of pipe elbows in a greyscale image. Just as in Levitt's system, Mann dynamically creates a belief network using low-level features from an image and groups these features into higher-level objects. In his model, the pipe elbow is made up of several subparts: the female cylinder, the male cylinder, and the spherical joint. Each of these subparts gives rise to certain two-dimensional aggregate features. For instance, the cylindrical parts give rise to parallel ellipses on both ends that are joined by parallel line segments. The aggregated features themselves can be decomposed into various edges and shaded regions. Finally, Mann can reconstruct a model and create a two-dimensional greyscale image of the original pipe elbow.

2.4 Summary

Image understanding is a very diverse and active field of research. Even an exhaustive review of subtopics such as two-dimensional segmentation could fill several volumes. Hence, there are many different approaches one could use to solve any given image understanding task. Even after trading off complexity, generality, cost, and a host of other factors, there would still be many approaches worth pursuing. In this chapter, I have described several successful approaches used to solve various problems dealing with segmentation, shape representation, and image understanding. I have shown how computational operators such as thresholding, edge detection, and ridge detection are used extensively in low-level image processing tasks. Snakes and active contour models have the potential for handling more difficult image understanding tasks that require a greater degree of autonomy. Still, higher-level models such as those proposed by Binford and Levitt are required to perform complex three-dimensional object recognition. In summary, effective image understanding requires tight coupling between high-level, hierarchical, task-dependent models and low-level computational operators. The low-level operators are responsible for generating context-dependent data under the guidance of the high-level models. The high-level agents can synthesize the low-level data in a structured manner to make decisions about how to track boundaries, how to extract regions, how to aggregrate information, and to how determine when an object has been recognized correctly.

Chapter 3

Methodology

To understand the design specifications and the trade-offs made in the development of the vertebral-extraction algorithm, it is helpful first to review the techniques used to perform fusion of spinal images that are currently under development by the SPIF project team. Although the work presented in this chapter is directly motivated by the needs of the SPIF project, the principles developed here can be generalized to cover other medical imaging applications as well.

3.1 Fusion Procedure

Figure 3.1 shows a schematic overview of the current system for performing fusion of *cranial* images (although this system is not highly automated, it forms the basis for the SPIF system). CT and MR images of the head are first thresholded to reveal the outline of the bony cortex at each slice. A two-dimensional boundary-tracking operation is then performed to extract the outlines of the skull at each slice. The user can edit the boundaries found by the tracking algorithm. A surface polygon approximation is computed from the resulting contours at each slice. The optimal transform between the surface points derived from the CT images to the polygons derived from the MR images is computed using a Levenberg-Marquardt optimization technique. Using the resulting transform, the CT image points are finally mapped onto the MR image and then displayed.

The major drawback of the cranial fusion system is that a great deal of manual editing of the skull boundaries is required before the registration step can be performed. For spinal images, the bony structures are even more difficult to extract (particularly in MR). In addition, a unique transformation matrix must be computed for *each* vertebra. Unfortunately, individual vertebra are difficult to separate from one another near the facet joints and disk boundaries. Because of these difficulties, I have decided to use a generic finite-element model to extract the vertebrae from CT images. In this capacity, the finite-element model behaves as an *active surface model*, which adapts itself to the boundaries of a given patient's vertebrae by minimizing a predetermined global energy function (the mathematical interpretration of this

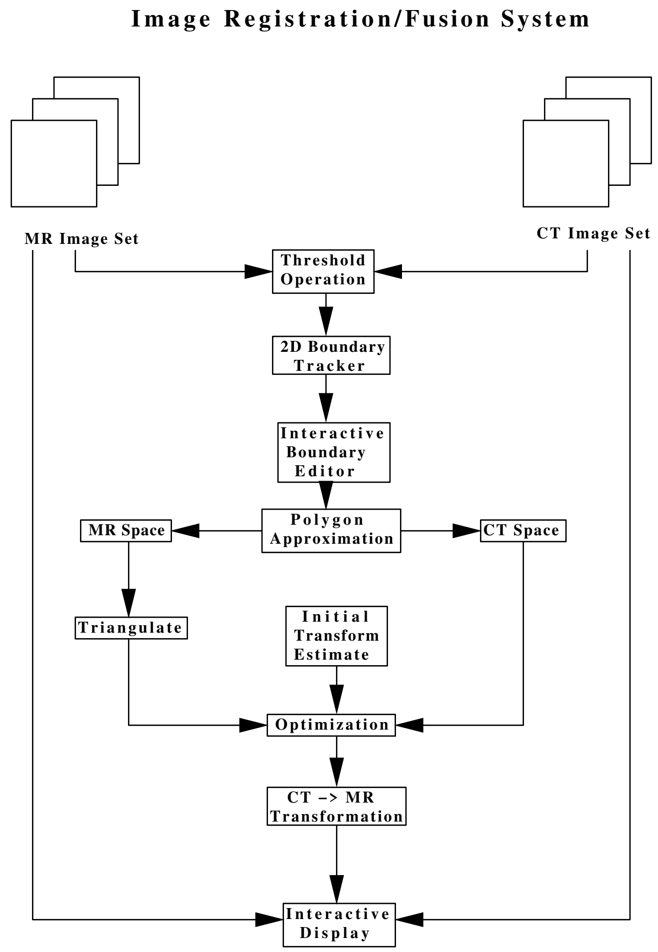


Figure 3.1: Schematic overview of the current system for fusion of cranial images. *(Diagram courtesy of Paul Hemler)*

energy function is simply the total error of the solution). Once the vertebral surfaces have been extracted, they can be mapped onto the MR image space by using the appropriate transformation matrix for each vertebra. The transformation matrix can be computed by using matching landmarks in the CT and MR images, or by extracting the vertebral surfaces from the MR images using the same finite-element model derived from the CT extraction step.

The steps for fusing spinal images can be summarized as follows:

1. Obtain high-quality CT and MR images of the patient's spine (preferably during the same scanning session).

2. Identify the cervical regions of interest on both data sets.

3. Choose a distinctive set of landmarks that can be readily extracted in an automated fashion. Landmarks can consist of points, curved segments, or surface patches.

4. Locate and extract matching landmarks from both CT and MR images using either a manual or an automated method.

5. Perform a coarse match of the generic spine model to the patient data.

6. Locate and extract from CT images the vertebral segments to be fused using a generic finite-element model. The generic model conforms to the image data to create a patient-specific model.

7. Find the optimal rotation and translation parameters that minimize distances between matching landmarks, or match the patient-specific model obtained from the CT images to the MR images.

8. Apply the optimal transformation parameters to map CT bone pixels onto the MR image space (in either two or three dimensions).

This dissertation only deals with the vertebral-extraction steps in the previous list (steps 3, 4, 5, and 6). Figure 3.2 shows an overview of the steps involved in the vertebral-extraction process. This technique will accept as input a volumetric CT data set of a patient's cervical spine. It will produce an *finite-element* model (see Section 3.3) that represents the shape, position, and orientation of the patient's cervical vertebrae. There are three major steps in this process: (1) extraction of predetermined landmarks in the patient data set to determine the affine transformation matrix that performs coarse match of the generic model to the data, (2) local deformation of the generic model by refining the surface elements of the model to precisely match the surface boundaries of the patient's vertebrae, and (3) improvement of the *match score* using a non-linear optimization subroutine (NPSOL). The optimization subroutine will repeatedly adjust the affine transformation matrix until the match score computed in step 2 is maximized. The match score is computed during the local deformation stage and reflects the degree to which the refined generic model fits the patient data (see Section 3.6 for a precise definition of the score). After each score is computed, a

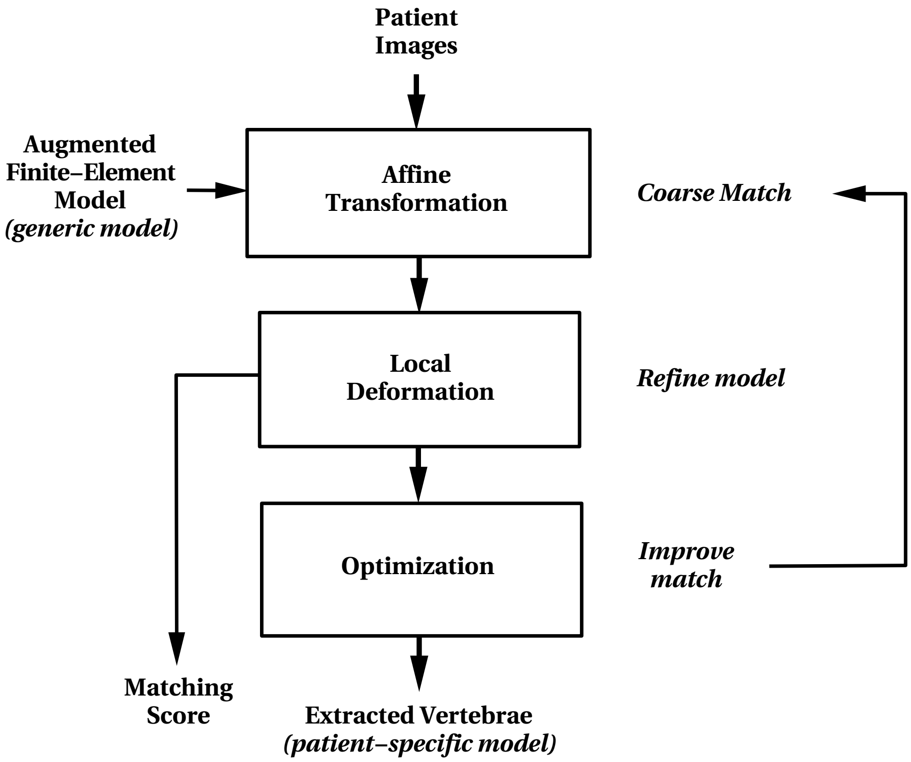


Figure 3.2: Extraction of vertebral segments consists of three steps: (1) coarse matching of the finite-element model using an affine transformation, (2) local deformation of the model to match individual patient variations, and (3) improvement of match score using an optimization subroutine.

new global transformation matrix is computed and steps 1 through 3 are repeated until the score can no longer be improved.

Before the landmarks can be located, the cervical region of interest must be specified. It is crucial to know which vertebrae need to be extracted and roughly where they occur in the images. Once the spinal region of interest has been specified, the search for landmarks can begin. The landmarks are used to refine the positions, orientations, and sizes of each of the vertebral elements. I have found four strategic point landmarks (see Section 3.2.3) that can be quickly and reliably determined to be sufficient for performing a crude match of the generic model to the input data. These landmarks will provide information on how to set up the initial *local* fit for each individual vertebral segment. The locations of the landmarks will also determine the affine transformation matrix needed to register the CT data with the MR (provided that landmarks have been successfully determined in both data sets). Although these landmarks currently are determined manually using an image editing program, they were chosen because of their simple geometric characteristics (they can all be described by conic sections) and should be straightforward to determine using automated methods if desired. The landmarks do not need to be determined precisely because the optimization loop will improve upon the initial transformation matrix that is computed from the landmarks. Because the landmarks can be quickly determined and because the vertebral-extraction algorithm is relatively insensitive to landmark errors less than $\pm$ 20 mm, the need for automated landmark detection is not critical.

3.2 Model Matching

Fitting a generic model to the patient data can be performed in either two or three dimensions. The two-dimensional model-fitting method matches a sequence of edges, contours, or two-dimensional regions from the model with their corresponding features in the patient data. The match should be performed on a per-slice basis and the global cost function should be a weighted combination of the cost functions computed on the individual slices. The three-dimensional model-fitting method matches surfaces, curves, or three-dimensional regions from the model with their corresponding features in the patient data. The match should be performed using three-dimensional operators over the entire patient data volume.

There are trade-offs involved in the choice of two- versus three-dimensional model fitting. Two-dimensional fitting is computationally less expensive and requires less memory than three-dimensional fitting. A typical vertebral segment can be represented by 30 to 40 contiguous image slices. If we use a contour-matching method, there will be roughly 40 separate generic contour models to be matched. However, if we take into account the tilt along the vertical axis of the spine, then the number of generic contours is multiplied by the number of individual tilt angles allowed. If we allow for a tilt of $\pm$ 20 degrees with an increment of 1 degree, then the number of generic contours could reach as high as 1600. Furthermore, each vertebral segment differs somewhat from the next. An accurate model would also have to take into account any significant variations in the contours across different segments. In practice, a significant fraction of these contours will be similar enough in appearance to be merged into a single contour. Due to the large number of initial contours, merging these homologous contours will require some sort of automated comparison technique.

As an alternative to contour matching, a region-based matching approach can provide more accurate results because it can take into account the variation in image intensity values in addition to the contour information. However, the memory and computation time increase significantly because of the increased number of image data points to be compared. Furthermore, any region-based technique that does not use a rigid-body assumption would require a warping transformation over the entire generic slice model. The warping transformation could add a large number of additional parameters to each of the generic region models.

A better two-dimensional approach would be to start with a three-dimensional generic model and extract contours from planar slices at various tilt angles and slice locations. The extracted contours would then be used to perform a two-dimensional match with the patient image slice data. This approach would preserve the geometric constraint information because the slice model would be derived from three-dimensional operators applied to the three-dimensional model.

Performing a match of a three-dimensional model using two-dimensional cross sections can yield a fit that is significantly different from the fit obtained by matching in three dimensions. For example, suppose we wish to match an object O in $\Re^3$ by decomposing it into a series of N objects O_n in $\Re^2$. We can represent the decomposition as

$$O = \bigcup_{n}^{N} O_n. \tag{3.1}$$

We can define a two-dimensional error measure f_2 for each slice D_n, and a global weighting function g that combines all of these error terms. We wish to minimize the error e_2,

$$e_2 = g(\sum_{n} f_2(O_n, D_n)), \tag{3.2}$$

where D_n represents the two-dimensional image data at slice n. On the other hand, to perform the match in three dimensions, we first define a three-dimensional error measure f_3 which we apply to the entire three-dimensional image data D. We then seek to minimize,

$$e_3 = f_3(O, D). \tag{3.3}$$

Because there is no general relationship between f_2 and f_3, there is no guarantee that the resulting fits will even approximate one another. In fact, if the positions and orientations of the individual D_n elements are free of constraints with respect to one another, then we can expect

$$min(e_2) <= min(e_3). \tag{3.4}$$

To illustrate the problem, suppose we have a model of an object in $\Re^2$ (see Figure 3.3a). We can decompose this model into the three components shown in Figure 3.3b. If we are given the image shown in Figure 3.3c, then the elements labeled (1) and (2) in Figure 3.3b would both match region (1) in Figure 3.3c. In the absence of geometric constraints (such as non-overlapping elements), the best fit of the model to the data would be depicted in Figure 3.3d. This problem can be avoided by constraining the elements so that they cannot overlap one another. Although this problem may not occur in practice, two-dimensional matching techniques in general must rely on some sort of constraint mechanism to compensate for the loss of three-dimensional structural information. Note that the aforementioned problem would not occur if the subregions in Figure 3.3c were matched against the model's subobjects (instead of the converse). Unfortunately, this approach would require us to know *a priori* the locations, sizes, and orientations of the different subregions that need to be matched. This, of course, is the problem we are trying to solve.

Although it is feasible to match three-dimensional objects in $\Re^2$ in the interest of reducing complexity as well as computational costs, there are significant trade-offs involved. The two-dimensional submodels should be carefully selected in order to reduce the number of matches that need to be performed. Such data-reduction steps affect the generality of the matching technique. A three-dimensional model implicitly preserves the geometric constraints among its submodels provided the deformation process is well behaved.

3.2.1 Error metric

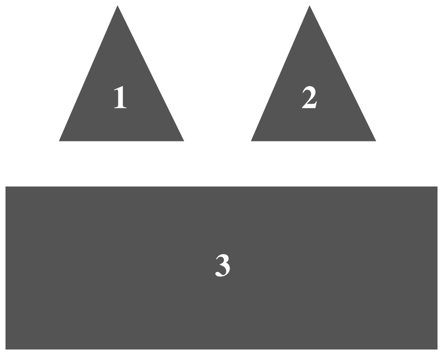
The task of fitting the generic model to the patient data requires defining an appropriate error metric, and then finding a configuration of vertex positions that minimizes this error. The error can be further minimized by adding new vertices at locations with high error contributions. Therefore, the best match of the generic model to the data is dependent on the error metric used to measure the goodness of fit. I have defined an error metric called the *total tetrahedral error*, which is the sum of the volumes of the tetrahedra formed by each triangular surface element and the intersection of its perpendicular medial axis with the inferred surface of the object to be segmented (see Figure 3.4).

If the vertices of the tetrahedron are defined as the set (A, B, C, S), as indicated in Figure 3.4, then the volume can be computed using the formula

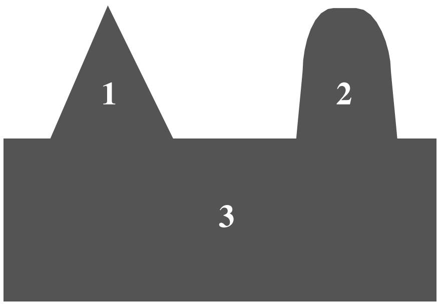
$$\text{Volume} = \frac{1}{6} \times \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ s_x & s_y & s_z & 1 \end{vmatrix}, \tag{3.5}$$

where (a_x, a_y, a_z) are the x, y, z coordinates of vertex A, etc. Equation 3.5 is referred to as the *signed volume* of a tetrahedron because the quantity can be either positive or negative depending on the ordering and orientation of the four vertices. For the purposes of model fitting, the sign of the volume is irrelevant and the absolute value will be used instead. The exception occurs when interior volumes must have a

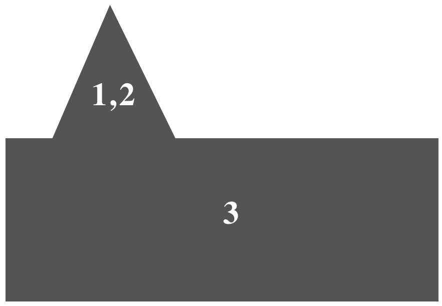
(a) Generic model of object *O*.



(b) Decomposition of *O*.



(c) Image data D.



(d) Resulting match.

Figure 3.3: Decomposing the object model O, shown in (a), into its components, shown in (b), can yield unexpected results (d) when a match to the data shown in (c) is performed.

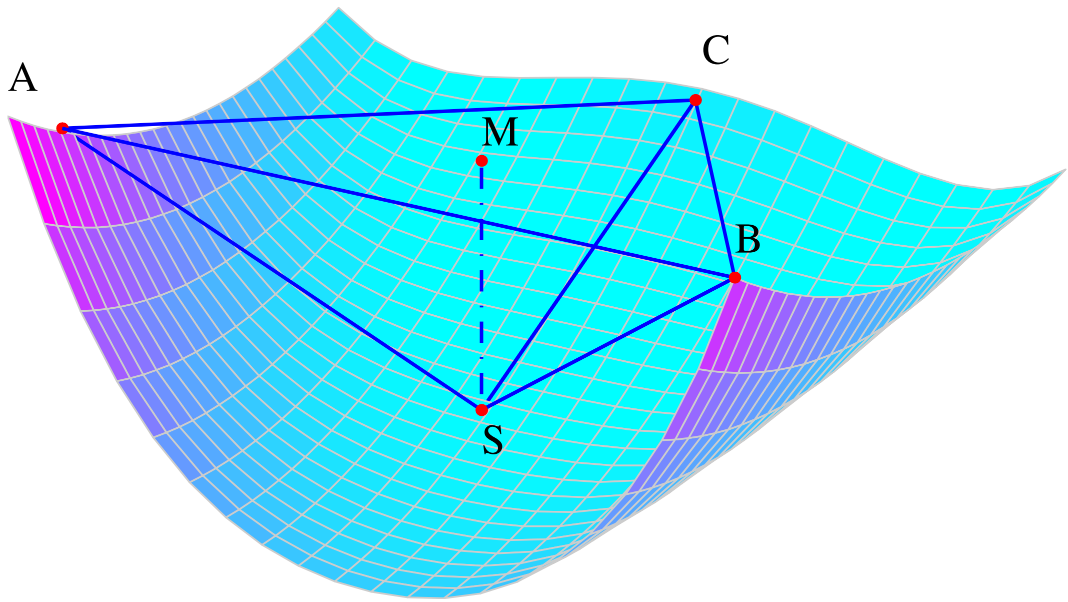


Figure 3.4: The tetrahedral error of a surface triangle with vertices A, B, and C is defined as the volume of the tetrahedron formed by the face ABC as its base and whose apex is the point on the inferred object surface S intersecting the normal situated at the midpoint of the surface triangle M.

different significance from that of exterior volumes. For example, if we wish to penalize volumes *outside* the generic model more than those on the *inside* (making the model confine itself to the interior regions of the data), then positive volumes can be given a higher weight than negative volumes:

$$E(A, B, C, S) = \begin{cases} 2 \times V & \text{for V > 0} \\ -V & \text{otherwise.} \end{cases} \tag{3.6}$$

The advantage of using the tetrahedral volume as opposed to the distance between the point on surface S and the midpoint M of the triangle (i.e., the apex length $|M - S|$) is that the volume is an area-weighted measure of the separation between the triangular element and the object surface. The distance measure $|M - S|$ does not take into account the size of the triangle from which it is being projected. Therefore, it will be biased toward those polygonizations that have fewer triangles. The tetrahedral metric is not a direct measure of the mismatched volume but rather a *sample* of the mismatch at a given surface element. There are many interior volumes that are not counted by this metric, but the important point is that this metric is *proportional* to the mismatched volume. Henceforth, unless otherwise noted, the term *error* used in the context of surface-element matching will refer to the tetrahedral error measure.

3.2.2 Landmark extraction

In the context of medical imaging, a landmark is an anatomical structure used as a point of reference for image orientation or as a localization point for finding other related structures within an image. A landmark will usually have a unique location (origin or center) associated with it. Complex landmarks can have additional parameters such as length, area, volume, curvature, and axes of orientation. In order to be useful, landmarks should have the following properties:

- They should be detectable on both CT and MR images. This is difficult because CT and MR generally provide complementary visual information. However, there are many techniques that can be used to deal with this problem.
- They should be robust. Landmarks that are inconsistent in their frequency of occurrence and that have highly variable parametric descriptions will be more difficult to detect and to disambiguate.
- They should be representable in a computationally tractable manner. Landmarks that can be described by points, lines, and triangular patches are computationally less expensive to detect than those that are described by higher-order curves and surfaces.

Landmark extraction can also be performed in two or three dimensions. Two-dimensional extraction is easier to implement, but it does not use all the information present in a three-dimensional data set. Furthermore, because of the strong influence of orientation angles and scanning parameters (e.g., slice thickness, slice separation, and scan origin), two-dimensional landmarks may not yield accurate results. Three-dimensional extractors can take advantage of spatial coherence information in all three dimensions. Hence, they can yield results that are less sensitive to scan parameters such as orientation and slice separation. However, three-dimensional extractors are still prone to undersampling errors and geometric distortions. One of the advantages of the geometric model of the spine is that it can eventually be used to improve the accuracy and reliability of an automated landmark-detection and -extraction algorithm. The model can be used to orient and guide the search for landmarks and can provide information useful in resolving ambiguous interpretations of an image.

I have identified four *principal landmarks* (to be extracted manually from the two-dimensional CT cross sections) that will provide the necessary parameters for performing the coarse match. The operator must first locate the correct slices for each vertebra using the CT scout scans. The scout scans provide a visual index to the cross-sectional slices by overlaying a graduated ruler onto a sagittal projective X-ray image of the cervical spine. Alternatively, a skilled operator with a good understanding of spinal anatomy can skip the scout scans and extract the landmarks directly from the cross-sectional images.

3.2.3 Spinal landmarks

As a first pass in determining the landmarks that would be most useful for image fusion, I interviewed several experts in radiology [28] [30] and neurosurgery [1] [55] and asked them to list which landmarks they look for when manually registering CT and MR images. The results of this survey are summarized in Table 3.1. Because human experts can rely on visual processing abilities that go far beyond the reach of even the most advanced machine-based techniques, the landmarks listed in Table 3.1 should not be considered an exhaustive set of computationally efficient landmarks.

Specialty	*Landmarks*
Neurosurgery	Posterior spinous processes Vertebral arteries Ligaments Spinal canal Disk saddle
Radiology	Cortical bone CSF Disk

Table 3.1: Summary of landmarks that experts in neurosurgery and radiology look for in performing image fusion.

Posterior spinous processes can be identified in CT images because of their high density and their characteristic Y shape. The density of cortical bone makes the spinous process appear as a brightly outlined region on CT (see Figure 3.5). On MR, however, the low concentration of water makes the spinous process appear as a dark region (see Figure 3.6). The interior region of the spinous process (see Figure 3.5) has a mottled pattern on CT images that is characteristic of bone marrow.

The chief drawback to using the posterior process as a landmark is that its anatomical structure varies considerably from patient to patient. In fact, it is one of the most variable structures in the human spine. Also, the intrapatient intravertebral variation can be as significant as the interpatient variation. However, from the perspective of model matching, the spinous process is an important feature that must be taken into account. As a result, I have chosen the posterier notch of the spinous process as one of the four principal landmarks.

Much of the blood supply to the brain comes from the pair of opposing *vertebral arteries* that run laterally through the foramina transversaria. The vertebral arteries originate from the subclavian artery deep within the neck and typically enter the spinal column through the 6th cervical vertebra. From there they ascend through each of the remaining vertebrae until they emerge through the foramina of the atlas (C1). The arteries themselves are not visible on plain CT, although the foramina transversaria through which they pass are visible as dark circular voids (see Figure 3.7). The arteries are visible in most MR images as low-intensity regions (see Figure 3.8). Static fluids generally produce a strong image signal in MR, but rapidly moving blood has less net magnetization and produces a low-intensity signal.

Unfortunately, there are several variables that can limit the accuracy of landmarks based on the vertebral arteries:

Figure 3.5: The posterior process, indicated above, has a distinctive appearance on this CT image. The cortical regions of the process are denser than the marrow, and hence, appear somewhat brighter than the inner regions.

Figure 3.6: The posterior process, indicated above, has a different appearance in MR images. The low concentration of water in the posterior process makes it look like a dark region. Contrast with respect to the surrounding tissue makes the posterior process stand out prominently.

- The foramina are designed to provide enough clearance so that the vertebral arteries are not constricted by normal movement and flexion of the spine. As a result, the radii of the transversaria as perceived on the CT images will differ from the radii of arteries as seen on the MR images.
- Depending on the curvature of the spine, the arteries may not necessarily pass through the centers of the foramina transversaria. As a result, the centers of the arteries may not coincide with the centers of the transversaria.
- Most people have a dominant vertebral artery whose size can vary significantly from the opposite artery. In fact, it has been estimated that 15 percent of the population have vertebral arteries whose relative diameters can vary by at least a factor of two [75]. As a result, the shift of the center of the suborbinate artery can be even greater than that of the dominant artery.
- The vertebral arteries can enter into the cervical column at locations above or below the 6th cervical vertebra. It is not unusual for the arteries to enter at either the 7th or 5th vertebra (there is even a reported incidence of the arteries entering at the 3rd cervical vertebra [78]).

As a result of these variables, the vertebral arteries and transversaria may not be robust enough to be used as landmarks for MR images. However, for extraction of vertebra in CT images, the arterial foramina through which the arteries pass provide excellent landmarks. They are easy to locate and are clearly visible on all spinal scans. For this reason, I have selected the two opposing foramina to be included in the set of four principal landmarks.

The *spinal canal* (also known as the vertebral foramen) is the opening through which the spinal cord and CSF pass and is present in each of the cervical vertebra. It has a very distinctive appearance in both CT and MR images. In CT, it appears as a dark, bowl-shaped void region when sliced through the middle of the vertebral body (see Figure 3.9). In MR, it generally appears as a bright region surrounding the spinal cord (see Figure 3.10). The spinal canal appears to provide the single most convenient landmark for aligning the central axis of the spinal column.

Because of its distinctive appearance, the spinal canal makes an ideal landmark for both CT and MR images. I have chosen the posterior trough of the spinal canal as the fourth principal landmark (see Figure 3.9). This landmark is easy to locate and is clearly visible in virtually all spinal scans.

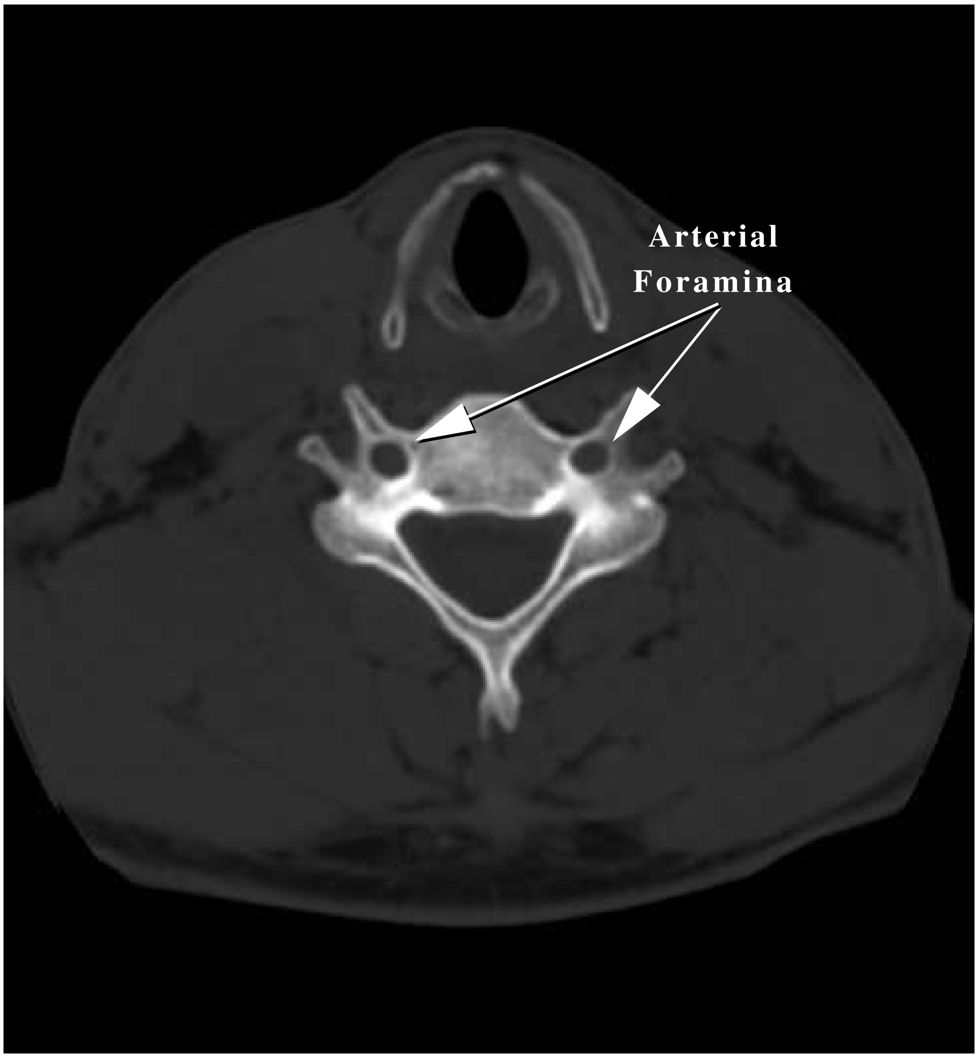


Figure 3.7: The arterial foramina (shown on CT) allow the vertebral arteries to pass through the cervical column parallel to the spinal column.

Figure 3.8: The arterial foramina are less distinct in MR images than in CT. However, the vertebral arteries which pass through the foramina appear as bright regions due to the high concentration of fluid (i.e., blood).

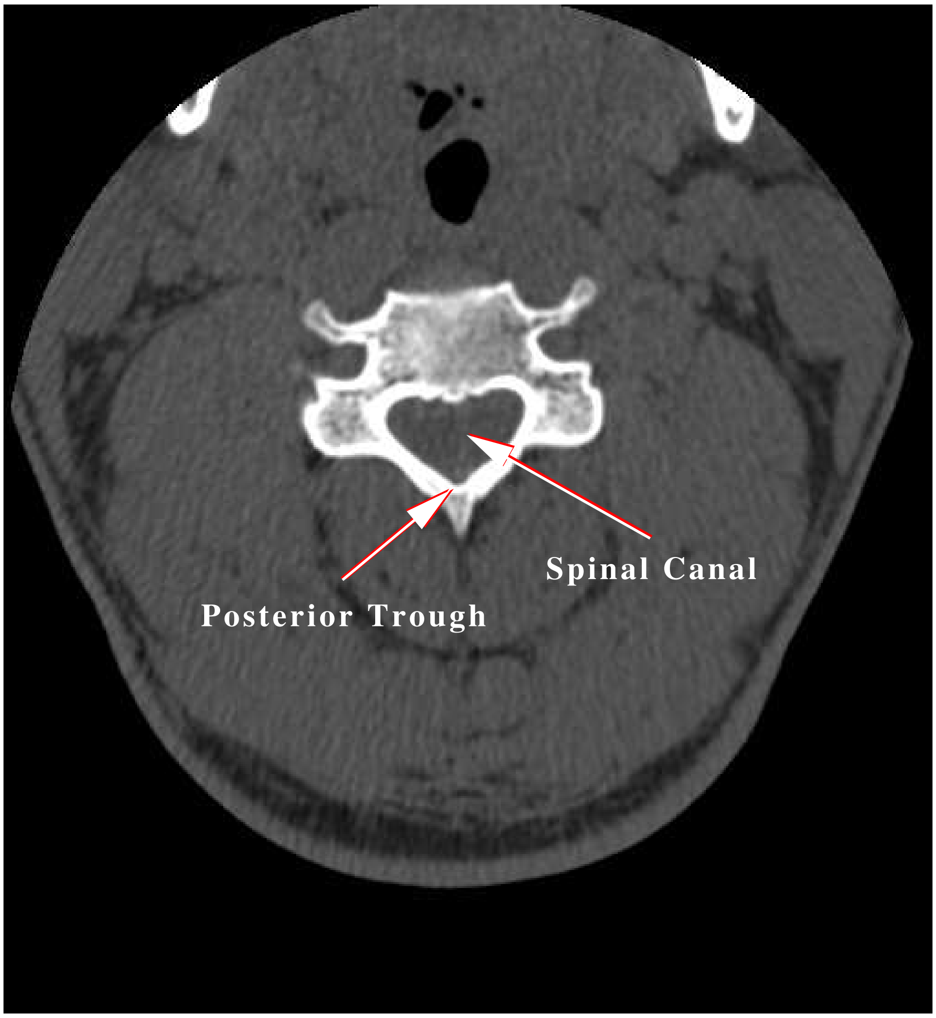


Figure 3.9: The spinal canal, which surrounds the spinal cord, is visible as a low-intensity region surrounded by the higher intensity lamina, pedicles, and vertebral body.

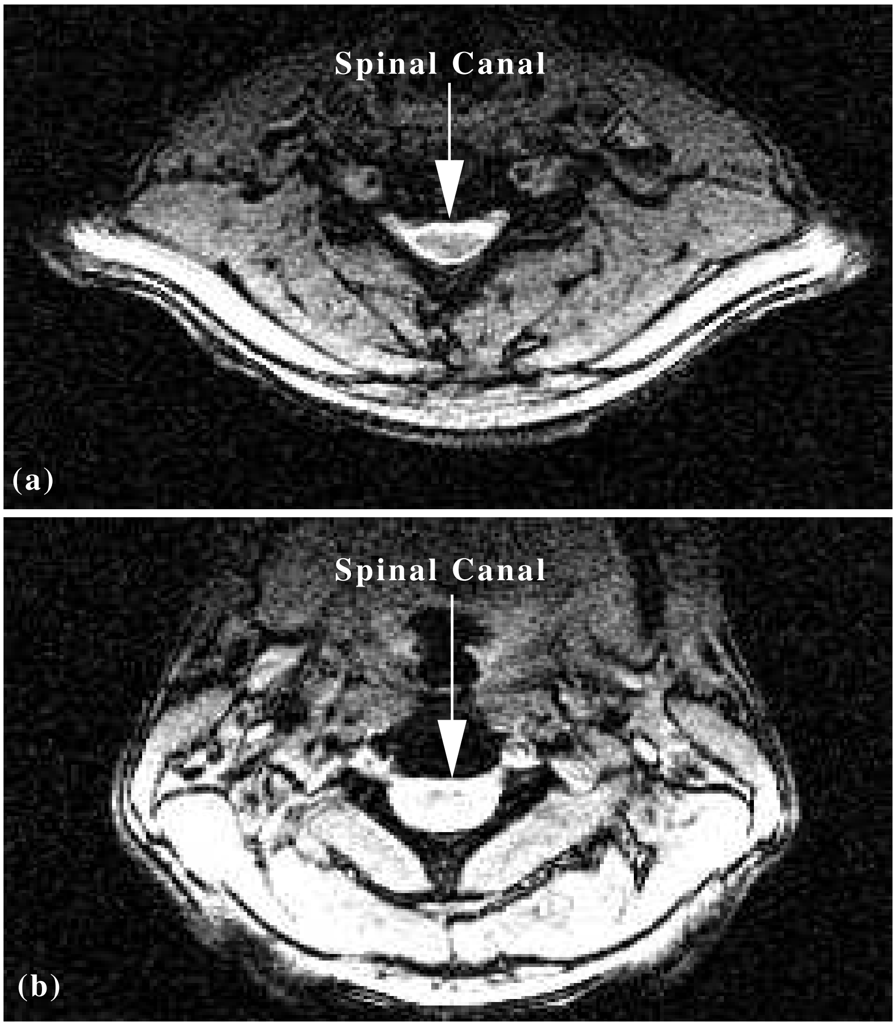


Figure 3.10: In MR images, the appearance of the spinal canal varies depending on the slice plane level. In (a), the foramen appears as the dark region surrounding the spinal cord and CSF. In (b), the foramen appears as a dark region interrupted by dorsal and ventral root nerves.

Based on my study of spinal landmarks, I have defined four principal landmarks that are adequate for performing a coarse match of the generic model to a CT data set:

1. Left arterial foramen (AFL), medial
2. Right arterial foramen (AFR), medial
3. Spinous process notch (SPN), medial
4. Spinal canal trough (SCT), superior

These four point landmarks specify a homogeneous transformation matrix that translates, scales, and rotates the generic model so as to match the size, position, and orientation of the vertebrae in the data. Furthermore, these four landmarks can be used to create three non-orthogonal basis vectors which, in turn, provide a means for parameterizing the surface elements of the generic model in a localized coordinate system (see Figure 3.11).

All of the principal landmarks must be located at a specific z-location. In particular, the AFL, AFR, and SPN landmarks must all be specified in their medial z-location. This means that if the landmark is visible across multiple adjacent slices, the operator must choose the slice that lies in the middle. The SCT landmark must be specified at its most anterior location. This is to maximize the likelihood that the four landmarks will be non-coplanar. If the landmarks are coplanar, the SPN must then be specified at a different z-location (preferably the medial location). Non-coplanarity of the four landmarks is a necessary condition for determining the homogeneous transformation matrix that performs the crude match of the model to the data.

3.3 Cervical Spine Model

Because the vertebral model was the key to the segmentation algorithm I have developed, it was imperative to design a model that satisfied several important criteria. Webster's dictionary defines 13 meanings for the word *model*. The definition that comes closest to describing the cervical spine model presented in this chapter is the following:

A system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs [84].

The term *system* describes an important attribute of the model developed in this dissertation. The spinal model is not a monolithic entity. Instead, it is a system made up of geometric descriptors, statistical data, and mathematical operators for extracting features from CT and MR images. In this section, I will describe in detail the different design goals and implementation aspects of the spine model.

There are several key questions that need to be addressed when designing a model. The most important is, "What is the primary purpose of this model?" The answer to this question will help determine the choice

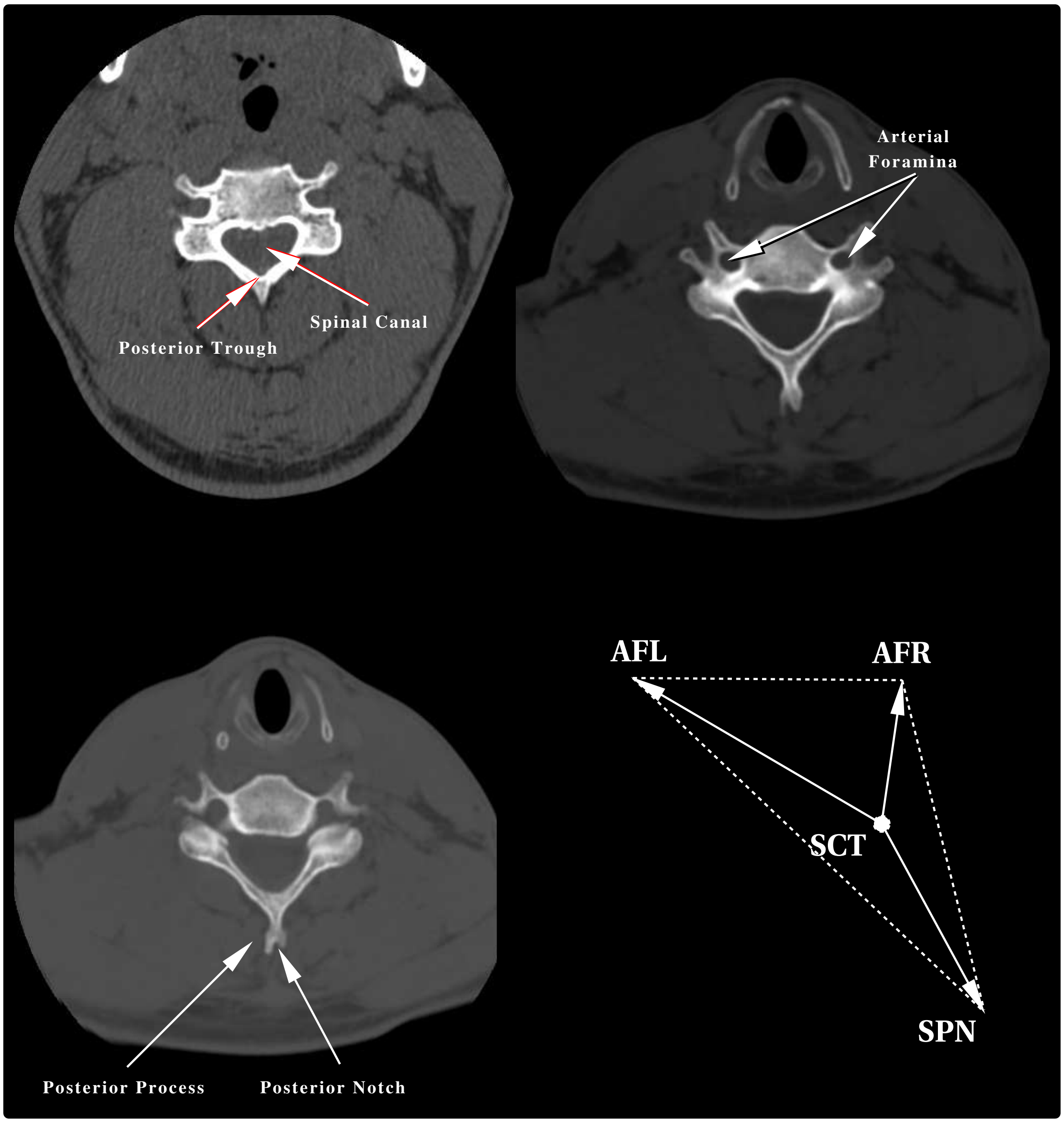


Figure 3.11: The four principal landmarks (AFL, AFR, SPN, SCT) provide sufficient information to determine the homogeneous transformation matrix that performs a crude match between the generic vertebra model and the input data set. When the SCT is taken as the origin, the four landmarks can also be used to define a set of non-orthogonal basis vectors for parameterizing the local features of the spine model.

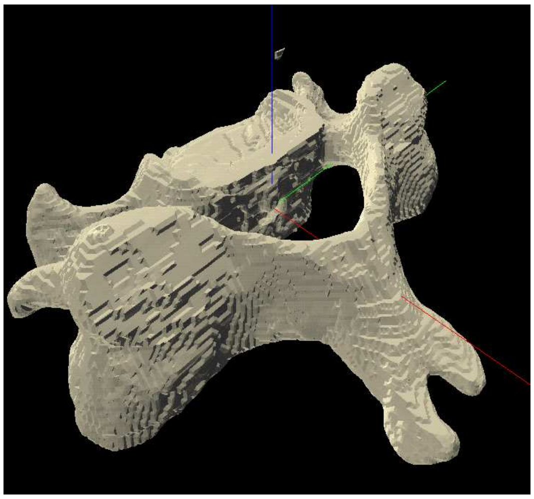
of various model types (e.g., qualitative or quantitative, static versus dynamic, fixed versus adaptive) and will influence the parameters that need to be captured by the model. Another important question deals with the level of accuracy required of the model. This will influence factors such as the complexity of the model components (for example, whether a flat as opposed to a hierarchical organization is more appropriate). Occam's principle (and common sense) would suggest the simplest possible model, although in practice models tend to grow in complexity during the implementation process. Finally, in practical applications, cost considerations will affect the performance and accuracy of the actual model. In most research settings, models tend to deemphasize factors such as cost and speed. Instead, they are biased toward delivering exact analytic solutions, even though such solutions rarely exist in the real world.

I selected the following criteria to guide the design of the spine model:

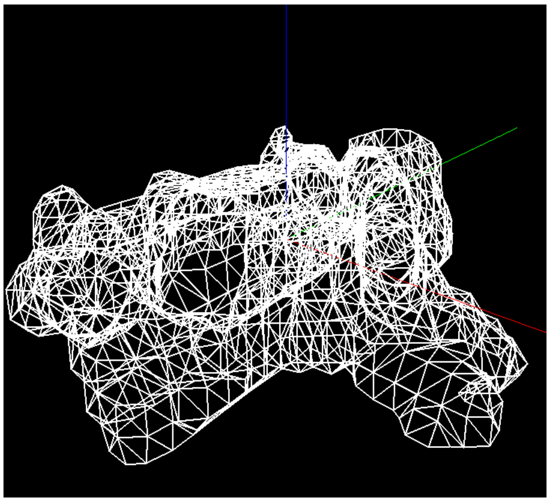
- The primary purposes of the spine model are to guide the operation of low-level extraction algorithms and to help interpret ambiguous or missing information.
- To support this function, the model has to provide geometric information regarding the size, shape, position, and orientation of all the relevant anatomical objects and landmarks.
- The model should aid in the visualization of both fused and unfused CT and MR data sets.
- Because the model has to adapt to both normal and abnormal variations seen in the patient population, it should provide a means for representing statistical information such as mean positions, variances, and covariances of the geometric objects being modeled.
- The model should provide a framework to support the long-term goal of performing biomechanical analysis and simulation.

To satisfy all of these criteria, I have developed a hybrid model called an *augmented finite-element model* (**AFEM**) (see Figure 3.12). An AFEM consists of a *finite-element model* (**FEM**) augmented by additional structures that can be used to locate landmarks, contours, surfaces, and regions. These surfaces and volumes can be used by statistical estimation modules (such as Kalman filters) to automatically extract landmarks from various regions, using the finite-element model as a road map. Once these landmarks have been found, the desired vertebrae can be extracted by refining the AFEM to minimize the distance between its triangular surface elements and the boundaries in the image data.

The AFEM uses the structural modeling capabilities provided by well-established finite-element techniques that have been used extensively in the fields of civil engineering, structural engineering, solid mechanics, and fluid mechanics. Augmenting the finite-element model with statistical parameters allows it to adapt to geometric variations observed in the patient population. The model contains a state vector of parameters for each anatomical object (disks, vertebrae, etc). The state vector contains information such as the first- and second-order moments, principal axes, relative vertex positions, and points of curvature extrema. The mean and covariance matrix for the state vector can be computed from a representative



(a)



(b)

Figure 3.12: The vertebral segment shown in (a) can be represented by a finite-element surface model (b). The finite-element model (FEM) consists of a set of triangular surface elements (patches) that describe the coarse external boundaries of the vertebral segment. The model shown in (b) contains approximately 5,000 triangular surface elements.

sample of patient images (i.e., the training set). The mean vector defines the generic form of the geometric model (also referred to as the *generic model*). The diagonal elements of the covariance matrix define the statistical variance of each state vector parameter. The variance is an indicator of the degree of variation that should be expected for each parameter in the actual patient population. The off-diagonal elements of the covariance matrix capture the pairwise correlations among the state vector parameters. This information can be used by the extraction algorithm to make reasonable boundary estimates and ranges using a sparse or incomplete set of parameters, or to remove statistical outliers from the state vector.

3.4 Model Construction

Most analytical formulations of a physical problem start with mathematical equations that are both continuous and smooth. For example, the distribution of forces across a beam can be described easily in terms of an analytical function applied to a geometric object, which, in turn, can be described by another analytical function. In the real world, however, most objects cannot be described easily by continuous analytical functions. Anatomical objects such as vertebral bodies are too complex to be described by a single analytical function. Furthermore, computers are not yet sophisticated enough to perform closed-form analytical computations of large-scale systems in an efficient manner. However, by breaking down geometric objects into smaller *elements*, it is then possible to apply simpler analytical functions that adequately describe the physical behavior of the larger objects in terms of these elements.

In the context of finite-element analysis, the term *discretization* means (1) to partition the model into smaller subdomains and (2) to convert continuous variables into discrete variables. The benefits of discretization are simplified mathematical equations and smaller computational resources to solve complex models. High-order equations can usually be converted to lower-order equations and partial differential equations can be converted into ordinary differential equations. In turn, these differential equations can be converted to linear equations. Although the elements are small enough to be described by discrete equations, the term *finite* implies that these elements are large enough that their parameters can be solved for in a computationally efficient manner. One of the critical tasks in designing finite-element models is to choose the proper scale for each of the elements so that the physical domain is adequately described in a computationally efficient way.

The discretization of the three-dimensional finite-element surface model of the vertebral elements is performed by using a modified marching cubes tessellation algorithm (a three-dimensional threshold-based surface extractor). The CT data set used to construct the model was selected from a sample of patients and chosen on the basis of having geometric properties that are closest to the mean of the sample data. The data set was first edited to remove any artifacts before being fed to the tessellation algorithm. The output of the tessellation algorithm consists of a set of surface triangulations comprising each individual vertebra.

The original marching cubes tessellation algorithm proposed by Cline [19] creates a polygonization of an object by classifying each voxel in an image volume as either belonging to, or not belonging to, an object or class of objects (e.g., a voxel can be classified as being either bone or non-bone). Once all the voxels have been classified, the algorithm then "marches" through the volume in a sequential fashion and considers each cube whose vertices are the centers of eight adjacent voxels (see Figure 3.13a). For classifying bone, the usual procedure is to define a bone threshold value (typically around 1500 for CT images) and group all voxels with intensity values greater than or equal to the threshold into the bone category. If all the vertices of a given cube have the same classification, then there is no surface that tessellates (i.e., passes through) that cube. However, if at least two vertices differ from each other, then there must be a surface that tessellates the cube.

The tessellating surface can be approximated by a polygon whose vertices lie along three or more edges of the cube. The locations of the vertices can be found by interpolating the values of the two endpoints on the cube edge to find out where the transition between the two classifications occurs. In Figures 3.13b, c, and d, the vertices of the bone surface intersect the edges of the cube where the value is equal to 1500. In this example, the locations of these intersecting points are found by linear interpolation. In Figure 3.13b, the surface tessellates the cube at three locations, hence the polygonal surface has three edges. In Figure 3.13c, the surface tessellates the cube in four locations, while in Figure 3.13d, the surface tessellates the cube in six locations (the maximum possible).

Polygons with more than three sides can be decomposed into component triangles by choosing a common vertex point and drawing chords to all the non-neighboring vertices (see Figure 3.14). Figure 3.12b shows the resulting surface model generated from the fourth cervical vertebral segment (C4). Note that the choice of the common vertex is arbitrary and can affect the optimality of the resulting decomposition (for example, long skinny triangles can be created if the wrong vertex point is chosen). A somewhat more optimal triangular decomposition can be achieved by decomposing the cube itself into its five component tetrahedra and performing the tesselation on each of these smaller polyhedra (see Figure 3.15). Regardless of the tessellation volume used, it is important to note that, in some cases, it is impossible to infer the exact topology of the surface by merely examining the values at the vertices. Consider the two-dimensional example shown in Figure 3.16. The dark circles represent vertices that exceed the bone threshold while the light circles fall below the threshold. In this situation, there are two interpretations for the tessellating boundaries shown in Figures 3.16a and b. There is no way to resolve the ambiguity using the vertex values alone. Additional knowledge about the shape of the surfaces is required to make a correct decision. However, it is possible to use a *consistent* rule that always selects the interpretation with the fewest surfaces, as in 3.16b. Such a rule is called a *greedy* rule because it tends to form the largest surfaces possible. In some situations, a better approximation can be made by performing a bicubic interpolation of the surface intersection points and fitting a spline to the tessellation boundary. This approach would be accurate only in those cases where the surfaces are in fact polynomials of low degree (three or less).

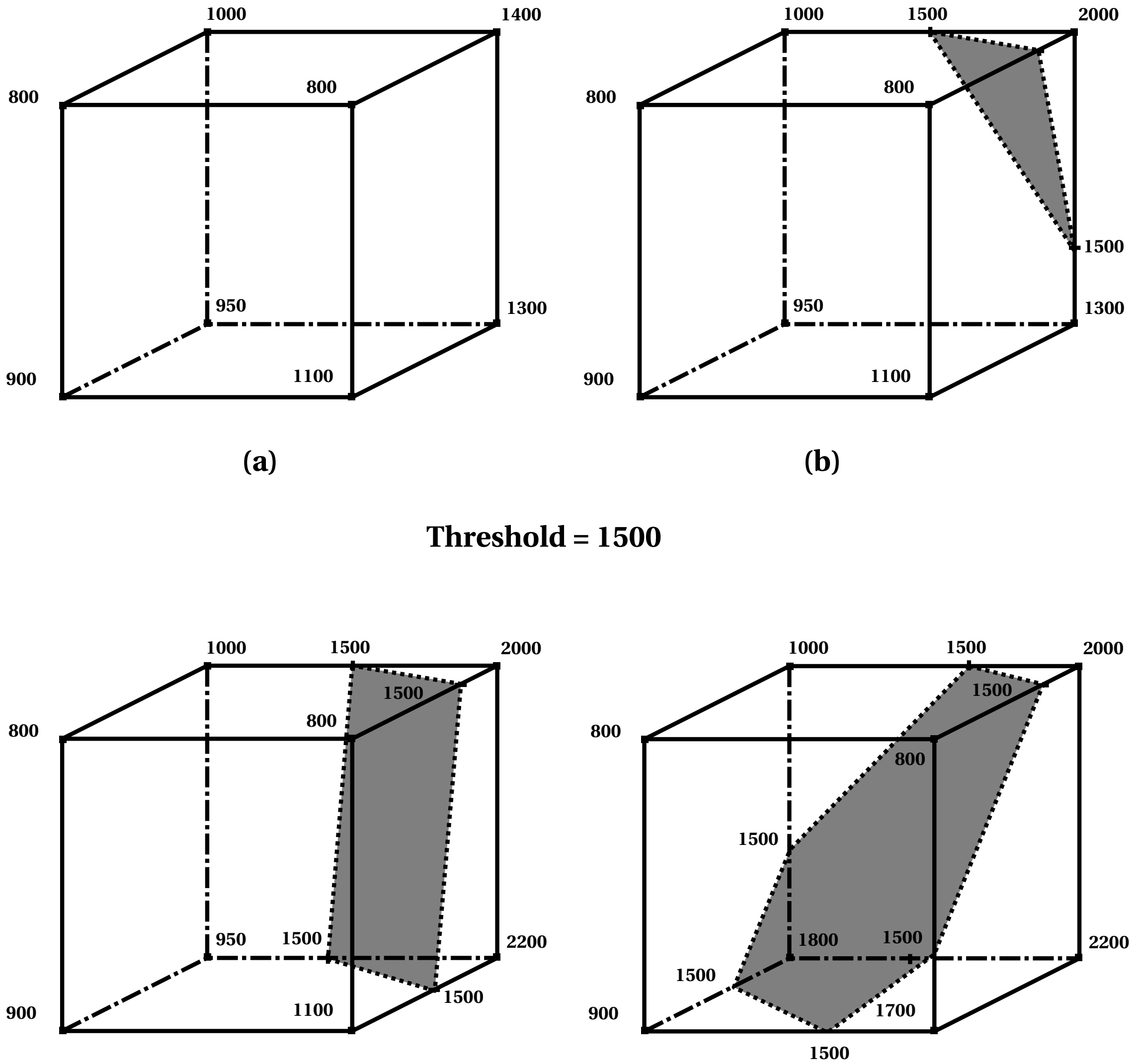


Figure 3.13: The basic principle behind the marching cubes algorithm is to (a) divide the volume into a stack of cubes (the dimensions of the cubes are usually equal to the size of each voxel). For those cubes that have differing voxel types at the vertices, a surface polygon is constructed with vertices that intersect the cube edges at the boundaries between two voxel types. In this example, the two voxel types are bone and non-bone. Those voxel values that are equal to or exceed the threshold value of 1500 are categorized as bone; all others are non-bone. In (b), the bone threshold intersects the cube in three locations, hence a surface triangle is formed. In (c), the threshold intersects the cube in four locations, hence a quadrilateral is formed. The maximum number of intersections possible from a single surface element is six, as shown in (d).

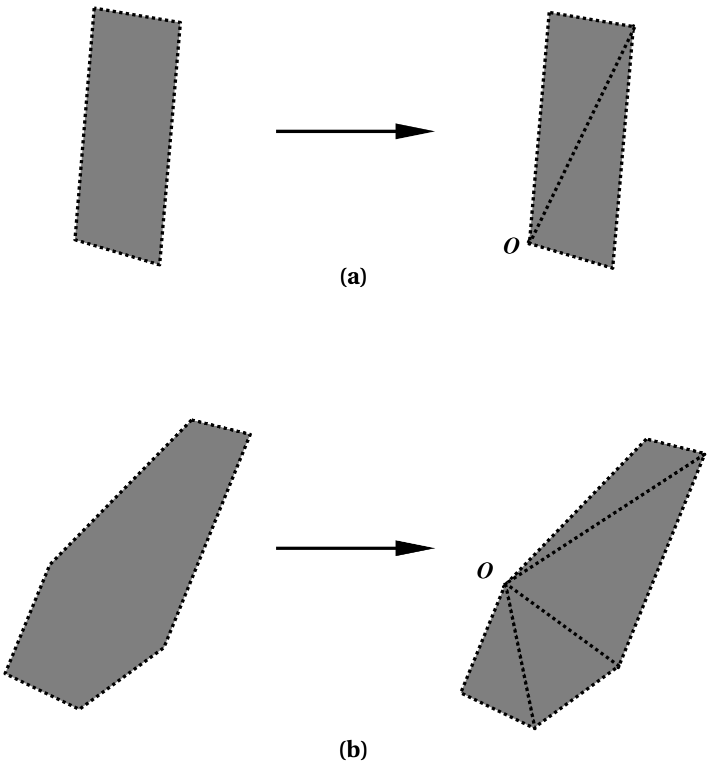


Figure 3.14: Polygons with more than three sides are decomposed into triangles by choosing a common vertex point called O and drawing chords to all the non-neighboring vertices. A quadrilateral (a) has one such chord, while a hexagon (b) has three.

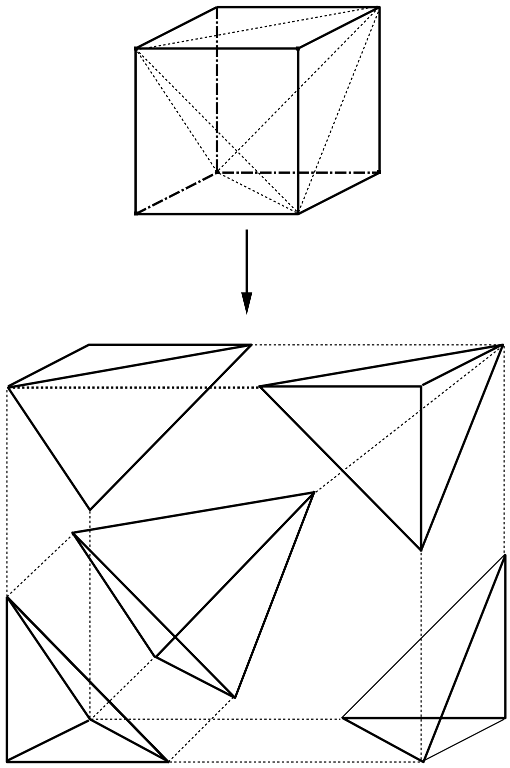


Figure 3.15: A slightly more optimal surface triangulation can be created by decomposing each cube into its five constituent tetrahedral volumes. As with the cube tessellation, the intersection points for the vertices are determined by interpolating the values along the edges of each tetrahedron. However, unlike the cube, the tetrahedral tessellation *always* produces triangular surfaces (at most three edges of a non-degenerate tetrahedron can be tessellated by a surface).

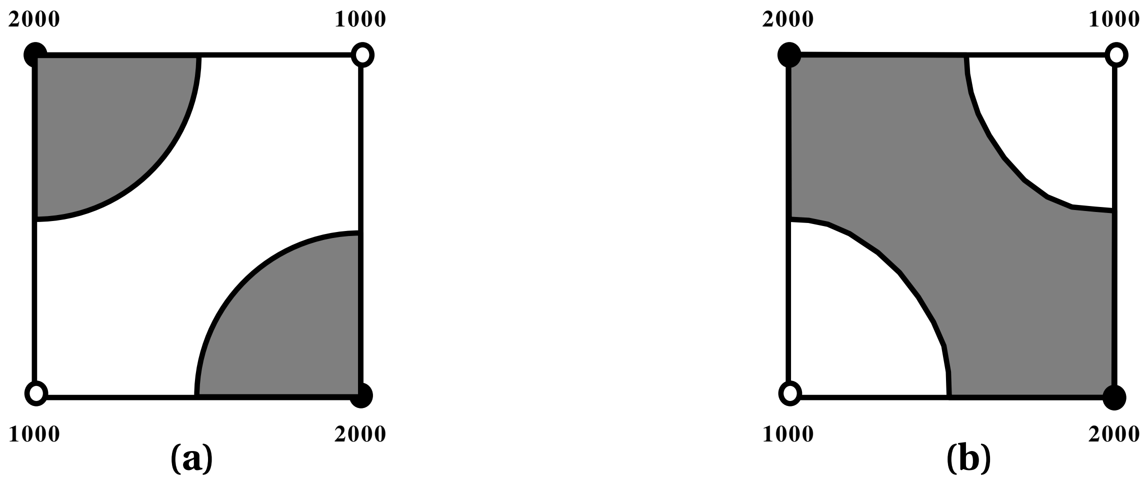


Figure 3.16: An example of an ambiguous surface tessellation in the two-dimensional case. A minimal disambiguiting rule would interpret the surface as shown in (a) while a greedy disambiguiting rule would interpret the surface as shown in (b).

3.4.1 Slice data segmentation

Before the data for building the FEM could be fed into the tessellation algorithm, the vertebral bodies in each slice had to be isolated (i.e., segmented) from their surroundings. The segmented data were then concatenated into a three-dimensional volume and fed to the tessellation algorithm. Because the number of resulting surface elements was so large, I developed an algorithm for pruning the number of triangles in the resulting surface model (described in Section 3.4.2).

An interesting phenomenon occurred as the model was successively pruned: in certain areas, the model began developing strange growths that resembled long polyps. After carefully examining the original data sets, I determined that this problem was caused by errors in the original vertebral classifications (i.e., inaccurate segmentation of the two-dimensional slices). The polyps were caused by regions that belonged to neighboring vertebrae or disk tissue that had not been properly delineated and separated. These misclassified regions were always present in each model regardless of the complexity level. The problem was that these misclassified regions were difficult to visualize in the higher complexity models. Misclassification errors did not get smoothed out by the model-pruning algorithm. They simply got propagated until they eventually became easier to visualize. Proper classification of the original data set proved to be crucial in building an accurate three-dimensional model.

Rather than develop a technique for fixing these errors (using a clever three-dimensional smoothing and editing programming), I developed a better two-dimensional interactive region-based segmentation technique to generate more accurately segmented vertebra. This incremental thresholding technique works by setting a high threshold value to create a binary image as shown in Figures 3.17a and b. The binarized image is then classified by assigning each distinct region a unique numeric label (Figure 3.17c).

These regions are then grown by incrementally reducing the threshold value until it reaches the proper bone surface threshold limit (Figure 3.17d). As the regions grow, those that were originally distinct remain distinct even if they overlap at the final threshold value. A logical union operator is used to merge those regions that have different labels but belong to the same object.

The advantage of this technique is that it produces highly complex boundaries very rapidly without having to manually delineate the regions. It is analogous to building an object out of large jigsaw pieces instead of sewing it together using a fine needle and thread. One major benefit of this classification technique is that the initial regions are never discarded. If an error is found in the three-dimensional model, the user can go back to the original data and easily change the projection operator on a *local* basis and regenerate the entire model automatically without propagating any *global* errors. This is a significant improvement over other classification methods in which changing a single parameter can affect the entire model.

Figures 3.18 and 3.19 show the FEM generated by segmenting a C4 vertebra using this technique and feeding the resulting data to the tessellation algorithm. This particular model has 75,918 vertices and 151,842 surface elements and is quite cumbersome to manipulate, visualize, and perform computations on. Furthermore, it contains far too much patient-specific curvature information to be useful as a smooth generic model. In the next section, I address this problem and provide a simple remedy.

3.4.2 Model simplification

Because the marching cubes algorithm does not take into account the continuity information of the tessellated surfaces, large sets of triangles are needlessly generated in regions where the curvature is very low. Consequently, a smoothing and triangle decimation step is used to reduce the number of triangular surface elements in these model regions. A highly decimated and smoothed model can be used to perform a coarse match to the data, while a minimally decimated model can be used to refine the coarse match.

The number of triangles can be reduced by using one of two operations: (1) coalescing two adjacent vertices within a surface triangle along their common edge (as shown in Figure 3.20a), or (2) coalescing all three vertices within a given triangle into a single vertex (as shown in Figure 3.20b. These operations only need to be performed on vertices for which the resulting triangulation remains within a prespecified error tolerance. A larger error tolerance will result in a greater degree of decimation with fewer triangles and a coarser approximation of the original model. Alternatively, if an absolute number of triangles needs to be eliminated, then each triangle can be ranked in order of its contribution to the local curvature. The triangles can then be eliminated in reverse order until the desired number of triangles remains in the list.

The difficulty with this approach is that it requires extensive computations to determine the contribution to the curvature made by each triangle. Furthermore, this approach is counterproductive if the goal is to achieve smoother models. As a result, I have devised a simpler strategy based on the minimum altitude of the surface triangles. This method ranks each triangle according to the length of its smallest altitude,

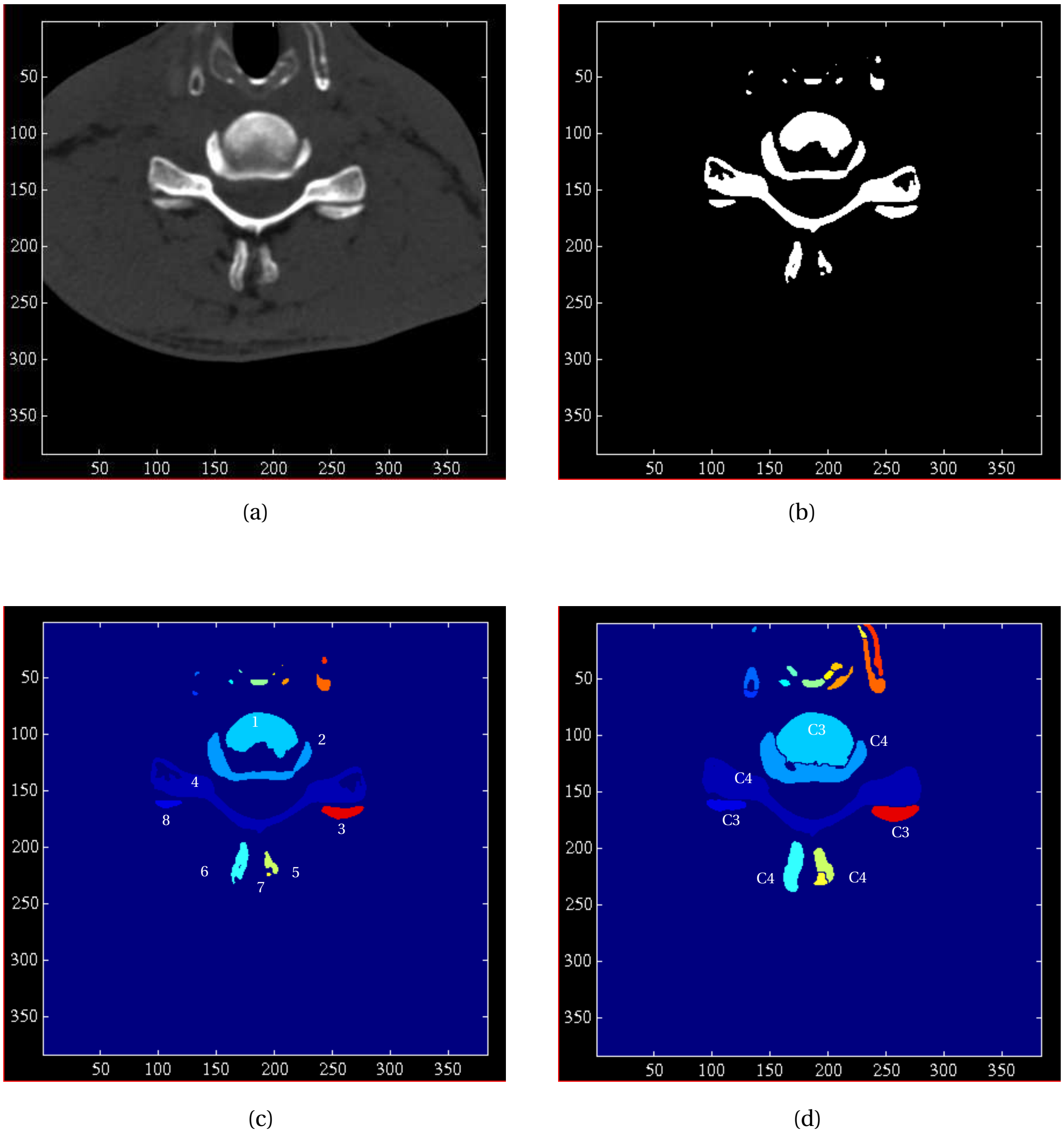


Figure 3.17: The incremental thresholding algorithm for segmenting the spine data (for the initial model) starts by binarizing the original image (a) into regions that contain bony pixels (b). The distinct regions are assigned numeric labels (c) and as the threshold value is incrementally lowered, the regions are regrown while preserving the boundaries between distinct regions. When the final threshold value is reached, the distinct regions can be grouped together according to their proper structural membership (d).

Figure 3.18: The FEM generated from a segmented C4 vertebra by using the marching cubes tessellation algorithm with polygonal decomposition. This model contains 75,918 vertices and 151,842 triangular surface elements.

and discards those triangles that fall below a standard deviation threshold limit, σ_a. This computation is much faster and yields smoother-looking models (see Figure 3.21). The holes in the model left by the discarded triangles are then patched up by merging the adjacent vertices as described earlier. By default, the threshold value is set to zero sigma (i.e, the median value), but can be set to any arbitrary value. With a zero sigma threshold, each iteration of this algorithm reduces the complexity of the model by a factor of two. A pyramidal hierarchy of models can be constructed by repeatedly eliminating half of the triangles starting with the original high-complexity model. I have successfully iterated the initial model through ten cycles, ultimately reducing the overall complexity of the model by over a thousandfold. Figure 3.21 shows the results of this decimation technique for reduction factors of 16, 256, 1024, and 4096. The highly reduced models (with the exception of the last one) capture all the salient features of the vertebra.

It should be noted that decimation and smoothing are two distinct operations that can be performed on the model. With decimation, triangles are selectively removed from the model. With smoothing, the vertices of the existing triangles are adjusted in order to reduce the curvature variations of the model. If we assume that vertex points are adjusted along the direction of their surface normals to match the input data, then it is essential that the generic model be smoother than the patient data (as shown in Figure 3.22a). Otherwise, it will not be possible to match the local variations and local features of the patient's anatomy. A model that captures too much specific information from the training data sets is said to be *overfitted* (as shown in Figure 3.22b). The most common method for performing smoothing is to replace the value of vertex location with a weighted mean of itself with its adjacent vertices. That is,

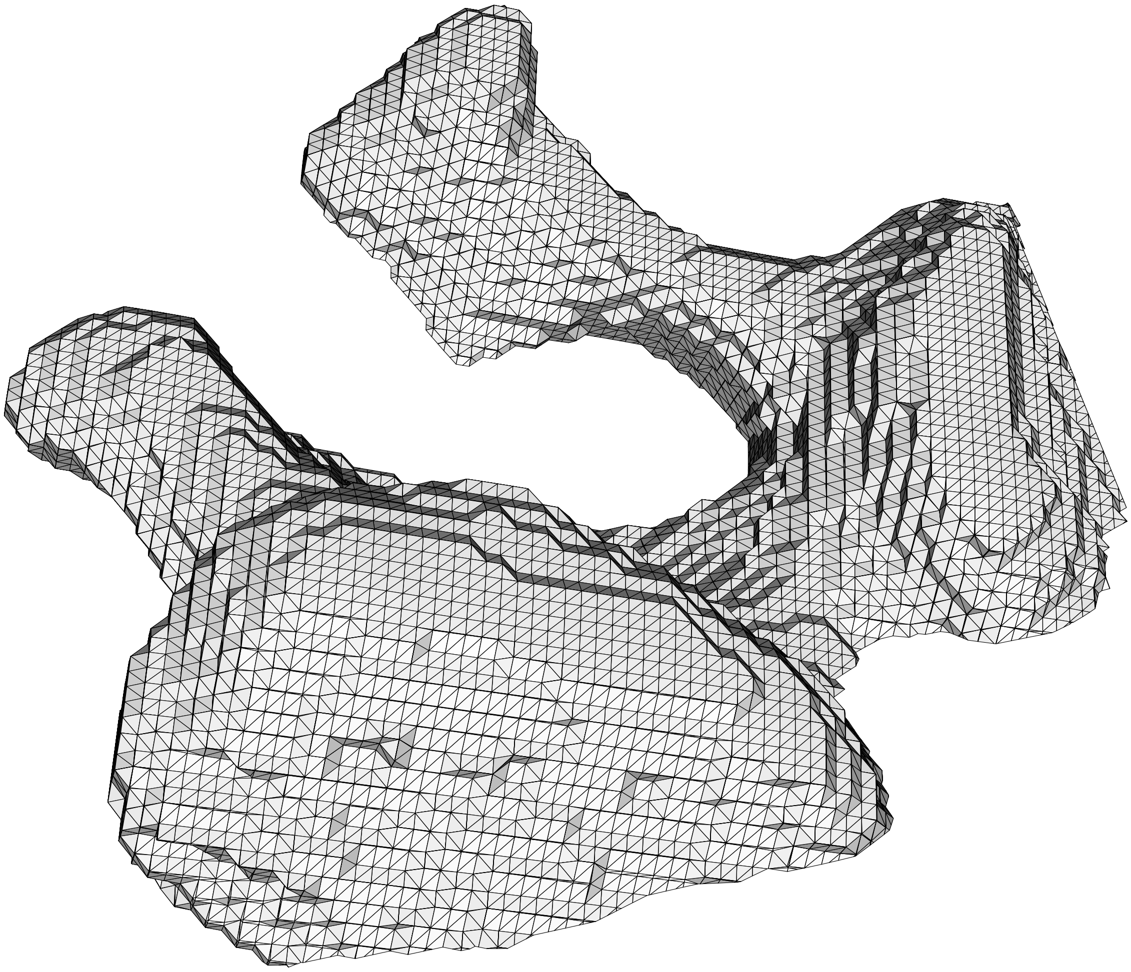


Figure 3.19: A close-up view from a section of the FEM from 1 showing the individual triangular surface elements.

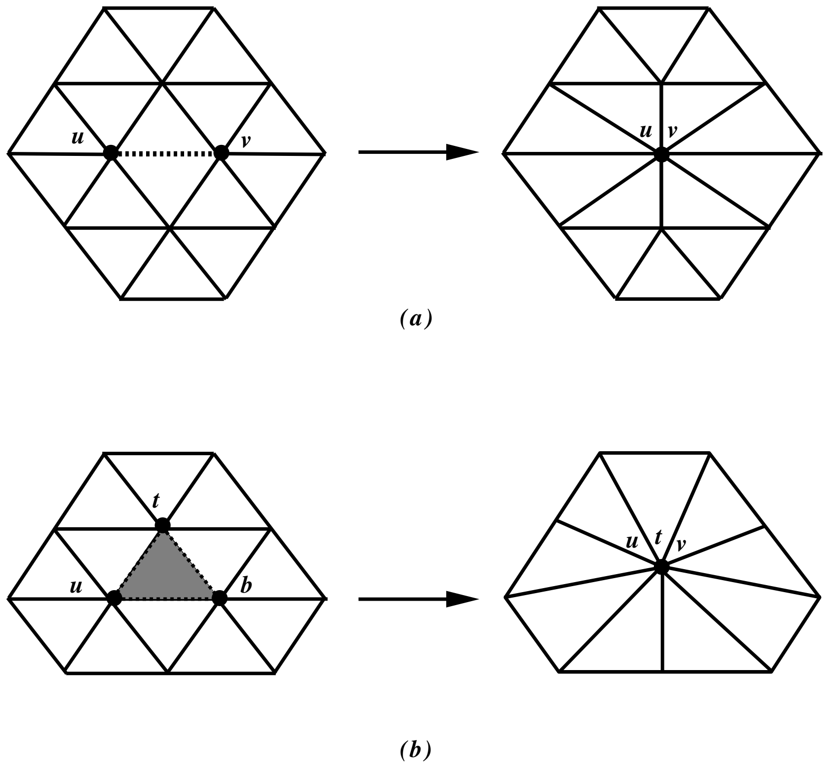


Figure 3.20: Triangles can be eliminated by (a) coalescing adjacent pairs of vertices within an edge, or (b) coalescing three vertices within a triangle.

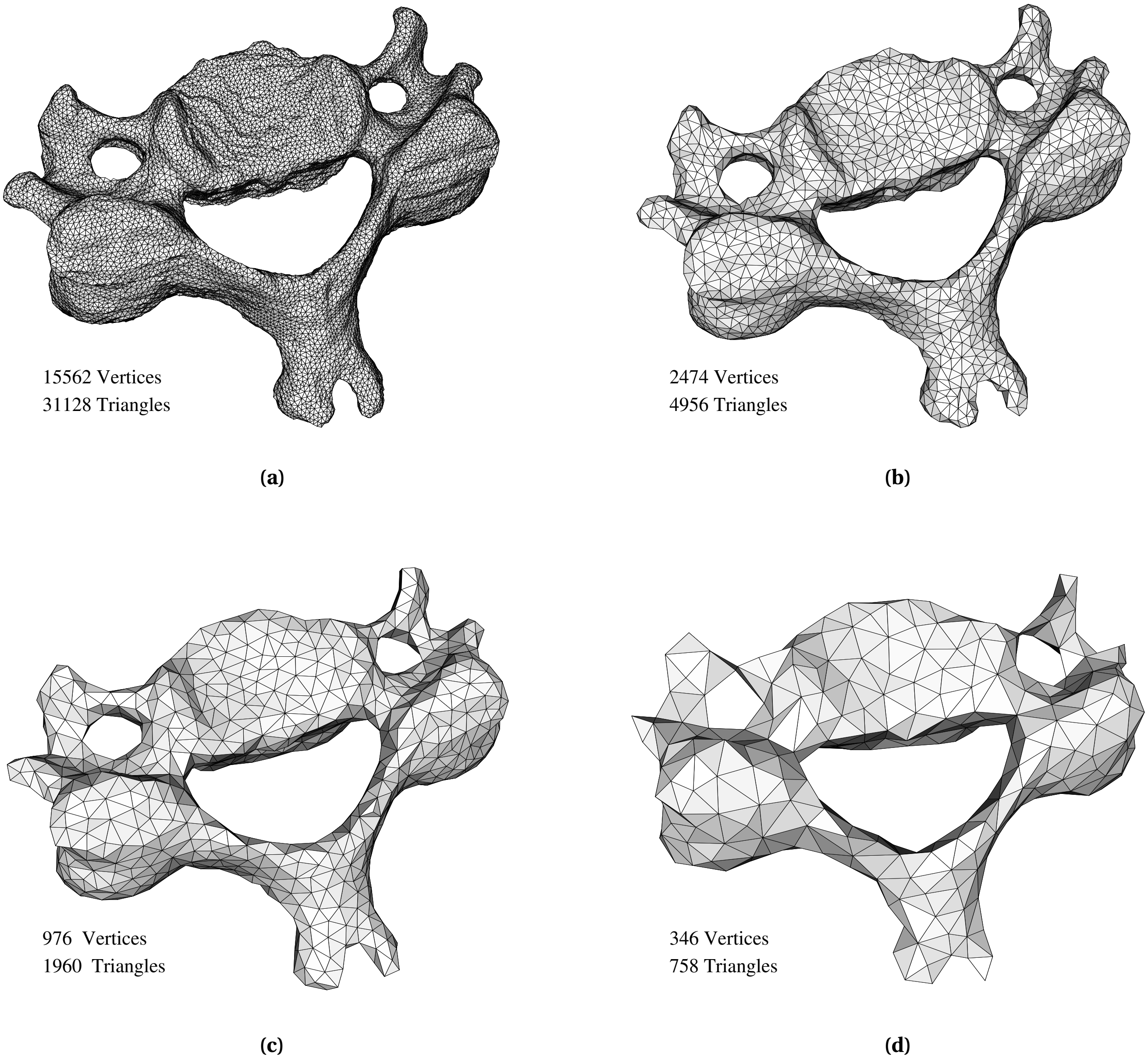


Figure 3.21: The original vertebra FEM of 1 has been decimated by factors of (a) 6, (b) 256, (c) 1024, and (d) 4096. With the exception of (d), the decimated models capture all the salient features of the original vertebra.

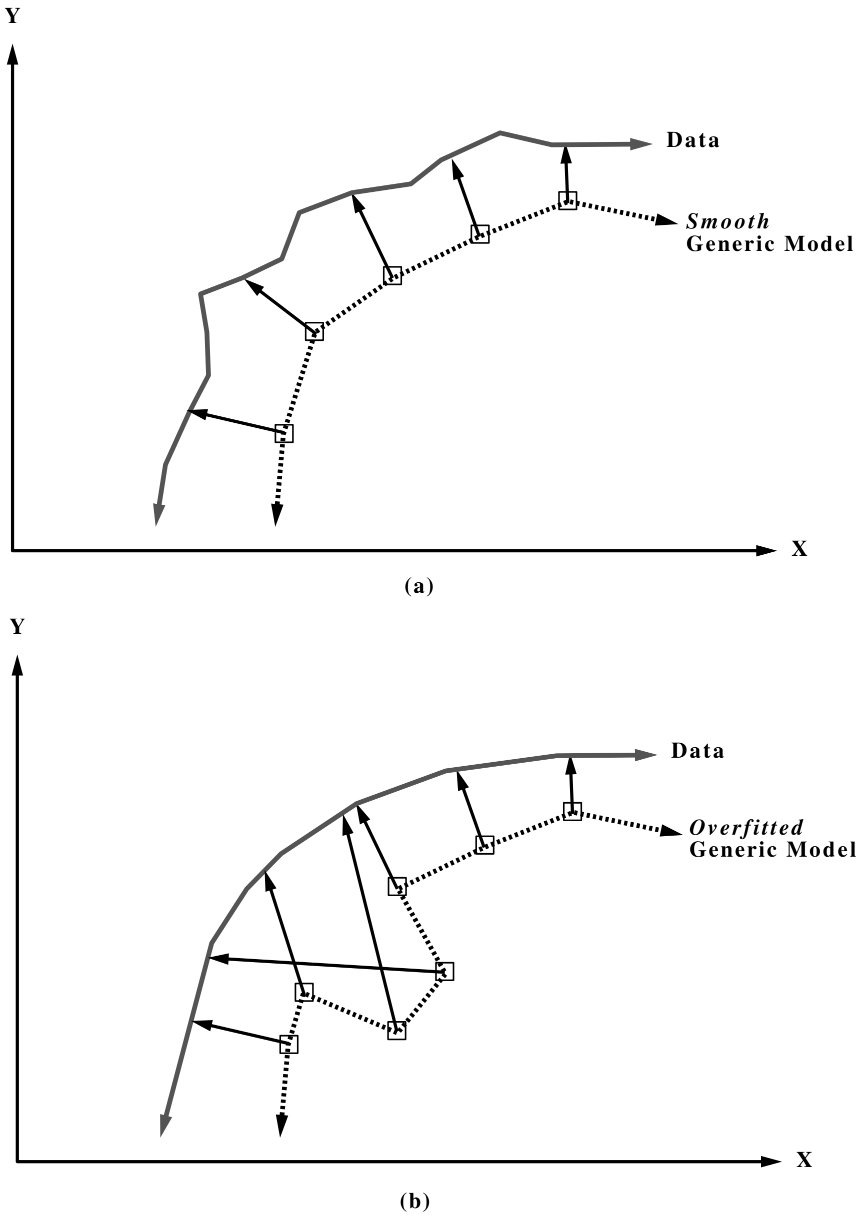


Figure 3.22: A model that is smoother than the input data can be more easily matched to the input data by adjusting the vertex locations along the direction of the local surface normal (a). A model that is overfitted is more difficult to match to the data using this method (b).

$$\mathbf{v}_n' = (1-\alpha)\mathbf{v}_n + \alpha \sum_{i \in adj(\mathbf{v}_n)} \mathbf{v}_i, \quad 0 \leq \alpha \leq 1, \tag{3.7}$$

where $\mathbf{v}_n$ is the original vertex location, $\mathbf{v}_n'$ is its new location, α is the smoothness factor, and $adj(\mathbf{v}_n)$ is the set of all vertices adjacent to $\mathbf{v}_n$ (i.e., connected by a single edge).

The actual degree of smoothness required by the generic model is a factor that is explored empirically in Chapter 4. Most of the preliminary matching experiments used a default decimation factor of 2048. This factor was chosen because the number of triangles in the resulting model (1224) was large enough to capture the essential structure of the vertebra without overfitting it, yet it was small enough to permit computations to be performed in reasonable amounts of time.

3.5 Model Representation

In order to make use of the vertebral AFEM model, it is crucial to have a highly expressive yet efficient respresentation scheme. Because of the need to create a model that can be used for multiple purposes, a very general and highly extensible representation is needed to handle all the complexities of anatomical structures and the manipulations performed on them. One powerful method for building geometric models is to use *hierarchical trees* constructed from *symbolic expressions*. Hierarchical trees are useful in applications where a problem domain can be recursively decomposed into smaller subproblems (for example, Levitt's radiographic analysis program in Section 2.3.6). Symbolic expressions are useful for representing and manipulating heterogenous components in a dynamic fashion. Symbolic trees provide a convenient mechanism for representing, constructing, and operating on hierarchical models containing heterogenous components. Before delving into the details of the spine model, I will briefly describe the basic concepts of symbolic expressions and show how they can be used to represent geometric objects.

3.5.1 Symbolic representation

Symbolic expressions (also referred to as *s-expressions*) were formalized by McCarthy [56] when he introduced the LISP programming language more than 30 years ago. An s-expression is nothing more than an ordered set of *atoms* and *lists*. An atom is an object that is considered to be indivisible. The following are examples of valid atoms:

```
a
0
atom
"atom"
3.14159265
```

PI
nil

The following are not atoms:

hello world *(this is a compound object containing two atoms)*
A + B

A list is an ordered collection of atoms and lists enclosed within a balanced pair of parentheses. This definition is intentionally recursive because s-expressions were originally designed as a language for processing recursive functions of symbolic expressions. According to this definition, the following constitute valid lists:

(a b c)
(a)
()
(pi (are square) 2)
(this (is (a list (of lists))))
nil

Notice that *nil* is both an atom as well as a list. This is because nil is actually a synonym for the empty list (), thereby making the empty list an atom as well.

Although it may appear that there is some sort of sequential ordering to the elements of a list, there is, in fact, no such ordering. The ordering only arises as a result of performing symbolic operations on the lists. Strictly speaking then, it is not really valid to ask, What is the first element of the list (1 2 3)? However, it is sometimes useful in practice to think of a list as having a numerical order imposed on it. The nature of the ordering depends on the semantics of the list. For instance, if the list (1 2 3) represents an array of three integers, then it makes perfect sense to ask, What is the first element of this list? However, if the list (1 2 3) represents a graph with three nodes labeled 1, 2, and 3, it might make more sense to ask, What is the leftmost node? rather than, What is the first node?

The power of s-expressions arises from the fact that when atoms are used to represent functions and operations, what results is a complete, general-purpose programming system. In addition, there is no difference between a program and its data; they are both represented by s-expressions and can only be distinguished by their context. There are five primitive symbolic operators in LISP that can be combined recursively to form all computable functions:

1. The **atom** function, which returns true if and only if its sole argument is an atom[1]:

[1] Normally, LISP will try to evaluate an atom if it appears by itself or if it is the first atom within a list or sublist. The quote mark is used in front of atoms and lists when it is necessary to suppress this evaluation. For instance, the atom a may contain the value (a b) in which case (atom 'a) true but (atom a) false. Note that this is a feature of LISP and not an inherent property of s-expressions.

(atom 'a) true
(atom '(a b)) false

2. The **eq** function, which returns true if and only if its two arguments (which must be atoms) are the same:

(eq 'a 'a) true
(eq 'a 'b) false

3. The **car** function, which only accepts a list and returns its "first" element:

(car '(a b c)) a
(car '((a b) c)) (a b)

4. The **cdr** function, which returns the "complement" of car:

(cdr '(a b c)) (b c)
(cdr '((a b) c)) (c)

5. The **cons** function, which performs the "inverse" of car and cdr:

(cons a '(b c)) (a b c)

Although these five primitive functions are sufficient to derive all other computable functions, any practical LISP system will provide several hundred auxiliary functions in order to create a usable and convenient programming system (e.g., arithmetic operations $\{+, *, sin, cos, ...\}$ and control structures $\{if, while, loop, \ldots\}$ to name just two).

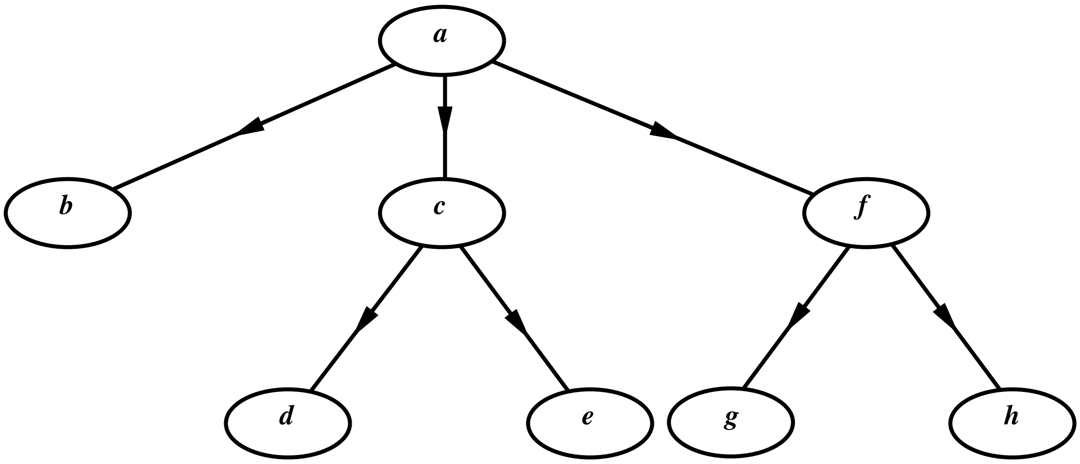


Figure 3.23: An example of a directed, acyclical tree with a single root node (labeled *a*.

3.5.2 Graphs and s-expressions

S-expressions provide a very simple and concise notation for representing graphs of any type — hierarchical, directed, undirected, cyclical, or acyclical. For example, the hierarchical, acyclical, directed graph shown in Figure 3.23 can be represented by the following s-expression:

$$G = (a\ (b)\ (c\ (d\ e))\ (f\ (g\ h))).$$

In this notation each list contains an atom representing some node, which is followed by a list of its children. This process is repeated recursively until a leaf node (i.e., a node with no children) is encountered. One could formalize this representation using a regular expression-style grammar (where * indicates zero or more occurrences of a given entity):

$$\begin{array}{lcl} \mathit{Tree} & \rightarrow & (\mathit{Parent}\ (\mathit{Children}^*)^*) \\ \mathit{Parent} & \rightarrow & \mathit{atom} \\ \mathit{Child} & \rightarrow & \mathit{Tree} \mid \mathit{atom} \mid \mathit{nil} \end{array}$$

Although this notation can represent all singly rooted trees, it is not powerful enough to represent more general directed graphs. A directed, *cyclical* graph (sometimes referred to as a *network*) does not have any notion of parent or child nodes, only nodes followed by successor nodes (see Figure 3.24). A better representation of this graph would be given by the following s-expression:

$$G = ((a\ (c\,f))\ (b\ (a))\ (c\ (d\ e))\ (d\ (b))\ (e\ nil)\ (f\ (g\ h))\ (g\ nil)\ (h\ (d))).$$

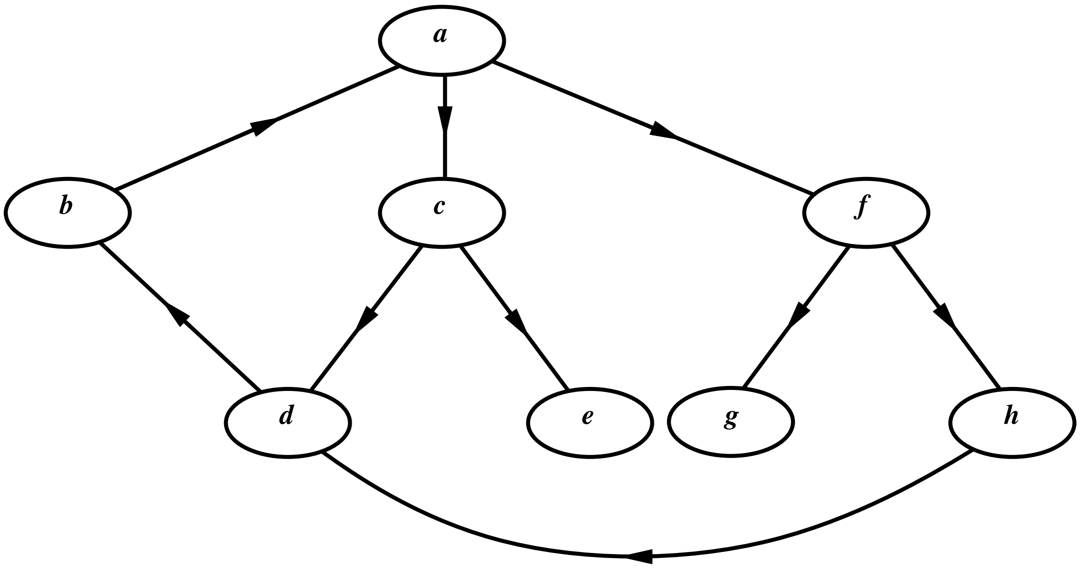


Figure 3.24: A network can be represented by a directed, cyclical graph. In this example, there is no apparent root node. In practice, the root node may be designated by the network designer or may be implied by the functional behavior of the network.

The formal grammar for a cyclical graph could be represented in the following manner:

$$
\begin{array}{lll}
\mathit{Graph} & \rightarrow & ((\mathit{Node}\ (\mathit{Successors}))^*) \\
\mathit{Successors} & \rightarrow & (\mathit{Node}^*) \\
\mathit{Node} & \rightarrow & \mathit{atom} \mid \mathit{nil}
\end{array}
$$

If, on the other hand, the nodes in this tree could be considered sequential elements in an array, then a much simpler notation could be used:

$$G = ((2)\ (2\ 3)\ (4)\ (5)\ (6)\ (1\ 3)).$$

The grammar for this list would look as follows:

$$
\begin{array}{lll}
\mathit{Graph} & \rightarrow & ((\mathit{Successors})^*) \\
\mathit{Successors} & \rightarrow & (\mathit{atoms}^*)
\end{array}
$$

The idea here is that the list contains only the successors of nodes that are implied by their position in the list. Hence, node 1 contains the successors indicated in the first sublist, node 2 contains the successors indicated in the second sublist, etc.

3.5.3 S-expressions and hierarchical geometric objects

Utilizing the preceding notions of s-expressions applied to graphs, a similar sort of representation can be developed for anatomical structures. The formal grammar for specifying a symbolic anatomical model is shown in Table 3.2. Although this grammar is intended to be a framework for building a complete anatomical model, there are a few application-dependent components that have been left unspecified at this time. These components will be defined by future applications as they are needed.

Rule 3.8 in Table 3.2 states that an anatomical *Model* is nothing more than a list of *Objects*. Objects, in turn, can be either a list of *Mesh* descriptors (Rule 3.9), a list of *Analytic* descriptions of surfaces and volumes (Rule 3.10), a list of *Landmark* descriptors (Rule 3.11), or a list of *Constraint* descriptors (Rule 3.12). Each object also has a unique *Id* and a user-defined list of parameters called *Udata*. Objects always have a tag field associated with them that identifies the type of the object (the syntax $\# < \ldots >$ indicates that the enclosed item is an atom and not a list). Rule 3.13 states that *Objects* can be preceded by an identifier called #<$Class$>, which indicates that the following descriptor is to be used as a template for replicating or referencing a class of objects. Hence, *Objects* are really classes that can be instantiated zero or more times using Rule 3.14. Rule 3.15 states that *Objects* can be recursively nested inside other objects as well as inside other models. In fact, because Rules 3.8 and 3.15 are identical, they imply that a *Model* is just a synonym for a *Object*. Hence, *Models* can also be composed hierarchically to form larger models. Having these two separate rules is merely for pedantic purposes (calling a model an object confuses the role of the model at the highest level).

Each object can have its own $Xform$ descriptor, a homogeneous transformation matrix that specifies the position, scaling, and orientation of that object relative to its parent object (Rule 3.16). The homogeneous transformation matrix is a 4 x 4 matrix that transform the vector $V(x, y, z, w)$ corresponding to the point $P(x/w, y/w, z/w)$ in $\Re^3$:

$$Xform = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.33}$$

The last column in Equation 3.33 translates the point $P(X, Y, Z)$ to a new location P' in the following manner:

$$P' = \begin{bmatrix} X + t_x \\ Y + t_y \\ Z + t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{3.34}$$

The square 3 x 3 submatrix in the upper left part of Equation 3.33 will scale, rotate, and shear a set of points using the following values:

$$Model \rightarrow ((Object)*) \tag{3.8}$$

$$Object \rightarrow (\#<Tag>\ Xform\ ((Mesh)*)\ Udata) \tag{3.9}$$
$$Object \rightarrow (\#<Tag>\ Xform\ ((Analytic)*)\ Udata) \tag{3.10}$$
$$Object \rightarrow (\#<Tag>\ Xform\ ((Landmark)*)\ Udata) \tag{3.11}$$
$$Object \rightarrow (\#<Tag>\ ((Constraint)*)\ Udata) \tag{3.12}$$
$$Object \rightarrow (\#<Class>\ \#<ClassName>\ Object) \tag{3.13}$$
$$Object \rightarrow (\#<Class>\ \#<ClassName>) \tag{3.14}$$
$$Object \rightarrow (\{Object\}*) \tag{3.15}$$

$$Xform \rightarrow \#<4 \times 4\ array> \mid nil \tag{3.16}$$

$$Constraint \rightarrow (\#<ConstraintOp>\ (\{Udata|Object\}*)) \tag{3.17}$$

$$Landmark \rightarrow Point \tag{3.18}$$
$$Landmark \rightarrow Curve \tag{3.19}$$
$$Landmark \rightarrow Surface \tag{3.20}$$
$$Landmark \rightarrow Volume \tag{3.21}$$

$$Point \rightarrow ((\#<Tag>\ Xform\ \#<vector>\ \#<MeshPtr>\ Udata) \tag{3.22}$$
$$Curve \rightarrow ((\#<Tag>\ Xform\ \#<array>\ \#<MeshPtr>\ Udata) \tag{3.23}$$
$$Surface \rightarrow ((\#<Tag>\ Xform\ \#<array>\ \#<MeshPtr>\ Udata) \mid (Object) \tag{3.24}$$
$$Volume \rightarrow ((\#<Tag>\ Xform\ \#<array>\ \#<MeshPtr>\ Udata) \mid (Object) \tag{3.25}$$

$$Mesh \rightarrow ((\#<Tag>\ Xform\ Vertices\ Edges\ Faces\ Regions\ PList\ Udata) \tag{3.26}$$
$$Vertices \rightarrow \#<N_{pe} \times N_v\ array> \tag{3.27}$$
$$Edges \rightarrow (\{\#<vector>\}*) \mid \#<2 \times N_e\ array> \tag{3.28}$$
$$Faces \rightarrow (\{\#<vector>\}*) \mid \#<N_{fe} \times N_f\ array> \tag{3.29}$$
$$Regions \rightarrow (\{\#<vector>\}*) \mid \#<N_{rf} \times N_r\ array> \tag{3.30}$$
$$PList \rightarrow (\{Udata\}*) \tag{3.31}$$

$$Udata \rightarrow (\#<Id>\ sexp) \mid nil \tag{3.32}$$

Table 3.2: The formal grammar specification for representing anatomical models using the s-expression syntax.

$$P' = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \rightarrow Scaling. \tag{3.35}$$

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \rightarrow Rotation\ about\ X{-}axis\ by\ angle\ \alpha. \tag{3.36}$$

$$P' = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \rightarrow Rotation\ about\ Y{-}axis\ by\ angle\ \beta. \tag{3.37}$$

$$P' = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \rightarrow Rotation\ about\ Z{-}axis\ by\ angle\ \gamma. \tag{3.38}$$

The matrix A will perform a rotation if and only if it is orthonormal and has a determinant of one. Otherwise, anisotropic non-rigid shearing will occur if the matrix A is applied to a collection of points describing an object.

A *Constraint* object can be used to represent a set of geometric, arithmetic, logical, or even rule-based constraints among various objects. To allow for such a wide variety of constraints, the form of Rule 3.17 is as general as possible: a constraint operator followed by a list of object operands (which, in turn, can also include constraints). Because the constraints are s-expressions, they can be either *declarative* or *procedural*. A declarative constraint merely *states* that a constraint must hold true, whereas a procedural constraint specifies *how* that constraint must be enforced. Examples of declarative constraints include *positional equality* (all the operand objects must have the same (x, y, z) location), *positional range* (all the operand objects must be located within a given distance of each other), or *implicit functional relations* of the form $f(obj_1, obj_2, \ldots, obj_n)\ \{= \mid < \mid >\}\ 0$. Procedural constraints include the general class of *assignment* operations ($f(obj_{lhs}) \leftarrow g(obj_1, obj_2, \ldots, obj_n)$). The functions f and g are general s-expressions that represent mathematical or logical operators. The constraint syntax, while very general, may be difficult to implement for all but the simplest constraint types. Furthermore, this generality may admit constraints that are NP-hard or even infeasible to achieve.

Landmark objects are used to augment the model to provide information that is useful for the segmentation process, and are not part of the *geometric* description of a model. A *Landmark* object can consist of *Points, Curves, Surfaces*, or *Volumes* (Rules 3.18–3.21). All landmarks include an *Id* tag, a *Xform* matrix, and a *Udata* field. A *Point* is the simplest landmark and includes an (x, y, z) location vector (Rule 3.18). A *Curve* consists of an array of points that are interpreted as control points for either a piecewise linear curve or a variety of spline curves (Rule 3.19). *Surfaces* and *Volumes* can be defined as an extension of a curve with more than one control parameter, or they can be defined in terms of another geometric object (Rules 3.20–3.21). Defining a landmark as a geometric object allows the landmarks to be virtual objects that can be defined using the same syntax as other objects in the model. Although they have different tag values, the syntactic forms for surfaces and volumes are identical.

One of the most important objects in the model is the *Mesh*. This object is used to define either surface or volume regions in $\Re^3$. In fact, there is no syntactic difference between a surface mesh and a volume mesh. A mesh begins with a collection of *Vertices* represented by an array with dimensions $3 \times N_v$, where N_v is the total number of vertices in the mesh (Rule 3.27). Each column in the array represents a unique *vertex* (a point (x, y, z) in $\Re^3$) of the mesh. A collection of vertices may be sufficient for describing the points on the mesh surface or mesh region, but it is not sufficient for describing the topology of the mesh. The topology can be described by defining the *Edges* of the mesh (Rule 3.28). An edge is defined as a piecewise linear curve connecting at least two vertex point in a given order. Edges usually correspond to boundaries of polygonal elements (for example, the three sides of a triangle). Edges can be defined either as a list of vertex point indices or as an array of vertex point indices. The list form is recommended if the number of vertices defining an edge is indeterminate. However, if the number of vertices in an edge (N_{pe}) is fixed for a given mesh (usually $N_{pe} = 2$), then the array form is recommended. A mesh can also have a *Faces* descriptor, which specifies how the edges are connected to form face planes (Rule 3.29). Similarly, a *Regions* descriptor specifies how the *Faces* are connected (Rule 3.30) to form solid regions. Finally, each mesh can have an optional *PList*, a set of property descriptors (Rule 3.31), and user-defined data elements (*Udata*, Rule 3.32). Most three-dimensional modeling systems share the notion of surface meshes that are described by a collection of vertices, edges, and faces. The anatomical modeling system described here extends this collection to include regions and property lists as well as user-defined data descriptors. These extensions are useful for performing model-based image analysis and biomechanical simulation.

3.5.4 Benefits of symbolic modeling

One of the advantages of using s-expressions is that new elements can be transparently added to the model description without affecting programs that rely on the original structures. Programs refer to the model elements symbolically as opposed to using a static or positional reference. As a result, data structures do not have to be strongly typed. This makes it easier to generate data for different programs that share some common data elements but have not been explicitly written to share data.

Graphs and s-expressions are really different manifestations of the same entities. Whereas graphs are visual, they nonetheless possess dual representations in the textual s-expression alphabet. Languages such as LISP, which are based entirely on s-expressions, are ideal for writing applications that perform extensive manipulations on graph-based data objects. Unfortunately, most LISP implementions lack the speed and efficiency required to handle scientific programming tasks. Furthermore, most scientific programmers have expressed unwillingness and/or discomfort with programming in languages such as LISP. FORTRAN and C, in that order, are the two most commonly preferred languages for writing scientific programs. In order to provide the benefits of dynamic s-expressions for static programming environments such as FORTRAN, C, and C++, I have implemented a dynamic *model definition language* (MDL), which allows programmers to create and manipulate s-expressions using static procedural calls (currently the language has only been implemented in C++, although other implementations are feasible). The MDL language is actually implemented as a run-time library package, and hence does not require any additional compilation or external resources (see Appendices B and C for a complete specification of the language and library). Users can either create s-expressions dynamically or read them in from external files. S-expressions can be manipulated using library calls, passed as function arguments, and saved to external files if needed.

The MDL library allows scientific applications (or any program that makes extensive use of both graph-based objects as well as traditional numerical arrays) to implement high-level dynamic data objects using s-expressions. Code that is computationally intensive can be written in a static language such as C or FORTRAN and make calls to the MDL library to manipulate the s-expressions. Such a program design strategy can preserve the benefits of static compilation (namely, faster execution speed) while retaining the power and flexibility needed to manipulate complex data structures.

3.6 Surface Matching

Once a coarse match of the generic model to the input data has been achieved, each vertex of the FEM must then be adjusted so that it lies on the surface of the vertebra. This is done by searching along the direction of the surface normal near the vertex (in both the positive and negative directions) until a voxel is found that matches the bone threshold value. More precisely, the image intensity value will cross the threshold value somewhere within the voxel (the intensity values are trilinearly interpolated with the voxel). The exact location is found by applying a quadratic line search root-finding algorithm on the following expression:

$$\begin{aligned} f(u) = I'(u \cdot \mathbf{n}_v + \mathbf{v}) - T_b \\ u_{min} < u < 0 \\ 0 < u < u_{max}, \end{aligned}$$

where I' is the interpolated intensity value at location (x, y, z), $\mathbf{n}_v$ is the vertex normal for vertex $\mathbf{v}$, and T_b is the bone threshold value. In order to deal with the possibility of multiple roots within the search bracket, the limits u_{min} and u_{max} are initially set to one pixel unit and increased in a Fibonacci sequence until a valid root is found, or until they reach a maximum allowable value for the given vertex (the default

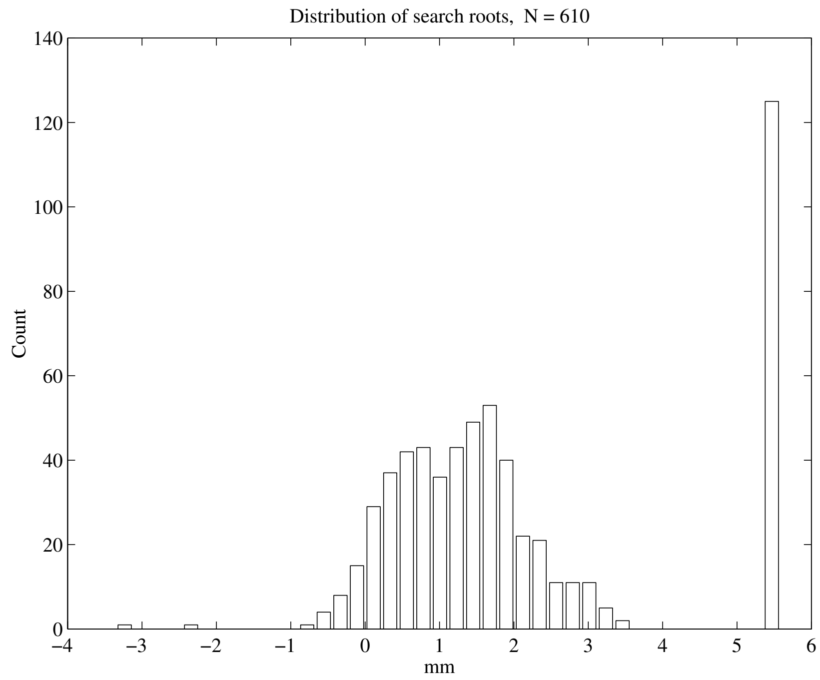


Figure 3.25: Distribution of search roots for vertices from a typical C4 vertebra. The large bin near 5.5 mm contains all outlier roots. Roughly 80 percent of the roots lie underneath the normal distribution and 20 percent are outliers.

maximum value is a tunable parameter in the software implementation). Figure 3.25 shows the frequency distributions of the roots for a sample C4 vertebra. If a root is found in both the positive and negative search directions, both roots are saved for further analysis, as described in Section 3.6.1.

Five distinct scenarios can occur in the process of matching the individual vertices in the model to the local surfaces of the data. These five cases are illustrated in Figure 3.26. The best-case scenario occurs when a vertex matches a nearby surface and the surface normals on both the model and the data are consistent (depicted by p_1). This point is considered to be a *good match*. However, if the distance between the matching surface and model vertices exceeds a predetermined limit L_T (depicted by p_2), then regardless of whether the surfaces are consistent or not, the vertex is considered to be a *bad match*. A corollary to this case occurs when there is no surface that can be found along the search direction (depicted by p_3). The more interesting cases occur when the search distance falls within the search limit tolerance L_T but the surface normals are not consistent. If the deviation of the surface normals exceeds a predetermined angular limit ω_T, then the vertex is considered to be a bad match (depicted by p_4). Otherwise, if the angular deviation is within the tolerance limit ω_T, the vertex is considered to be a good match (depicted by p_5).

The cumulative angular deviation between the surface normals of the model vertices and their respective matching data normals provides an indication of the quality of the overall match. More precisely, I

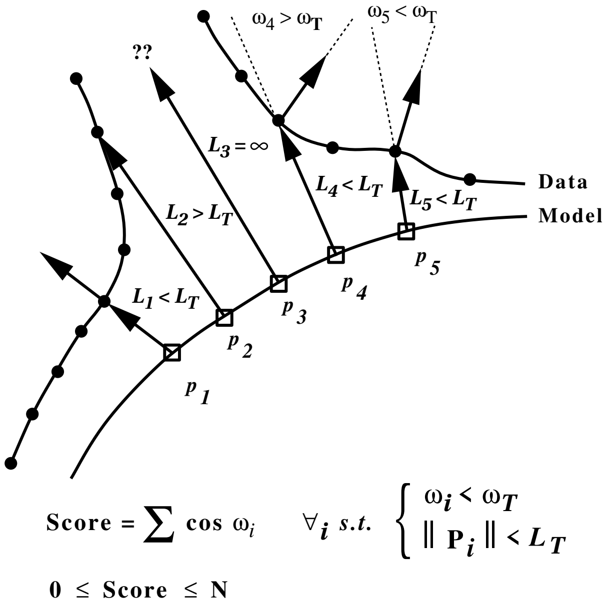


Figure 3.26: The five vertex matching scenarios are depicted graphically above. Only points p_1 and p_5 are considered good matches because (1) the surface normals are coherent with respect to the model and the data, and (2) the distances between the model points and the matching points on the data are less than L_T.

define the *matching score* to be the sum of the cosines of the deviations of these normals,

$$Score = \sum_i^N \cos \omega_i = \sum_i^N \mathbf{n}_i \cdot \mathbf{s}_i, \ \ \forall\, i \ \ s.t. \begin{cases} \omega_i < \omega_T \\ L_i < L_T \end{cases}, \tag{3.39}$$

where $\mathbf{n}_i$ is the normal of the model vertex v_i, $\mathbf{s}_i$ is the corresponding normal at the data surface, and L_i is the distance between the matching model vertex and data surface point. A model that matches its data perfectly would have a score of N (the total number of vertices), while a completely mismatched model would have a score of zero. In theory, the matching score could be defined using the actual angles ω_i, but this would involve unnecessary computations of the inverse cosine.

3.6.1 Consistency of normals

In addition to matching the surface threshold value, the image gradient at the matching voxel must be consistent with the surface normal of the model. If the normals are not consistent, it is quite likely that the surface that is being matched is invalid. For example, in Figure 3.27, the surface normal near the disk surface at a given vertex v_m may intersect the vertebral surface at three different points: two on the current vertebra (s_1 and s_3) and one on the adjacent vertebra (s_2). Only voxel s_1 will have an image gradient that is consistent with the vertex normal of the model (corresponding to the correct surface).

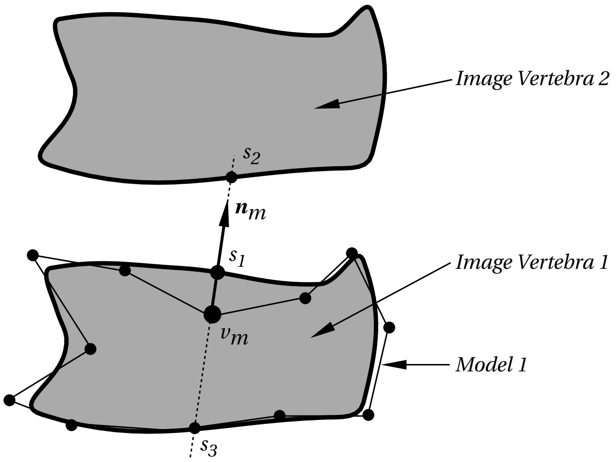


Figure 3.27: A surface match is achieved when a vertex on model, v_m, is adjusted along the positive and negative directions of the model surface normal, n_m, so that it lies on a voxel that is tessellated by a bone surface. In this figure, there are three voxel locations that satisfy this criterion, s_1, s_2, and s_3. Because the model surface normal and the image gradient at the surface must be consistent, only s_1 is chosen as the correct matching voxel.

The *vertex normal* of vertex v_i is defined as the average of the normals of all the triangles that contain v_i. Unfortunately, a triangle can have two normals, depending on the order in which the vector cross products are computed. These two normals, n_+ and n_-, can be defined in terms of its vertices a, b, and c and the vectors connecting them $(v_{ab}, v_{ba}, v_{ac}, v_{ca}, v_{bc}, v_{cb})$. For example,

$$n_+ = v_{ab} \times v_{bc} \tag{3.40}$$

$$n_- = v_{cb} \times v_{ba}, \tag{3.41}$$

and

$$n_+ = -n_-.$$

Because the tessellation algorithm is not guaranteed to produced consistently oriented normals, it is necessary to order the vertices of each triangle in such a way that all adjacent triangles have consistent normals.

Definition 3.1 *A pair of triangles is considered to have consistent normals when the common edge connecting the two triangles has a net flux of zero.*

Definition 3.2 *The net flux of an edge v_{ab} connecting vertices a and b is the sum of the corresponding adjacent vectors used to compute the normals of the two triangles containing v_{ab}.*

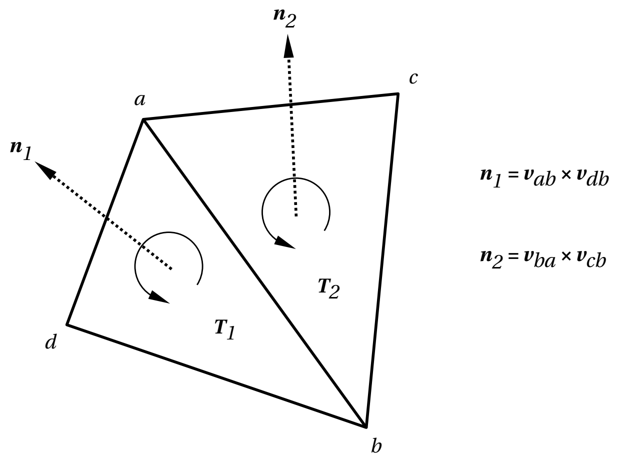


Figure 3.28: The edge v_{ab} connecting triangles T_1 and T_2 will have a net flux of zero if the vectors used to compute the normals n_1 and n_2 contain v_{ab} and v_{ba}, respectively.

An example of an edge with a net flux of zero is shown in Figure 3.28. The flux of edge v_{ab} is the sum of vectors v_{ab} and v_{ba}, which are used to compute the two normals n_1 and n_2, respectively.

To guarantee a model with consistently oriented normals, I have developed an algorithm for ordering the vertices of each triangle so that every edge has a net flux of zero. The algorithm starts with an artibrary edge v_{ab} and computes the normal for triangle ABC which contains vertices (a, b, c) using the adjacent edge v_{bc}. It then visits triangle ABD adjacent to $v_a b$ containing the vertices (a, b, d) and computes the normal using v_{ba} and v_{ad}. Once the direction of any edge in a triangle has been defined, the ordering of the vertices can then be fixed by visiting the vertices in a circular path. That is, if we start with edge v_{ab}, then the ordering of the vertices of triangle ABC is (a, b, c). In order for the flux of vertex v_{ab} to be zero, the ordering of the vertices of triangle ABD would be (b, a, d), as shown in Figure 3.28. The last step of the algorithm is to recursively visit the triangles adjacent to the remaining two edges of the initial triangle. The algorithm quits when the flux terms of all edges have been computed. The normals of the resulting mesh will then be consistent across all triangles.

It is also desirable to have the triangle normals point outward with respect to the object boundary. The algorithm described does not guarantee this scenario. Instead, it guarantees that *all* the normals will be pointing either outward or inward. Fortunately, the sense of the normals can be fixed by simply negating the x, y, and z components of each normal. This operation can be automated, if needed, by making sure that the normals point in the direction opposite to the surface gradient at every triangle. This may not hold true in regions where the gradient is ill-defined (due to non-smooth object regions). As a result, the gradient test should be averaged over all the triangles to see which orientation yields better results.

3.6.2 Consistency of curvatures

Another useful consistency check is to ensure that the curvatures of the matching regions are consistent. If the curvature of a surface region in the data is considerably higher or lower than the curvature in the model, it is likely that there is a mismatch. The curvature at a given vertex v, $\omega_C(v)$, can be computed by averaging the absolute differences between the angles of the vertex normal $\mathbf{n}_v$ and the normals of all the triangles adjacent to the vertex,

$$\omega_C(v) = \frac{1}{N}\sum_{i=1}^{N} |\arccos(\mathbf{n}_v \cdot \mathbf{n}_i)| \ , \ \ \forall_i \in adj\{v\}, \tag{3.42}$$

where N is the number of triangles in the adjacency set ($adj\{v\}$) of vertex v. Because variations in curvature can be rather high, the upper limit for ω_C is set to a very high value (approximately one radian).

Typically, when the curvatures are inconsistent, there is a high probability that neighboring vertices are also poorly matched even though they may fall within the ω_C curvature limit. To deal with this problem, I define a *contagion depth* which specifies how many neighboring vertices are classified as bad matches when they are adjacent to, or near other vertices that fall outside the curvature limit. A contagion depth of n means that all vertices that can be reached from a bad vertex by n or fewer edges will also be classified as bad matches. This is analogous to a morphological dilation operation where the boundaries of a given region are expanded by a fixed number of pixels. The contagion depth allows the curvature limit to be set to a high value to minimize the number of false positive outlier surfaces. At the same time, it catches a greater percentage of true positive outliers than might otherwise be detected with such a high limit.

3.6.3 Unmatched vertices

Typically, after the initial surface matching is performed, roughly 20 percent of the vertices remain unmatched. This could be due to the fact that there is no surface to match (e.g., the boundary is missing, as is the case with facet joints), or the surface is outside the allowable limits (and is presumed to be invalid). For these unmatched vertices, a spring-based relaxation technique is used to approximate the most likely position of the surface. The relaxation technique works by creating a spring force at each triangle edge. That is, every vertex is coupled to each of its neighboring vertices via two types of springs. The first type is a simple *linear* spring whose force of repulsion or attraction $F_l(n)$ on vertex v_n is governed by the linear spring law,

$$F_l(n) = -k_l(l - l_0), \tag{3.43}$$

where k_l is the *spring constant* and is a measure of how stiff the spring is. The stiffer the spring is, the more force it exerts on its attachment points. The parameter l_0 is the *rest length* and indicates the length of the spring when there are no external forces acting on it. The variable l is the actual length of the spring. The negative sign indicates that when the actual spring length exceeds its rest length, there will be a force of contraction that will tend to shorten the spring and bring it back to its rest length. Conversely, if the spring in compressed, the force will tend to expand the spring toward its rest length. The vector form of the linear spring equation can be written as

$$\mathbf{F}_l(n,m) = -k_l(\mathbf{v}_n - \mathbf{v}_m), \tag{3.44}$$

where $\mathbf{F}(n,m)$ is the vector force exerted by spring $S_{n,m}$ on vertices $\mathbf{v}_n$ and $\mathbf{v}_m$. This force is exerted on the two vertices in opposite directions. The total force felt by vertex $\mathbf{v}_n$ can be found by summing up all the forces exerted by all the springs connected to $\mathbf{v}_n$,

$$\mathbf{F}_l(n) = -k_l \sum_{m=1}^{m_n} (\mathbf{v}_n - \mathbf{v}_m), \tag{3.45}$$

where m_n is the total number of springs (typically equal to the number of edges) connected to $\mathbf{v}_n$. The linear spring constant need not be fixed for all vertices. For example, $k_{l_{n,m}}$ may be larger for n,m in the set of known good triangle edges. As a result, the distribution of $k_l(n,m)$ can be adjusted so that certain regions are more rigid than others. The variation in rigidity of the spring model can be specified empirically from statistical studies of vertebrae. However, for the purposes of this dissertation, the spring constants are held fixed over all the vertices.

The second type of force is exerted by *angular springs* whose forces are proportional to the angular deviation of vertex normals from their rest positions, as defined by the generic model. The force $\mathbf{F}_n$ exterted on vertex $\mathbf{v}_n$ of an angular spring is governed by the equation

$$\mathbf{F}(n,m) = k_\omega \mathbf{n}_{n0}(\omega_{nm} - \omega_{nm0}), \tag{3.46}$$

where k_ω is the angular spring constant and is a measure of the spring stiffness, and $\mathbf{n}_{n0}$ is the vertex normal associated with vertex $\mathbf{v}_n$ *prior* to the initial surface match. The springs defined by Equation 3.46 are a mathematical abstraction and do not exist in the physical world. The effect of these springs is to adjust the position of $\mathbf{v}_n$ along the direction of its *original* vertex normal such that the resulting angles among its neighboring vertices after the surface match approximate the angles prior to the surface match. Hence, the angular springs tend to create a coherent orientation of vertex normals by utilizing information from the generic model. The surface comprised of a set of unmatched vertices will tend to have the approximate shape of the surface in the original generic model.

As with the linear spring constant, k_ω need not be constant with respect to the vertex number. The parameter ω_{nm0} is the angle between $\mathbf{n}_{n0}$ and the ith vertex adjacent to $\mathbf{v}_n$ *prior* to the initial surface match, and ω_{nm} is the angle *after* the surface match. These angles can be computed as

$$\omega_{nm} = \arccos\left(\frac{(\mathbf{v}_i - \mathbf{v}_n)}{\|\mathbf{v}_i - \mathbf{v}_n\|} \cdot \mathbf{n}_{n0}\right), \tag{3.47}$$

and

$$\omega_{nm0} = \arccos\left(\frac{(\mathbf{v}_{i0} - \mathbf{v}_{n0})}{\|\mathbf{v}_{i0} - \mathbf{v}_{n0}\|} \cdot \mathbf{n}_{n0}\right), \tag{3.48}$$

where $\mathbf{v}_{n0}$ is the position of the vertex $\mathbf{v}_n$ *prior* to the initial surface match and $\mathbf{v}_n$ is the position *after* the match. Similarly, $\mathbf{v}_{i0}$ is the position of the ith vertex adjacent to $\mathbf{v}_n$ *prior* to the initial surface match, and $\mathbf{v}_i$ is the position *after* the match. Likewise, the total angular force exerted on a vertex $\mathbf{v}_n$ is given by

$$\mathbf{F}_\omega(n) = k_\omega \sum_{i=1}^{m_n} \mathbf{n}_{n0}(\omega_{nm} - \omega_{nm0}), \tag{3.49}$$

where again, m_n is the total number of angular springs (typically equal to the number of edges) connected to $\mathbf{v}_n$.

Given these two types of spring forces, the model can be viewed as a dynamic system having a potential energy state that is stored within all the springs. The system will then try to reach a minimal energy state (not necessarily a zero energy state) by converting the potential energy into kinetic energy, and thereby adjusting the points to more accurate locations. The total potential energy is defined as

$$E_{total} = \alpha \sum_{n=1}^{N} \|\mathbf{F}_l(n)\|^2 + \beta \sum_{n=1}^{N} \|\mathbf{F}_\omega(n)\|^2, \tag{3.50}$$

where α and β serve as both energy conversion factors as well as weighting factors for the two types of springs. The best match between the model and data can be determined by minimizing E_{total} subject to certain constraints. The primary constraint is that the vertices that found good matches are taken to be fixed. However, the vertices that were unmatched are allowed to move within a certain interval. Indeed, it is the positions of these unmatched vertices that is ambiguous.

The formulation of E_{total} in Equation 3.50 bears some resemblance to the two-dimensional snake model proposed by Kass, with several key exceptions. First, the snake model does not incorporate the notion of rest lengths for individual vertices. In fact, the original formulation by Kass yields snakes that tend to shrink toward zero length. Second, the snake model tries to minimize curvature extremas in the global sense. Hence, the Kass snake tries to find a *smooth* boundary for an object, but not necessarily the most accurate. The snake does not make extensive use of *a priori* knowledge regarding the geometry of the object it is detecting (except through the explicit use of volcanos). In contrast, the vertebral spring model developed here uses local curvature and distance information from the vertex coordinates in the generic FEM description. Finally, the vertices in the snake are not constrained in their search for image features. They are free to find the shortest path to the image feature for which they are searching (as prescribed by the gradient of the global energy function). In constrast, I have constrained the search direction to lie along the direction of the vertex normals during the initial surface-matching step. During the relaxation step, the spring forces tend to move the vertex positions toward their relative positions specified *a priori* by the FEM.

To demonstrate the feasibility of these extraction steps, I performed a match of the FEM shown in Figure 3.18 (decimated by a factor of 2048) to a new patient data set. The match of the landmarks yielded the following transformation matrix:

$$A = \begin{bmatrix} 0.9828 & -0.0031 & -0.2322 & 74.8316 \\ 0.0088 & 0.9894 & 0.0974 & 70.4927 \\ 0.0022 & -0.1268 & 0.8429 & 176.9274 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix}. \tag{3.51}$$

Note that the rotation submatrix in A is diagonally dominant with only three off-diagonals (a_{13}, a_{23} and a_{32}) making any modest contributions. Except for the translation component $[a_{13}a_{23}a_{33}]'$, the elements of the transformation matrix A indicate that the two vertebral elements to be matched are relatively congruent to one another.

After performing the initial surface match using a search range of ± 5 mm, 483 vertices found valid roots and 125 vertices went unmatched (see Figures 3.29a and 3.30a). After the relaxation step, the majority of the unmatched vertices settled to locations that were consistent with curvature of the generic model (see Figures 3.29b and 3.30b).

3.6.4 Energy minimization

The assumption behind the spring model I have formulated is that minimizing the energy of the dynamic system will produce a good fit of the generic model to the patient data. There are two techniques I have used to minimize the energy: (1) iterative relaxation and (2) non-linear optimization. The iterative relaxation technique assigns a unit mass to each vertex and allows it to move according to Newton's law,

$$\mathbf{a}_n(t) = \frac{\mathbf{F}(t)}{m_n} = \mathbf{F}_l(t) + \mathbf{F}_\omega(t), \tag{3.52}$$

where $\mathbf{a}_n(t)$ is the acceleration experienced by vertex $\mathbf{v}_n$ at any time t. Because the force at any given time is specified by Equations 3.44 and 3.46, the position of vertex $\mathbf{p}_n$ at any time t is given by

$$\mathbf{p}_n(t) = \int\int_0^t (\mathbf{F}_l(s) + \mathbf{F}_\omega(s))ds + t\mathbf{p}'_n(0) + \mathbf{p}_n(0), \tag{3.53}$$

where $\mathbf{p}'_n(0)$ and $\mathbf{p}_n(0)$ are the initial velocity (which is always taken to be zero) and position, respectively, of $\mathbf{p}_n$ at time t_0. Unfortunately, Equation 3.53 assumes a frictionless system with no dissipative forces. Such a system will never reach an equilibrium state, and hence, the solution will never converge. To fix this problem, a damping force $\mathbf{F}_d$ can be introduced whose magnitude is directly proportional to the velocity of a given vertex and whose direction is opposite to the velocity vector,

$$\mathbf{F}_d(t) = -k_d\mathbf{p}'(t), \tag{3.54}$$

where k_d is known as the *damping coefficient*. Substituting this term into Equation 3.52, we have

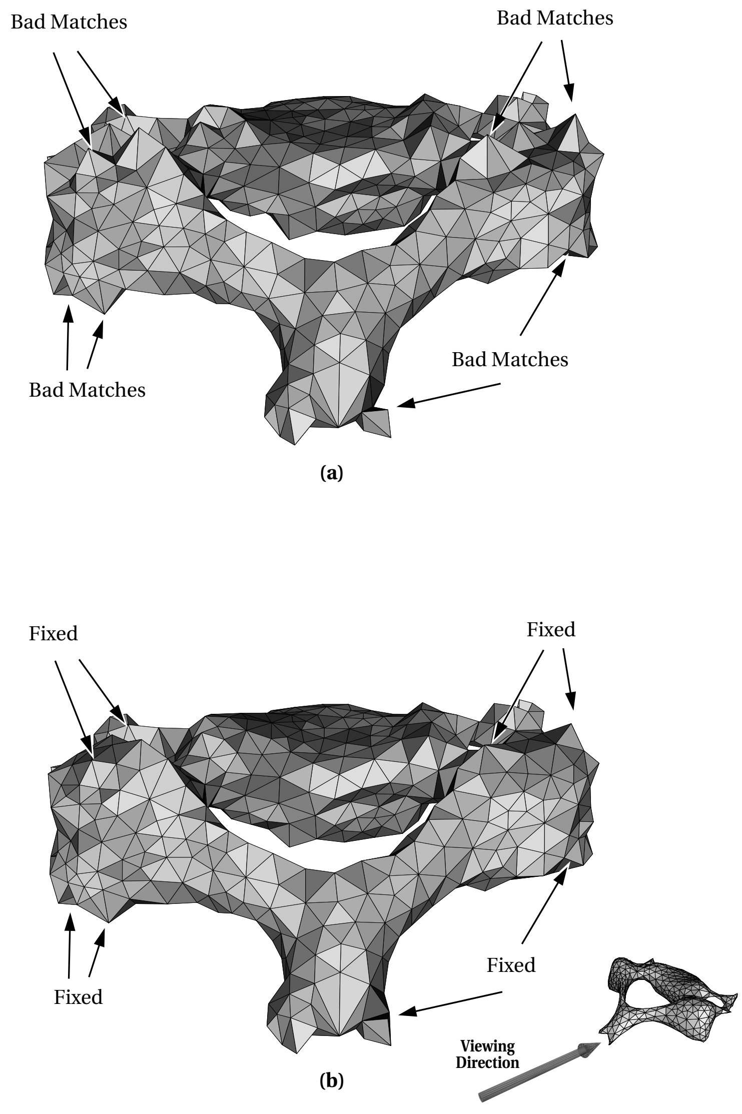


Figure 3.29: The C4 vertebra as seen from a posterior viewing position after the initial surface match (a) reveals many vertices that could not find valid surface points. After the relaxation technique (b), most of the unmatched vertices moved to locations that were more consistent with the original FEM.

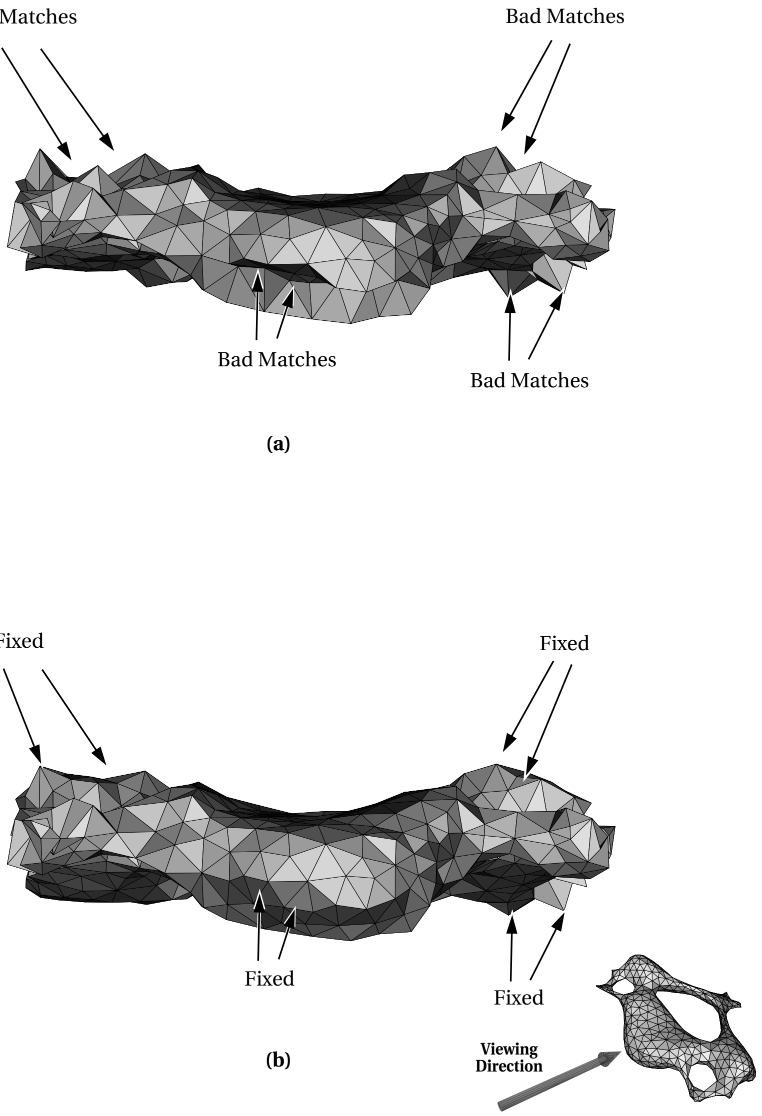


Figure 3.30: The C4 vertebra as seen from an anterior viewing position after the initial surface match (a) reveals many vertices that could not find valid surface points. After the relaxation technique (b), most of the unmatched vertices moved to locations that were more consistent with the original FEM.

$$\mathbf{a}_n(t) = \mathbf{F}_l(t) + \mathbf{F}_\omega(t) - k_d \mathbf{p}'_n(t), \tag{3.55}$$

or equivalently, by noting that $\mathbf{p}'_n(t) = \int \mathbf{a}_n(t)dt$, we have

$$\mathbf{a}_n(t) = \mathbf{F}_l(t) + \mathbf{F}_\omega(t) - k_d \int_0^t \mathbf{a}_n(s)ds. \tag{3.56}$$

This equation can be solved more easily by rewriting it as a second-order linear differential equation,

$$\mathbf{p}''_n(t) = \mathbf{F}_l(t) + \mathbf{F}_\omega(t) - k_d \mathbf{p}'_n(t), \tag{3.57}$$

whose solution is given by

$$\mathbf{p}'_n(t) = e^{-k_d t} \int_0^t e^{k_d s}(\mathbf{F}_l(s) + \mathbf{F}_\omega(s))ds. \tag{3.58}$$

Finally, the unknown vertex location $\mathbf{p}_n(t)$ can be found by integrating $\mathbf{p}'_n(t)$,

$$\mathbf{p}_n(t) = \int_0^t e^{-k_d u} \int_0^u e^{k_d s}(\mathbf{F}_l(s) + \mathbf{F}_\omega(s))dsdu + \mathbf{p}_n(0). \tag{3.59}$$

Equation 3.59 represents a closed-form solution of the vertex positions *if* the functions $\mathbf{F}_l(s)$ and $\mathbf{F}_\omega(s)$ can be fully described. However, these functions in turn depend on the values of $\mathbf{p}_n(t)$ for all $n \in N$. For this reason, it is more practical to use a finite difference approximation to find the solution to Equation 3.59. If the positions, velocities, and accelerations of all the vertices are stored in separate arrays, the solution to 3.59 can be computed in a very straightforward manner using an Euler approximation,

$$\begin{aligned} \mathbf{p}''_n(t_{i+1}) &\leftarrow \mathbf{F}_n(t_i) - k_d \mathbf{p}'_n(t_i) \\ \mathbf{p}'_n(t_{i+1}) &\leftarrow \triangle t \mathbf{p}''_n(t_i) + \mathbf{p}'_n(t_i) \\ \mathbf{p}_n(t_{i+i}) &\leftarrow \triangle t \mathbf{p}'_n(t_i) + \mathbf{p}_n(t_i), \end{aligned} \tag{3.60}$$

where $\mathbf{F}_n(t_i)$ is the combined linear and angular forces acting on $\mathbf{p}_n$ at time t_i. That is, $\mathbf{F}_n(t_i) = \mathbf{F}_{ln}(i \triangle t) + \mathbf{F}_{\omega n}(i \triangle t)$. The time step parameter $\triangle t$ is chosen to be small enough to yield an accurate solution, but large enough to converge within a reasonable amount of CPU time. Better results can be achieved by using Simpson's rule for integration instead of the Euler approximation. This would require twice as many evaluations of the acceleration vectors but half as many evaluations of the position vectors.

In addition to the time step $\triangle t$, another parameter that must be chosen carefully is the damping coefficient k_d. If k_d is too small, the system will oscillate excessively before reaching a stable solution. On the other hand, if k_d is too large, the system will take a very long time to stabilize. To illustrate, consider the simple one-dimensional spring attached to a fixed object on one end and a movable object of mass m on the other (see Figure 3.31). The equation describing the displacement of the object can be written as

$$x''(t) = \frac{F_l(t)}{m} - k_d x'(t) = \frac{k_s(L - x(t))}{m} - k_d x'(t), \tag{3.61}$$

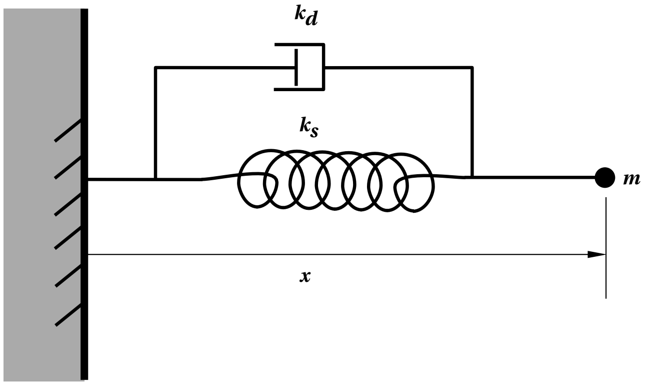


Figure 3.31: A simple one-dimensional damped spring system. An object with mass m is free to move along the x-axis subject to the spring tension and damping force.

where L is the rest length of the spring. The characteristic polynomial of Equation 3.61 can be written as

$$s^2 + k_d s + \frac{k_s}{m} = 0, \tag{3.62}$$

where s is the differential operator of the Laplace transform [27]. Critical damping occurs when the roots of Equation 3.61 are identical, or in other words, the discriminant of characteristic equation is zero,

$$b^2 = 4ac \quad \Rightarrow \quad k_d^2 = \frac{4k_s}{m} \quad \Rightarrow \quad k_d = 2\sqrt{\frac{k_s}{m}}. \tag{3.63}$$

For the unit mass case, critical damping occurs when $k_d = 2\sqrt{k_s}$. Figure 3.32 shows the behavior of $x(t)$ for $k_s = 1$ and various values of k_d. Note that the critical damping value is not a function of the rest length. Instead, it is only a function of k_d, k_s, and m. For the vertebral FEM, both k_s and m are fixed at unity, so only the value of k_d is relevant. Of course, the critically damped value of k_d is not necessarily the same for the multivariate case. However, the simplified analysis does provide a good starting point for selecting a reasonable value of k_d for the vertebral FEM.

3.6.5 Non-linear optimization

An alternative to the dynamic method of the previous section is to use a non-linear optimization algorithm (NPSOL) to minimize the total spring energy of the vertebral model. The problem can be defined in the following manner:

$$\begin{aligned} \text{minimize} \quad & E(\mathbf{v}) \\ \text{subject to} \quad & (\mathbf{v}_n - \mathbf{v}_{n0}) \times \mathbf{n}_{n0} = 0 \\ & \mathbf{v} \equiv \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}. \end{aligned} \tag{3.64}$$

The objective function E to be minimized is identical to the total energy expression previously described as

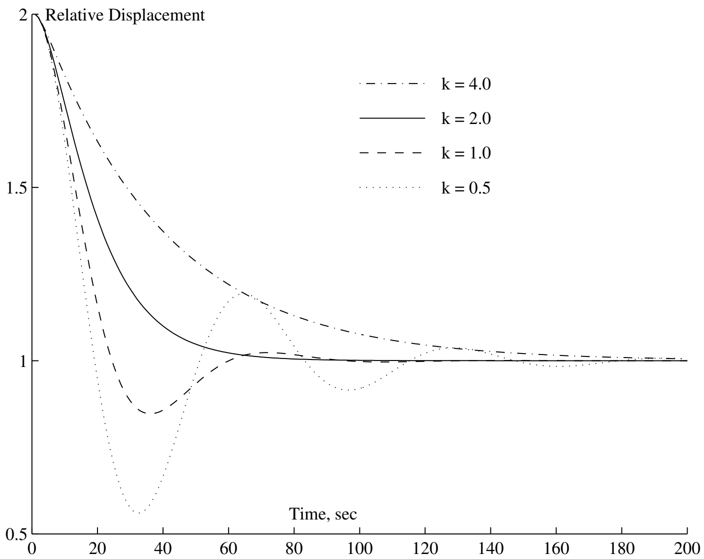


Figure 3.32: The time response of the one-dimensional spring system shown for various damping factors. For $k = 0.5$ and $k = 1.0$, the system is underdamped and oscillates toward the stable solution. For $k = 4.0$, the system is overdamped and requires a long time interval to reach the stable solution. For $k = 2.0$, the system is critically damped and arrives at the stable solution in the shortest amount of time.

$$E(\mathbf{v}) = \alpha \sum_{n=1}^{N} \|\mathbf{F}_l(n)\|^2 + \beta \sum_{n=1}^{N} \|\mathbf{F}_\omega(n)\|^2. \tag{3.65}$$

The vector $\mathbf{v}$ in Equation 3.64 aggregates all the individual vertices of the vertebral FEM $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}$ into a single state vector. However, only those vertices that are unmatched after the initial surface match step are considered to be independent variables. The rest are simply constants in the total energy function.

The constraints in Equation 3.64 are needed to limit the range of the search space and to ensure that the resulting surface has the same local curvature profile as the original model. Without these constraints, the free vertices could possibly move in directions that would destroy the topological consistency of the vertebral model (e.g., twisted or intersecting surfaces). The constraint basically states that the vertices be allowed to move only along the direction of their original normals ($\mathbf{n}_{n0}$). This reduces the likelihood of vertices crossing over into each other's territory.

Because the closed-form derivations of both the gradient of $E(\mathbf{v})$ and the Jacobian of the contraint matrix are quite onerous, it is more efficient to allow NPSOL to compute these quantities using its finite difference approximation. The resulting lack of accuracy of these estimated matrices can be compensated by specifying smaller step sizes in the quadratic search algorithm.

3.6.6 Verification

There are four important properties that the extraction method described thus far must satisfy in order to be considered valid. The first property is that it must be relatively insensitive to errors in the determination of the landmark positions. Second, the method should extract the same vertebral surfaces that the thresholding technique would achieve in those areas where the surfaces are unambiguous (that is, the isosurface areas are present and distinct). Third, in those areas of the data where the surfaces are ambiguous or missing, the method should use the information from the model to fill in or disambiguate the data. And four, the transition between the unambiguous data-inferred surfaces and the model-inferred surfaces should be as smooth as possible (that is, the two surfaces must be blended together with no apparent discontinuities).

To verify that the proposed kinematic solution achieves these goals, I generated several extracted patient-specific models from patient data sets, inspected them visually, and compared them with the known models which were generated by expert-guided manual segmentation. Figure 3.33 shows one of the patient-specific models after the coarse match and local deformations were performed. The ambiguous and/or missing surfaces (i.e., the bad matches) have been shaded in red color and come from the generic model whereas the green colored surfaces come from the data set. Note the discontinuities near the regions bordering the ambiguous/missing surfaces (for example, the facets and the vertebral endplates). After the kinematic system is allowed to relax to a steady-state energy configuration, the red and green regions merged smoothly together with no visible discontinuities, as shown in Figure 3.34. The total spring force of the system is plotted as a function of time in Figure 3.35. The kinematic solver is terminated when the change in the energy of the system falls below a predetermined tolerance limit.

These experiments provide a visual verification of all but the first of the four properties required for the extraction method. The only property which is not verified by this demonstration, the insensitivity to errors in the positions of the landmarks, will be verified as part of the quantitative evaluation study in Chapter 4.

3.6.7 Vertebra extraction

The final step of the segmentation method is the actual extraction of the bony regions from the images. This is performed by finding the intersections of the matched FEM surface triangles with each slice plane of the image. A surface triangle is said to intersect a given slice plane if at least one of its edges intersects the slice plane. Starting with an implicit definition of a slice plane P,

$$Ax + By + Cz + D = 0, \tag{3.66}$$

we then iterate over all the triangles in the FEM and find those edges that intersect Equation 3.66. For any edge $\mathbf{e}_i$ of triangle T with endpoints (x_1, y_1, z_1) and (x_2, y_2, z_2). This edge can be defined in terms of its two endpoints and a parameterization variable α as follows,

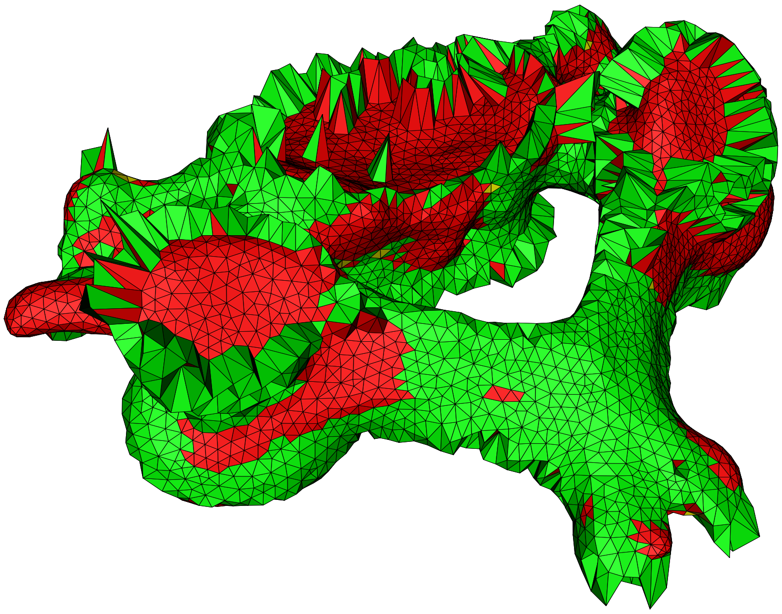


Figure 3.33: The configuration of the patient-specific model is shown here after the coarse match and local deformation steps have been performed and the matching score has been miminized. The red regions correspond to missing or ambiguous surfaces in the patient data set. The green regions correspond to areas where the data surfaces match the generic model.

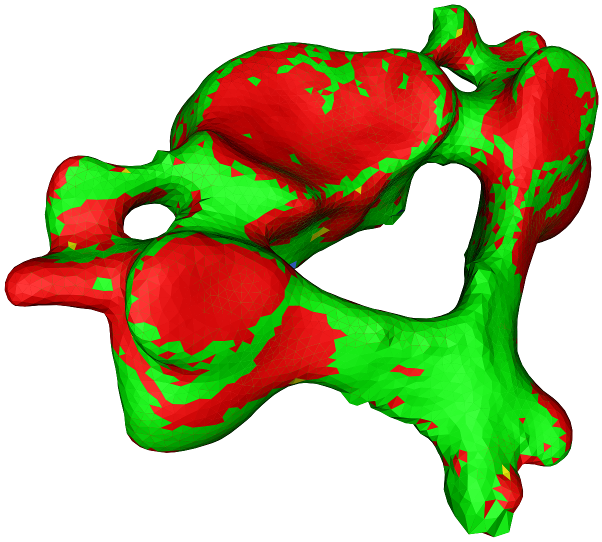


Figure 3.34: The configuration of the generic model is shown here after the relaxation process has reached its steady-state solution. The red regions corresponding to missing or ambiguous surfaces in the patient data set have been smoothly merged with the green regions corresponding to areas where the data surfaces match the generic model.

$$\mathbf{e}_i(\alpha) = \alpha \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \qquad 0 \le \alpha \le 1. \tag{3.67}$$

Then edge $\mathbf{e}_i$ intersects the plane P if there is a value of α within the interval $[0, 1]$ such that $\mathbf{e}_i(\alpha)$ lies on P. That is,

$$A(\alpha x_2 + (1-\alpha)x_1) + B(\alpha y_2 + (1-\alpha)y_1) + C(\alpha z_2 + (1-\alpha)z_1) + D = 0. \tag{3.68}$$

Solving this expression for α yields the following:

$$0 \le \frac{-(Ax_2 + By_2 + Cz_2 + D)}{A(x_1 - x_2) + B(y_1 - y_2) + C(z_1 - z_2)} \le 1. \tag{3.69}$$

Another way of deriving this condition is by noting that the two endpoints of e_i must not lie on the same side of P (unless one of them lies directly on the plane). Or in other words,

$$Ax_1 + By_1 + Cz_1 + D \ge 0 \implies Ax_2 + By_2 + Cz_2 + D \le 0, \tag{3.70}$$

and

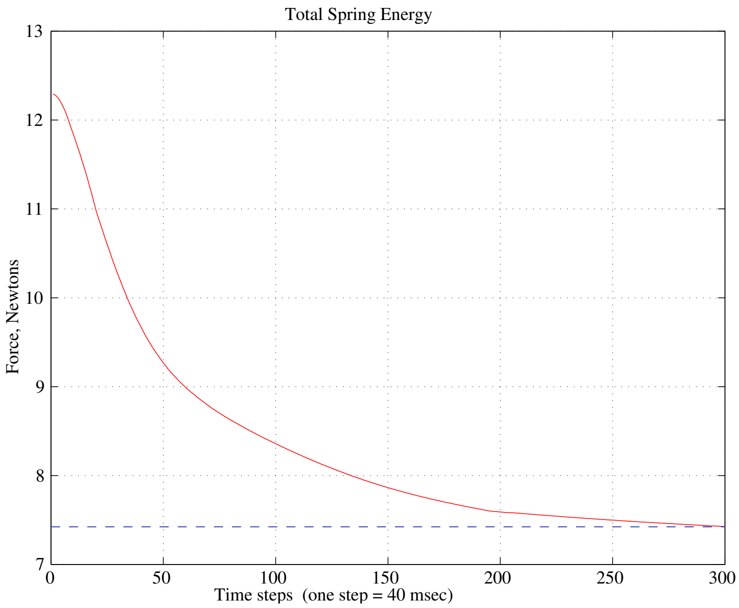


Figure 3.35: Total spring force (linear and angular) decreases over time (depicted by the solid curve) and converges toward the steady-state potential energy of the stable vertex configuration (depicted by the dashed line). Note that the complex shape of the curve is due to the number of modes of vibration of the coupled spring-mass system. However, the fact that the system does not oscillate toward the steady-state solution indicates that it is not underdamped.

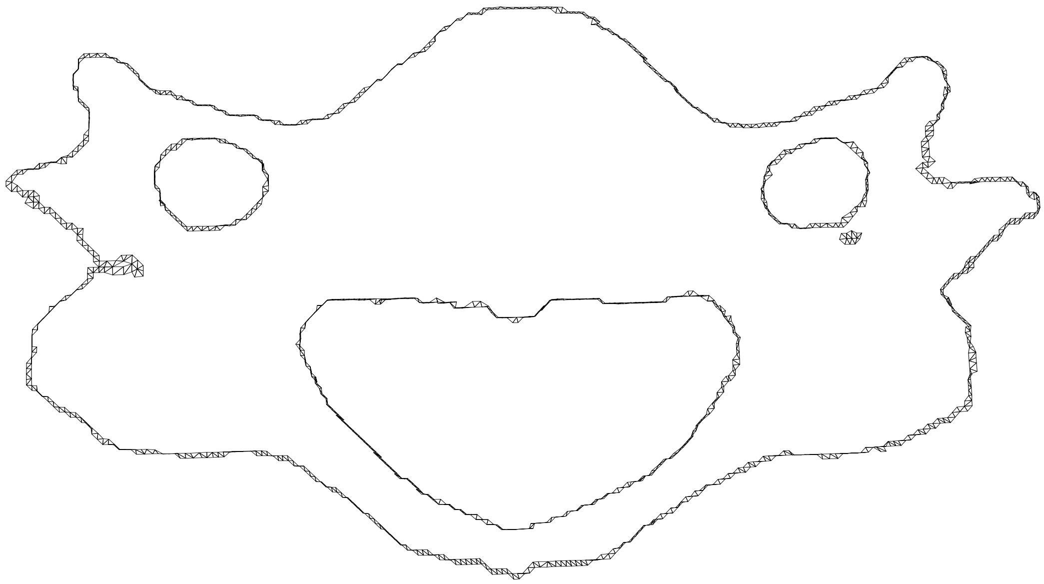


Figure 3.36: The set of all triangles from the FEM model of C4 that pass through the slice plane $z_n = 184$.

$$Ax_1 + By_1 + Cz_1 + D \leq 0 \implies Ax_2 + By_2 + Cz_2 + D \geq 0. \tag{3.71}$$

The value of α satisfying Equation 3.68 and condition 3.69 determines the point of intersection for each line segment. Each point can then be joined with the neighboring segments to form a polygon that encloses the vertebral body for the given slice. To reduce the number of edges that need to be tested, the FEM triangles are first sorted by minimum and maximum distances from the slice plane. Only those triangles whose min/max values span the positive and negative sides of the slice plane are considered for the precise test. Figure 3.36 shows a view of the set of triangles passing through the plane $z_n = 184$ (i.e., the plane corresponding to the $184th$ axial slice).

Once the polygon outline has been determined, the two-dimensional vertebra can then be displayed using a standard polygonal scan conversion algorithm to remove points outside the boundaries [31]. For a three-dimensional display, the extracted two-dimensional images can be stacked and fed to a standard volume-rendering program.

3.7 Summary

In this chapter, I have described a method for constructing a three-dimensional finite-element surface model of the cervical vertebrae. The volumetric CT image data is first edited to remove any artifacts and then segmented to remove any bridges between adjacent cervical segments. The segmented data is then fed into a modified polyhedral tessellation algorithm that produces surface triangulations of each of the given vertebra. A smoothing and triangle-decimation step is then used to reduce the number of triangular surface elements in the model. The CT data set used to construct the model was selected from a sample of patients and chosen on the basis of having geometric properties that are closest to the mean of the sample data. This data set, along with a set of parameterizations to account for geometric variations seen in the sample data, constitute the *generic* spine model. Using this generic model, *patient-specific* models are constructed using a three-step process: (1) perform a crude match of the generic model to the patient data by locating the four principle landmarks on the patient data and scale, translate, and rotate the generic model so as to achieve an initial best fit; (2) modify the locations of the triangle vertices of the model so as to deform the generic model to fit the vertebra in the CT data set; and (3) optimize the fit using the matching score generated in step two. The global error minimization between the model and data is found by using the NPSOL optimization package, an implementation of a sequential quadratic programming algorithm. Once an optimal fit has been achieved, the resulting model can then be used to extract and fuse the bony vertebra with the MR images, or to generate the biomechanical model for simulation or visualization purposes. Furthermore, once the patient-specific model has been generated, it is possible to scan convert the two-dimensional polygonal slices of the FEM model at each image plane location to extract the bony structures from the image through any given slice plane.

Chapter 4

Error Estimation

To quantitatively verify the feasibility of the model-based segmentation technique described in Chapter 3, I have carried out a series of experiments using a test model and a test data set. The two main purposes of these experiments are to show (1) that an optimal generic finite-element model can be constructed and (2) that such a model can be used to accurately segment similarly shaped variants from a synthetic volumetric data set. In Section 4.2.1 I will describe what constitutes an optimal generic model and how accurate the segmentation results are using this model. Not only do these experiments demonstrate the feasibility of the model-based technique, they also establish a lower bound on the accuracy of the segmentation results for real data. Because I start with analytical descriptions, the precise geometries of both the generic model and the synthetic data are known. The major steps in this feasibility study are the following: (1) create a mathematical description of the test object, (2) generate a synthetic volumetric CT data set using this description, (3) create a mathematical description of the generic model, (4) create a finite-element representation of the generic model, and (5) segment the test object from the synthetic data by deforming the generic model to match the boundaries of the test object. I have experimentally determined a lower bound on the segmentation accuracy by comparing the deformed generic model to the analytical test object. In all cases, the accuracy was assessed using the tetrahedral volume measure. The preliminary results show that it is feasible to use a deformable finite-element model to segment volumetric data representing simple objects. Furthermore, in the process of accessing the accuracy of the matching technique, I have also developed a post processing algorithm that has improved the accuracy of existing surface-extraction techniques such as marching cubes and tetrahedral decomposition.

4.1 Test Object

The criteria for a good test object are that the object should be moderately irregular, asymmetric, analytically tractable, and twice differentiable. The irregularity is necessary to ensure that the test case is not trivial to segment, and that regularity assumptions cannot be made by the segmentation algorithm.

The asymmetry is needed to ensure that there will be a unique spatial orientation of the object. Analytical tractability is necessary so that the segmentation results can be compared against the analytical description of the test object (i.e., the gold standard). And finally, smoothness of the second derivative is necessary for the optimization algorithm that uses the analytical description to determine the best case error (lower bound).

4.1.1 Analytical description of test object

The test object was created using a series of ellipsoids superimposed at various orientations (see Figure 4.1). The analytical description of this object is given by

$$\begin{aligned} f(x,y,z) = &\sum_{i=1}^{N} g(x, y\cos\omega_i + y\sin\omega_i, z\cos\omega_i - z\sin\omega_i) \\ &+ \sum_{j=1}^{N} g(x\cos\omega_j + x\sin\omega_j, y, z\cos\omega_j - z\sin\omega_j) \\ &+ \sum_{k=1}^{N} g(x\cos\omega_k + x\sin\omega_k, y\cos\omega_k - y\sin\omega_k, z), \end{aligned} \tag{4.1}$$

where

$$\omega_i = \frac{2\pi i}{N}, \omega_j = \frac{2\pi j}{N}, \omega_k = \frac{2\pi k}{N}, \tag{4.2}$$

and

$$g(x,y,z) = \begin{cases} \frac{1}{x^2+y^2+z^2} & \text{for } (x^2+y^2+z^2) > \epsilon^2 \\ 1/\epsilon^2 & \text{otherwise.} \end{cases} \tag{4.3}$$

Setting $f(x,y,z) = k$ yields the implicit definition of the surface of the test object (in these experiments, $k = 100$). The solution of Equation 4.1 for all values of x, y, and z that yield k is very difficult to achieve in closed form. Instead, numerical techniques are used to solve this equation to a specified degree of accuracy (roughly 10^{-10}, which is at least seven orders of magnitude smaller than the spatial resolution of the CT data sets). At first glance, Equation 4.1 bears a superficial resemblance to the discrete Hartley transform (DHT) of a three-dimensional signal [12]. In reality, it is closer to a wavelet decomposition [49] where $g(x,y,z)$ represents the wavelet basis function (in fact, the classic definition of a wavelet is a ripple or small wave). Interestingly enough, one can also view a specific model as a modification of the Fourier components of the generic model. In this case, another way to generate a specific model is to truncate the higher-order terms of the Fourier series representating the generic model. As is the case with the fast Fourier transform (FFT) [11] and the fast Hartley transform (FHT) [14], the transcendental factors

$\cos\omega_i$ and $\sin\omega_i$ can be precomputed because the values of ω_i are both discrete and finite. As a result, the evaluation of Equation 4.1 can be done very efficiently. Furthermore, the function is twice differentiable except near the surface of the ϵ-sphere centered about the origin. This is not of any practical significance because the ϵ-sphere is a tiny stationary region far away from the surface of the test object.

The test object can be made asymmetric by anisotropically scaling and shearing the (x, y, z) variables:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = SA \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & s_{13} & 0 \\ s_{21} & s_{22} & s_{23} & 0 \\ s_{31} & s_{32} & s_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x & 0 & 0 & 0 \\ 0 & a_y & 0 & 0 \\ 0 & 0 & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{4.4}$$

The matrix S controls the degree of shearing and the matrix A controls the scaling of the x, y, z coordinates. For moderate degrees of shearing, S should be a diagonally dominant matrix. Setting the diagonal elements of A to (1,2,4) yields a sufficiently anisotropic model. The test object can also be rotated and translated by applying the homogeneous transformation operators to the x, y, z coordinates,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = TSAR_xR_yR_z \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{4.5}$$

where T represents the translation matrix and R_x, R_y, and R_z represent the rotation matrices along the x, y, z axes respectively.

4.1.2 Polygonization of test object

A finite-element surface representation of the test model was generated first by using an implicit surface-tiling algorithm developed by Bloomenthal [9]. This algorithm will accept an analytical description of an object and generate a set of triangular surface patches using a tetrahedral decomposition technique (see Section 3.4). I have improved the speed and accuracy of the algorithm by using a quadratic line search instead of the linear search used by Bloomenthal for finding the surface intersection points. The quadratic line search is faster because it converges to the surface threshold in three or four iterations with a normalized error of 10^{-8}. The quadratic line search is more accurate than the linear search for volumetric data because it fits a quadratic function to voxel grid data, as opposed to performing a simple linear interpolation. The linear search method generates vertex coordinates that are further away from the true surface (except in those cases where the data is truly trilinear). Also, I have added the capability of tiling both volumetric data as well as analytical functions. The output of the modified surface-tiling algorithm applied to the test model is shown in Figure 4.2. A three-dimensional rendering of the test model is shown in Figure 4.3.

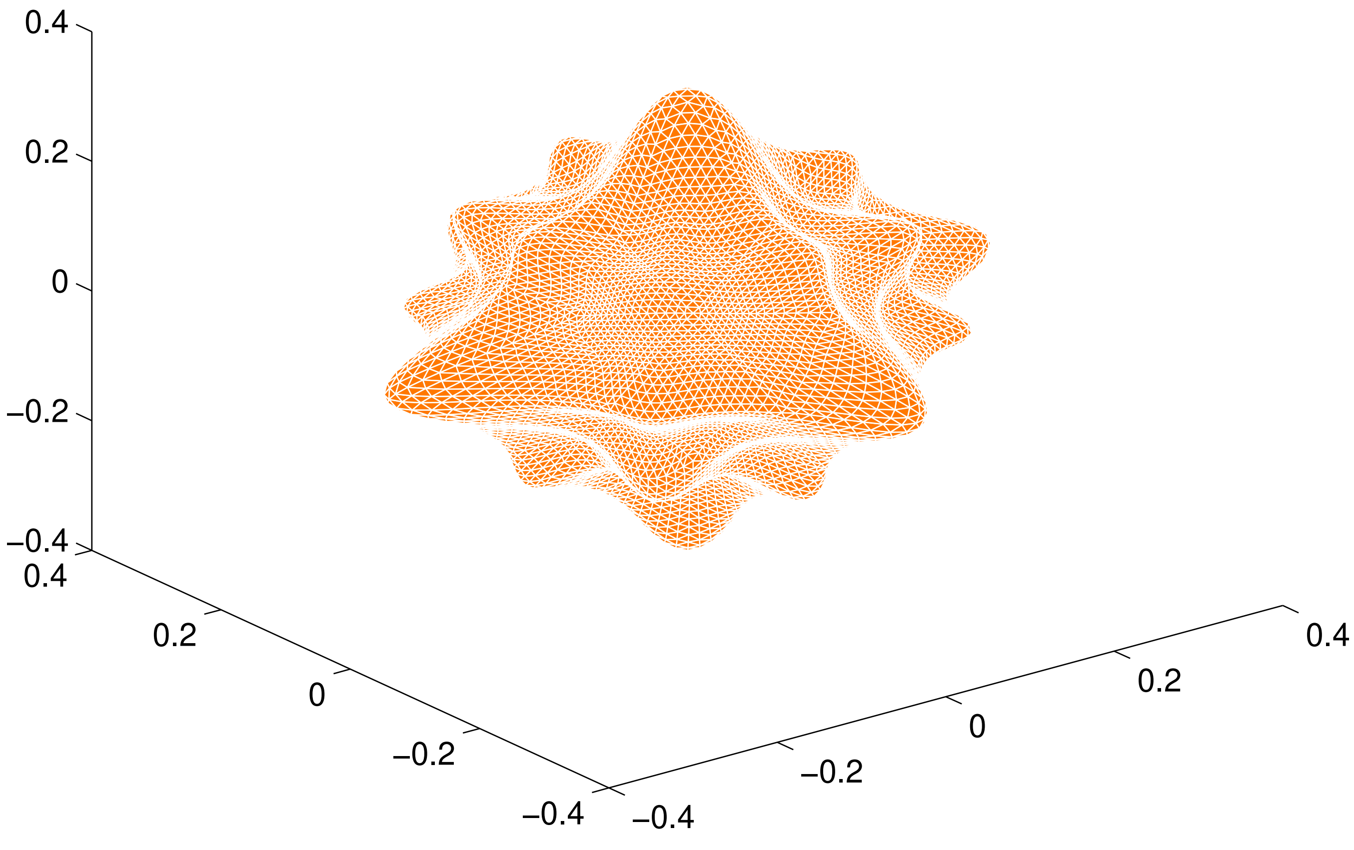


Figure 4.1: The test object shown above was constructed by overlapping ellipsoids at various orientations. Because the implicit surface is described by a series of conic sections, this object is continuous in the second derivative.

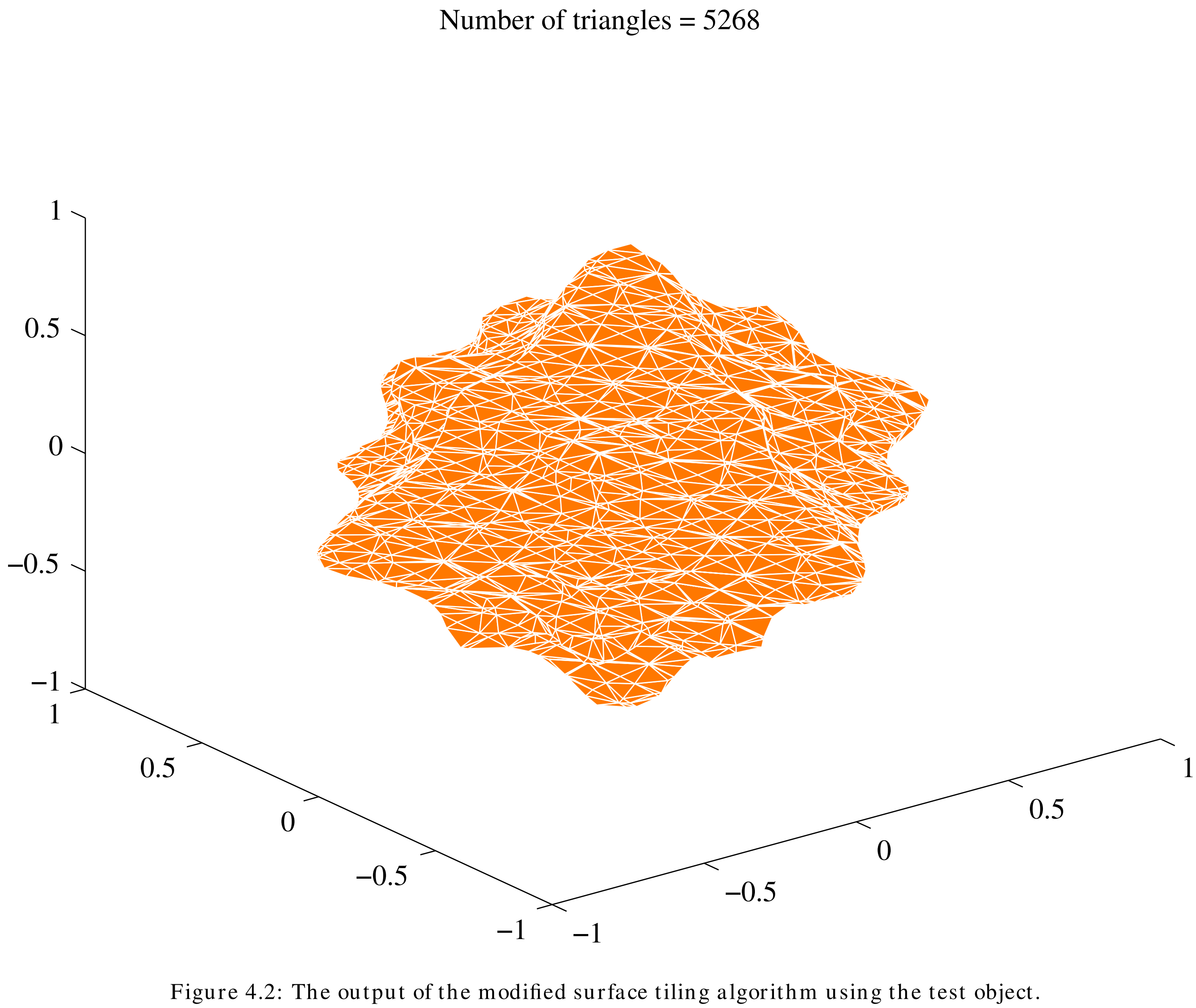
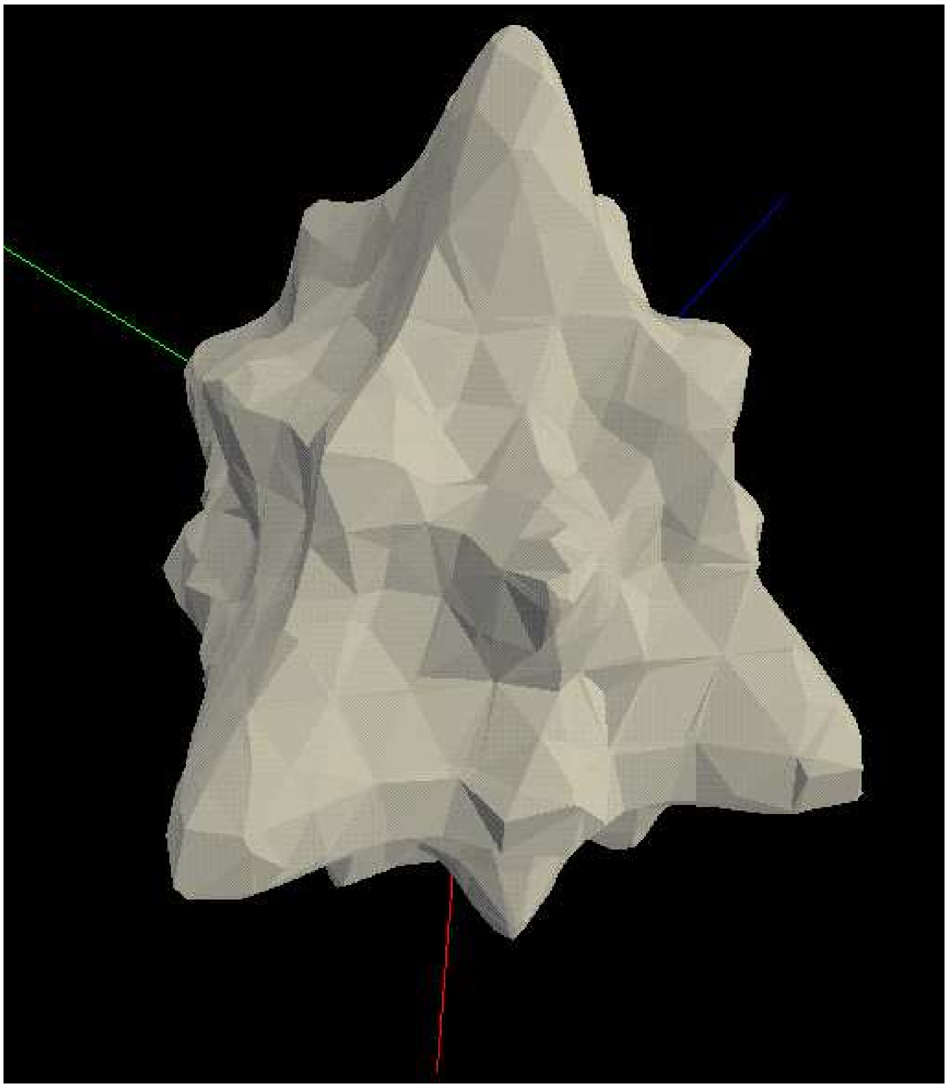


Figure 4.2: The output of the modified surface tiling algorithm using the test object.



Number of triangles = 5268

Figure 4.3: Three-dimensional rendering of the test model using the triangles generated by the surface-tiling algorithm.

4.1.3 Synthesis of volumetric test data

Converting the analytical description of the test object into a volumetric data set is a straightforward process of evaluating the function f at regularly spaced grid intervals in $\Re^3$. To make the function simulate the bone distribution found in typical spine CT images, values of f that exceed k are mapped into the range of bone intensity values (1500 to 3000). Values of f smaller than k are mapped to zero. This produces cross-sectional images of the test data shown in Figure 4.4.

Once the mapping has been performed, Gaussian noise with zero mean, standard deviation σ_n, and amplitude a_n is added to the voxel intensity values. Initially, the matching experiments were performed with $a_n = 0$.

4.2 Generic Model Construction

The simplest and most general shape that describes the test object is an ellipsoid. Hence, the generic model was chosen to be an ellipsoid with the same ratio of principal axes as the test object (1:2:4),

$$f_g(x, y, z) = \frac{x^2}{a_x} + \frac{y^2}{a_y} + \frac{z^2}{a_z}. \tag{4.6}$$

It is interesting to note that this generic model is also the basis function for the wavelet decomposition of the test model. However, this is not necessarily true in general (one can specifically construct an object by composing two or more different basis functions). Figure 4.5 shows the polygonization of the generic model with varying numbers of surface triangles.

4.2.1 FEM optimization

There is no guarantee that implicit tiling algorithms that use either quadratic or linear search techniques are optimal in producing the most accurate models using the fewest number of triangles. The notion of optimality can be formalized by the following definition:

Definition 4.1 *A polygonization P of a surface S is* globally optimal *with respect to an error metric E if and only if (1) there is no other arrangement of vertex coordinates of P that will lower the* total *error of P as measured by E (called E(P)) and (2) all other polygonizations of S that involve fewer surface elements than P will produce a strictly higher error than E(P).*

Except for a few simple cases, it is not possible to determine if a polygonization is globally optimal. Once the number of triangles in a polygonization exceeds the order of 10, it becomes impractical to search for the globally optimal arrangement of vertices. A more practical definition of optimality is the following:

Definition 4.2 *A polygonization P of a surface S is* simplicially optimal to degree r *with respect to an error metric E if and only if there is no other arrangement of (r-1)-connected vertices of P within the convex*

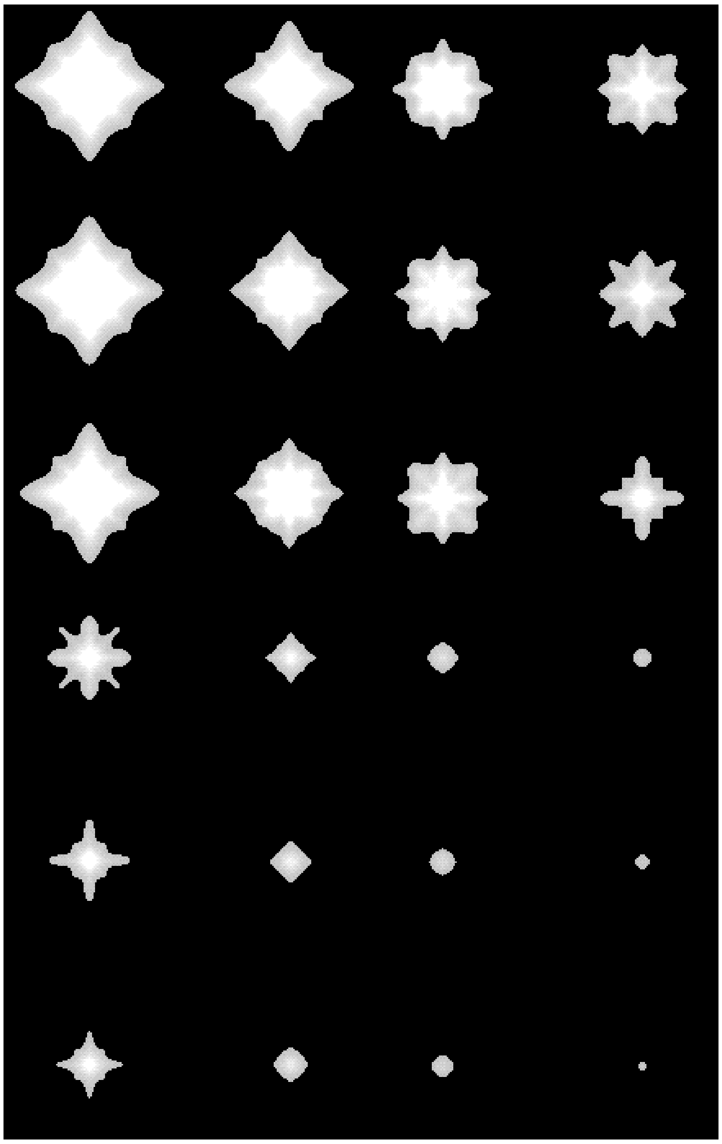


Figure 4.4: Cross-sections of the isotropic test object volume (256^3 voxels) after applying an intensity transformation step to simulate the distribution of bone surrounded by air.

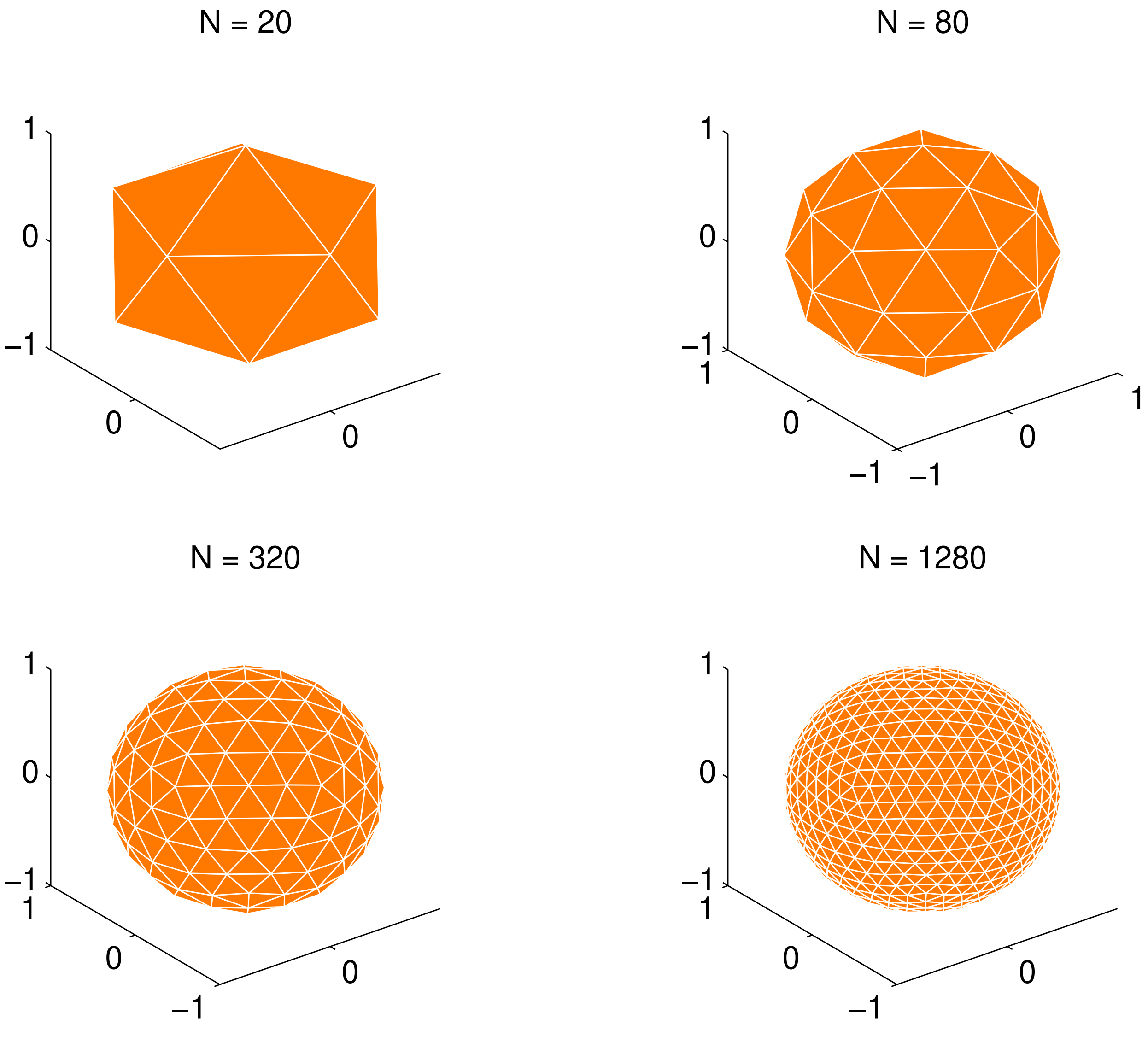


Figure 4.5: Initial finite-element surface representation of the generic model is shown with varying numbers of triangular surface elements (N).

region of the r-simplex X_v containing each vertex of P taken one at a time that will lower the error of X_v as measured by E. The r-simplex *is the polyhedron formed by a vertex v and all the neighboring vertices that can be reached by at most* r *edges. A vertex v_a is said to be r-connected to vertex v_b if there are at most r edges between v_a and v_b.*

In other words, if a polygonization is simplicially optimal to degree r, then moving the location of the (r-1)-connected vertices of a given r-simplex will not yield a smaller error for that polygonization. For the experiments in this chapter, the polygonizations are taken to be simplicially optimal to degree one. That means that moving any of the vertices one at a time will not lower the total error of that polygonization. Although it is not allowed by the definition of simplicial optimality of degree one, it may be possible that moving two or more 1-connected vertices simultaneously would yield a lower error. Note that global optimality implies simplicially optimality of all degrees.

4.2.2 Generic model optimization

Simplicial optimality is most useful when the initial polygonization is near the globally optimal configuration. In order to optimally polygonize the test data, it is necessary for the *generic* model to be as globally optimal as possible. I have used two different approaches for creating a near-globally optimal polygonization of the generic model. The first approach (Method A) starts with an icosahedron and recursively subdivides each of the triangles into four smaller, triangles as shown in Figure 4.6. Once a triangle has been subdivided, the newly created vertices no longer lie on the surface of the sphere and must be projected back onto the surface using the face normal of the original triangle. Because the icosahedron has 20 face triangles, the number of uniform triangles at recursion level n is given by $20 \times 4^{(n-1)}$. The reason for starting with a icosahedron instead of regular polyhedrons such as cubes or dodecahedrons is that the normals of the icosahedron are closer to the true surface normals of a sphere over the range of each face (a regular polyhedron is one that is made up of identical regular polygonal faces). Furthermore, recursively subdividing a cube or dodecahedron into subtriangles will not produce regular triangles. Also, it can be proved that there is no regular polyhedron with more than five faces meeting at a vertex [63]. Hence, the icosahedron is the largest (in terms of face count) regular polyhedron in three-dimensional Euclidean geometry.

The second approach (Method B) for creating a near globally optimal polygonization is a relaxation method whereby each vertex v_i is treated as a charged particle that exerts a repulsive force F_ij on every other vertex $v_j (i \neq j)$ that is inversely proportional to the square of the distance between the two vertices (Coulomb's law [70]):

$$F_{ij} = -\frac{q_i q_j}{r_{ij}^2}, \tag{4.7}$$

where q_i and q_j are the electrostatic charges associated with each particle and r_ij is the distance between particles i and j. For an uniform distribution, the charges q_i and q_j all can be taken to be unity. Each

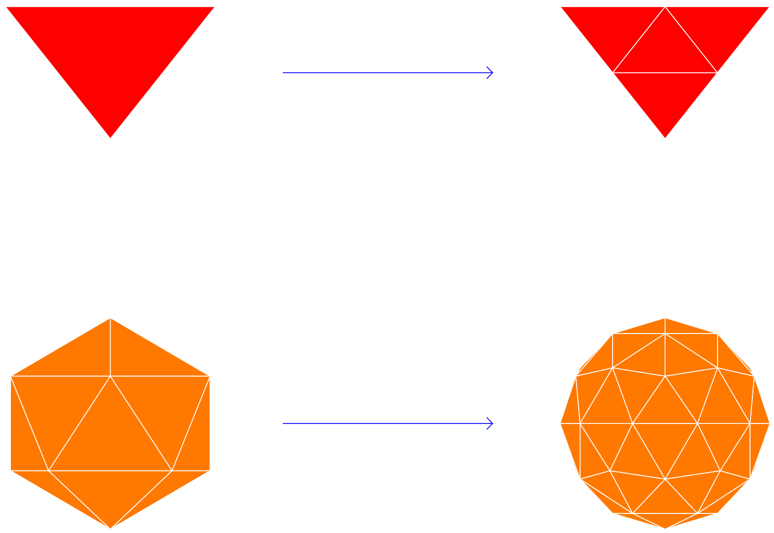


Figure 4.6: [Method A] Starting with an icosahedron, each face triangle is recursively subdivided into four subtriangles. The number of uniform triangles is $20 \times 4^{(n-1)}$ where n is the level of recursive subdivisions.

charge will move with a acceleration directly proportional to the magnitude of the net force vector acting on it. The direction of the acceleration is along the projection of the net force vector onto the tangent plane at the vertex location (each particle is constrained to the surface of the sphere). Using this kinematic model, any random distribution of particles will come to rest at an equilibrium distribution when all the net forces acting on each particle are zero (see Figure 4.7a).

The equilibrium distribution is found by numerically iterating the paths of each particle using a velocity-dependent time step. Once the equilibrium state has been reached (i.e., the magnitudes of all the velocities have fallen below some small threshold), the particles can be triangulated to create the surface elements. This is accomplished by using a convex hull subroutine developed by Pankratov [64] that connects vertices using the smallest possible triangular elements (see Figure 4.7b). One of the properties of the convex hull is that it is the smallest convex polygon P that encloses a set of points S (i.e., there is no smaller convex polygon that encloses both P and S) [63]. The results of the convex hull algorithm applied to the equilibrium vertex distribution are shown in Figure 4.7b.

4.2.3 Simplicial optimization (degree one)

I will now describe the procedure used to make a polygonization P simplicially optimal to degree one given a reference object R and an error metric E. The goal is to minimize the error between R and P as measured by E. Recall that degree-one optimality means that only one vertex at a time needs to be adjusted. For the generic model, moving one vertex will change the error of at most six adjacent triangles. The procedure for optimization is outlined below:

This procedure attempts to find a configuration of vertex locations such that the total sum (e_T) of the

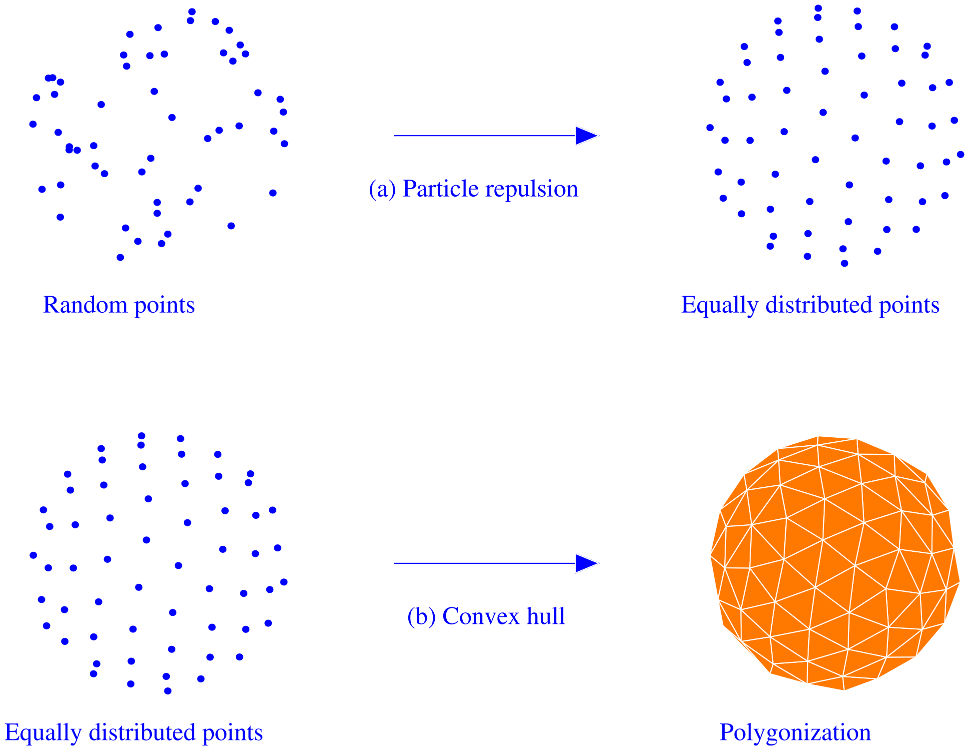


Figure 4.7: [Method B] A near globally-optimal polygonization of the test model can be created by taking a randomly distributed set of points representing charged particles lying on spherical surface and applying a repulsive Coulomb force to each particle (top). Once a uniform dispersion of particles has been achieved, there will be no net forces acting on any particle. A convex hull algorithm can then be used to triangulate the points and create a surface mesh (bottom).

```
Repeat
    For all vertices v in P
        Find the simplex s of degree 0 containing v
        Compute the centroid c of s
        Compute average normal n of s at c
        Move v to the intersection of n and object surface S
        Minimize error e(v, s, S) subject to v lying on S
Until total error e_T < e_0
```

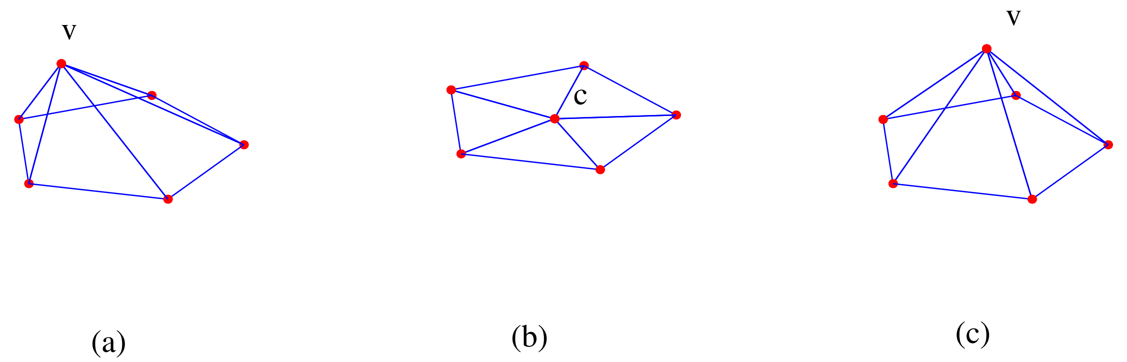


Figure 4.8: Before performing an optimization operation on the simplex shown in (a), the vertex v is moved as close to the optimal position as possible by first locating the centroid c shown in (b) and then projecting v along the average simplex normal until it intersects the object surface (c).

individual error terms $e(v, s, S)$ is below a specified value e_0. It does this by examining each vertex v and the simplex s formed by all the vertices immediately adjacent to v (separated by exactly one edge), as shown in Figure 4.8a. Because it is important to be as close to the minimum as possible before initiating the minimization step, the vertex is moved to the intersection of the perpendicular medial axis passing through the centroid of of the simplex (Figure 4.8b) and the object surface (Figure 4.8c). This step also reduces the possibility of producing degenerate triangles or topologically impossible configurations of the simplex during the minimization step.

Once the vertex has been initialized on the medial axis, the error minimization procedure is invoked to find the location of the vertex on the object's surface that minimizes the sum of the errors e_S of each of the triangles in the simplex. If the size of the simplex is sufficiently small, the error can be modeled by the following expression:

$$e_S(\mathbf{x}) \approx \mathbf{x}^T A\mathbf{x} + B\mathbf{x} + c, \tag{4.8}$$

where $\mathbf{x}$ is the (x, y, z) coordinate of vertex v, A is a 3×3 quadratic coefficient matrix, B is a 3×1 linear coefficient vector, and c is a scalar offset. Assuming A is positive semi-definite, Equation 4.8 has ten unknown terms (six from A, three from B, and one from c). These unknown terms can be determined by walking (stepping) along the path of steepest descent and evaluating the error function e_S at each location (shown in Figure 4.9).

Because the (x, y, z) coordinates are constrained to lie on the surface S, the steepest descent algorithm must be modified slightly to ensure that the path toward minimization does not leave S. This is done by taking the projection of the gradient onto the tangent plane at each step location, as shown in Figure 4.10.

The direction of steepest descent $\mathbf{d}$ must be projected onto the surface tangent plane whose normal $\mathbf{n}$ is simply the gradient of the object function $g(x, y, z)$. The component of $\mathbf{d}$ parallel to the tangent plane $\mathbf{p}$ can be found by subtraction:

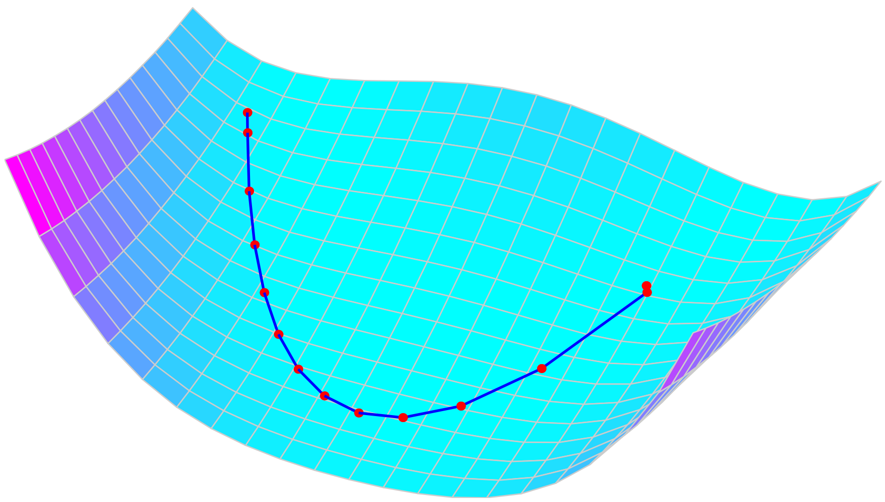


Figure 4.9: Shown above is the steepest descent path along the object surface S in the neighborhood of the local minimum.

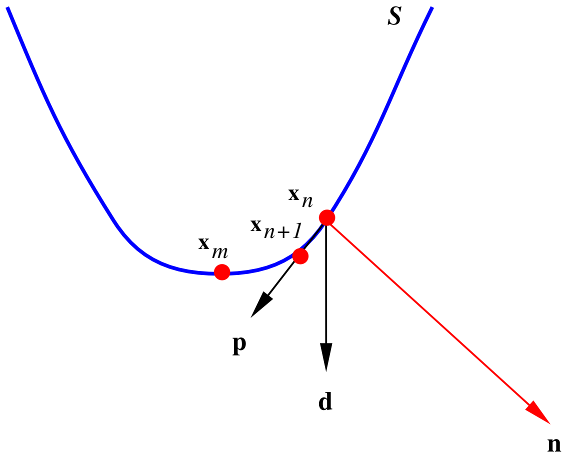


Figure 4.10: The surface-constrained minimum lies at location $\mathbf{x}_m$, whereas the unconstrained minimum lies somewhere in the direction of $\mathbf{d}$. To constrain the steepest descent path to the surface S at each step location $\mathbf{x}_n$, the error gradient vector $\mathbf{d}$ must be projected onto the tangent surface (indicated by its normal $\mathbf{n}$) to form the corrected step direction $\mathbf{p}$.

$$\mathbf{d} = \frac{\mathbf{d} \cdot \nabla \mathbf{g}(\mathbf{x})}{\|\nabla \mathbf{g}(\mathbf{x})\|} + \mathbf{p}$$

$$\Rightarrow \quad \mathbf{p} = \mathbf{d} - \frac{\mathbf{d} \cdot \nabla \mathbf{g}(\mathbf{x})}{\|\nabla \mathbf{g}(\mathbf{x})\|}. \tag{4.9}$$

The updated location of **x** can found by moving in the direction of **p** by a factor β:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \beta \mathbf{d}. \tag{4.10}$$

The value of β is nominally set to $0.01 \times \sqrt{\text{Area}(s)}$, that is, one percent of the square root of the area of the simplex. The location $\mathbf{x}_{n+1}$ is close to, but still not on, the surface S. The correct location can be found by determining the intersection of the direction vector $\nabla \mathbf{g}(\mathbf{x}_{n+1})$ at location $\mathbf{x}_{n+1}$ with S.

By proceeding in this manner and accumulating 10 samples of e_S, the matrix A, vector B, and scalar c can be determined by solving the following system of linear equations:

$$\begin{bmatrix} e_{S1} \\ e_{S2} \\ \vdots \\ e_{S10} \end{bmatrix} = \begin{bmatrix} x_1^2 & y_1^2 & z_1^2 & x_1 y_1 & x_1 z_1 & y_1 z_1 & x_1 & y_1 & z_1 & 1 \\ x_2^2 & y_2^2 & z_2^2 & x_2 y_2 & x_2 z_2 & y_2 z_2 & x_2 & y_2 & z_2 & 1 \\ \vdots & & & & & & & & & \vdots \\ x_{10}^2 & y_{10}^2 & z_{10}^2 & x_{10} y_{10} & x_{10} z_{10} & y_{10} z_{10} & x_{10} & y_{10} & z_{10} & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{22} \\ a_{33} \\ 2a_{12} \\ 2a_{13} \\ 2a_{23} \\ b_1 \\ b_2 \\ b_3 \\ c \end{bmatrix}.$$

Once the values of A, B, and c are known, the (x, y, z) coordinates of the error extremum can be found by setting the partial derivatives of Equation 4.8 equal to zero and solving the resulting set of simultaneous linear equations:

$$\frac{\partial}{\partial x} e_S(\mathbf{x}) = \mathbf{x}^T \begin{bmatrix} 2a_{11} & a_{12} & a_{13} \\ a_{21} & 0 & 0 \\ a_{31} & 0 & 0 \end{bmatrix} \mathbf{x} + B = 0,$$

$$\frac{\partial}{\partial y} e_S(\mathbf{x}) = \mathbf{x}^T \begin{bmatrix} 0 & a_{12} & 0 \\ a_{21} & 2a_{22} & a_{23} \\ 0 & a_{32} & 0 \end{bmatrix} \mathbf{x} + B = 0,$$

$$\frac{\partial}{\partial z}e_S(\mathbf{x}) = \mathbf{x}^T \begin{bmatrix} 0 & 0 & a_{13} \\ 0 & 0 & a_{23} \\ a_{31} & a_{32} & 2a_{33} \end{bmatrix} \mathbf{x} + B = 0. \tag{4.11}$$

The coordinates of $\mathbf{x}$ that satisfy Equation 4.11 will be the exact location of the extremum if e_S was exactly described by Equation 4.8. If the approximation is poor, or if $\|\mathbf{p}\|$ is very small, it is possible for A to be singular or poorly conditioned. A small value of $\|\mathbf{p}\|$ implies that $\mathbf{x}$ has nearly converged to the locale minimum, so the procedure will exit. However, if $\|\mathbf{p}\|$ is not small, and A is poorly conditioned, more samples of e_S should be collected, and older samples should be thrown away until A is better conditioned. Hence, this method suffers from a major drawback: Because it requires many sample points to determine A, it is neither fast nor robust.

For this particular application, the optimization technique can be simplified and made somewhat more robust by introducing a new parameter u and modeling the error as a function of u:

$$e_S(x, y, z) = e_S(u) = a_u u^2 + b_u u + c_u \tag{4.12}$$
$$x(u) = a_x u^2 + b_x u + c_x \tag{4.13}$$
$$y(u) = a_y u^2 + b_y u + c_y \tag{4.14}$$
$$z(u) = a_z u^2 + b_z u + c_z, \tag{4.15}$$

where u is equal to the arc length along the path of steepest descent. This form is more restrictive that Equation 4.8, but it has the advantage that the (x, y, z) coordinates are effectively decoupled from one another. As a result, there are only three unknown variables in each of three sets of independent equations. Therefore, only three sample values are required to solve for all the unknown parameters in the decoupled case, as opposed to ten in the coupled case. These unknown parameters can be solved quite easily:

$$a_u = \frac{e_3(u_2 - u_1) - e_2(u_3 - u_1)}{e_3\sqrt{e_2} - e_2\sqrt{e_3}}, \tag{4.16}$$
$$b_u = \sqrt{e_3}(u_2 - u_1) - \sqrt{e_2}(u_3 - u_1), \tag{4.17}$$
$$c_u = u_1, \tag{4.18}$$

and

$$a_x = \frac{x_3(u_2 - u_1) - x_2(u_3 - u_1)}{x_3\sqrt{x_2} - x_2\sqrt{x_3}}, \tag{4.19}$$
$$b_x = \sqrt{x_3}(u_2 - u_1) - \sqrt{x_2}(u_3 - u_1), \tag{4.20}$$
$$c_x = x_1. \tag{4.21}$$

Similar equations hold for $(a_y, b_y, c_y, a_z, b_z, c_z)$. Because $u_1 = 0$, it does not appear in any of the equations. This technique is robust as long as $e_3^2 e_2 \neq e_2^2 e_3$, that is, $e_3 \neq e_2$. The minimum of e can be found by solving the following:

$$\frac{\partial}{\partial u} e_S(u) = 2a_u u + b_u = 0, \tag{4.22}$$

or

$$u_m = -\frac{b_u}{2a_u}. \tag{4.23}$$

This value of u_m can then be substituted into Equation 4.12. If Equation 4.12 is not a valid approximation of e_S, then Equation 4.22 provides a meaningless value of u_m and hence a meaningless coordinate $[x(u_m)\ y(u_m)\ z(u_m)]^T$. To guard against this possibility, the value of $e_S(u_m)$ is compared with the current values of e_S, i.e., $e_S(u_1), e_S(u_1)$, and $e_S(u_3)$, and if it is greater than any of these, the search continues along the steepest descent path. That is, $\mathbf{x}_{n+1}$ is chosen over $\mathbf{x} = [x(u_m)\ y(u_m)\ z(u_m)]^T$ and the process of acquiring three samples of e_S is repeated until no further improvement in e_S can be made.

4.2.4 Simplicial optimization (higher degree)

For optimization of simplices of degree greater than one, I used the NPSOL non-linear programming subroutines developed by Gill, Murray, Saunders, and Wright [34]. NPSOL is designed to minimize a smooth objective function subject to a set of constraints, which may be simple bounds, linear bounds, or smooth non-linear constraints. The user provides the objective function subroutine $e(x_1, x_2, \ldots, x_N)$, along with the contraint bounds and non-linear contraint functions, and the NPSOL package finds the minimum of e subject to these constraints. In this case, the variables x_i represent the values of the (x, y, z) coordinates of all the vertices in the model. Hence, if there are m vertices, then $N = 3m$. The error function e is defined as the sum of the tetrahedral error terms for all the surface triangles. No simple bounds or linear constraints were imposed on e and the only non-linear constraint was that all the vertices must lie on surface S, i.e., $g(x_1, x_2, \ldots, x_N) = 0$.

4.3 Matching results

To find an approximate lower bound for the best-case matching error, I computed the error of the generic models (derived from both the particle simulation and triangular subdivision techniques) using the ideal ellipsoid (Equation 4.6) as the gold standard. Table 4.1 summarizes the tetrahedral errors (see Section 3.2.1) of these two generic models for various number of triangles. As the number of triangles in each model increases by a factor of 4096 (from 20 to 81920), the error decreases by a factor of 2600 (from 0.6554 to 2.5175×10^{-4}). This data is depicted graphically in Figure 4.11. Table 4.2 shows the difference between the tetrahedral volume metric and the tetrahedral apex height metric. Note that the volume metric decreases by more than 3 orders of magnitude whereas the height metric decreases by less than 2 orders of magnitude as the number of triangles range from 20 to 81920 (4 orders of magnitude). This corroborates the notion that the height metric favors models with fewer surface elements than does the volume metric. However, this does not mean that the height metric is not useful. Even though the volume metric is used by the minimization algorithm, the height metric is useful for reporting a measure of the linear mismatch error (e.g., 2 mm boundary estimation error, 5 mm localization error, etc).

Tables 4.3 and 4.4 show the improvement in the errors after simplicial optimization was performed on the two generic models. These results indicate that the two different approaches for creating a generic model are near globally optimal and produce very similar errors.

Finally, Table 4.12 shows the errors obtained when matching the generic model to the test object. First the generic model was fitted to the volumetric representation of the test object, thereby creating the specific model. The errors were calculated by comparing the specific model to the analytical representation of the test object. Also shown for reference in Table 4.12 are the errors obtained by comparing the generic model to its analytical representation. The reference values are the same errors shown in Table 4.1. The mean height metric varies from 0.0181 to 1.436×10^{-4} as the number of triangles is varied from 20 to 81920. If the test object were to be scaled to represent an object with a width of 100 mm, the height error would corresponded to mismatch errors ranging from 1.8 mm down to 0.014 mm. Using 320 triangles, the average height error would be 2.3 mm. It is interesting to note that for small numbers of triangles ($\sim$ 20), the specific model had a lower error than the reference model. This may be due to the fact that the test object has a non-uniform curvature that may have many flat regions. The generic model has a constant curvature, and hence, a constant error volume for each face triangle. However, as the number of triangles increases, the errors for the reference model diminish whereas they level off for the specific model. This may be due to a surface offset error induced by the sampling process that is used to create the volumetric data for the test object. This phenomenon will have to be investigated further to determine its exact cause.

4.4 Discussion of Results

The preliminary results presented in this chapter indicate that a finite-element surface model can be used to segment a simple object from simulated CT data sets. For a 100 mm sized test object, the average mismatch errors (using the tetrahedral height metric) ranged from 1.8 mm using the fewest number of triangles (20) down to 0.014 mm using the maximum number of triangles (81920). For the vertebral model, it is highly unlikely that the number of surface triangles will be near either of these two extreme values. Although the generic model with 320 triangles achieved an average mismatch error of 2.3 mm, such a figure should be viewed as a lower bound for the error expected using real CT images.

Generic model matching errors (before optimization)

Triangles	Face subdivision (Method A)		Particle simulation (Method B)	
	Total *error*	Mean *error*	Total *error*	Mean *error*
20	0.6554	1.9661×10^{-1}	0.6554	1.9661×10^{-1}
80	0.2286	1.7146×10^{-2}	0.2293	1.7199×10^{-2}
320	0.0625	1.1723×10^{-3}	0.0629	1.1785×10^{-3}
1280	0.0160	7.4961×10^{-5}	0.0160	7.4958×10^{-5}
5120	0.0040	4.7119×10^{-6}	0.0041	4.7462×10^{-6}
20480	0.0010	2.9492×10^{-7}	n/a	n/a
81920	2.5175×10^{-4}	1.8439×10^{-8}	n/a	n/a

Table 4.1: Summary of matching errors (in mm) for the generic models compared to the ideal ellipsoid *before* any optimizations. Entries marked n/a were unavailable due to the excessive computational time required.

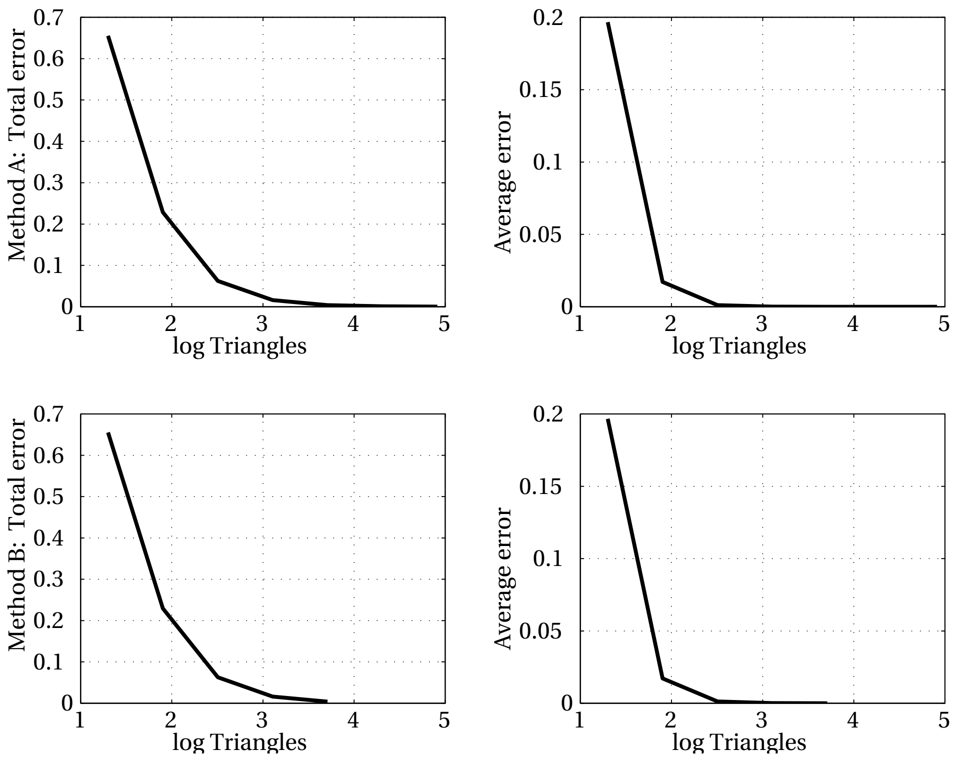


Figure 4.11: Comparision of matching errors (in mm) for the generic models compared to the ideal ellipsoid *before* any optimizations were performed.

Tetrahedral volume metric versus height metric

Triangles	Face subdivision (Method A)		Particle simulation (Method B)	
	Mean height	Mean volume	Mean height	Mean volume
20	0.6554	0.9183	0.6554	0.9183
80	0.2286	0.5243	0.2293	0.5272
320	0.3751	0.2712	0.3771	0.2735
1280	0.0959	0.1367	0.0959	0.1372
5120	0.0241	0.0685	0.0243	0.0696
20480	0.0060	0.0343	n/a	n/a
81920	0.0015	0.0171	n/a	n/a

Table 4.2: The differences between the tetrahedral volume metric and the tetrahedral height metric are shown above for both types of generic models.

Face subdivided generic model matching errors (after optimization)

Triangles	Face Subdivision (Method A)			
	Total *volume*	% Improvement	Total *height*	% Improvement
20	0.6554	0.0	1.9661×10^{-1}	0.0
80	0.2286	0.0	5.2426×10^{-1}	0.0
320	0.0625	0.2	2.7022×10^{-1}	0.3
1280	0.0160	0.7	1.3572×10^{-1}	0.8
5120	0.0040	0.9	6.7932×10^{-2}	0.9
20480	0.0010	0.8	3.4023×10^{-2}	0.7
81920	0.0003	0.6	1.7060×10^{-2}	0.5

Table 4.3: Summary of matching errors (tetrahedral volume and tetrahedral height) for the generic model (face subdivision) compared to the ideal ellipsoid *after* simplicial optimization to degree one.

Particle simulated generic model matching errors (after optimization)

Triangles	Particle simulation (Method B)			
	Total *volume*	% Improvement	Total *height*	% Improvement
20	0.6554	0.0	9.1833×10^{-1}	0.0
80	0.2287	0.3	5.2435×10^{-1}	0.5
320	0.0627	0.2	2.7137×10^{-1}	0.8
1280	0.0160	0.7	1.3637×10^{-1}	0.6
5120	0.0040	0.9	6.8623×10^{-2}	0.9

Table 4.4: Summary of matching errors in mm (tetrahedral volume and tetrahedral height) for the generic model (particle simulation) compared to the ideal ellipsoid *after* simplicial optimization to degree one.

Fitting the generic model (Method B) to the test object

Triangles	Total *error*		Mean *height*	
	Volumetric data	Reference	Volumetric data	Reference
20	0.3398	0.6554	0.0181	0.1966
80	0.2872	0.2286	0.0059	0.0171
320	0.2759	0.0625	0.0023	0.0017
1280	0.2792	0.0160	1.0868×10^{-3}	7.4961×10^{-5}
5120	0.2854	0.0040	5.6032×10^{-4}	4.7119×10^{-6}
20480	0.2903	0.0010	2.8475×10^{-4}	2.9492×10^{-7}
81920	0.2923	0.0003	1.4360×10^{-4}	1.8439×10^{-8}

Table 4.5: This tables summarizes the errors in mm (both tetrahedral volume as well as tetrahedral height) in matching the generic model to the volumetric data representation of the test object. The column labeled *Volumetric data* is the actual error and the column labeled *Reference* shows the original error in the generic model as a reference value (taken from 4.1).

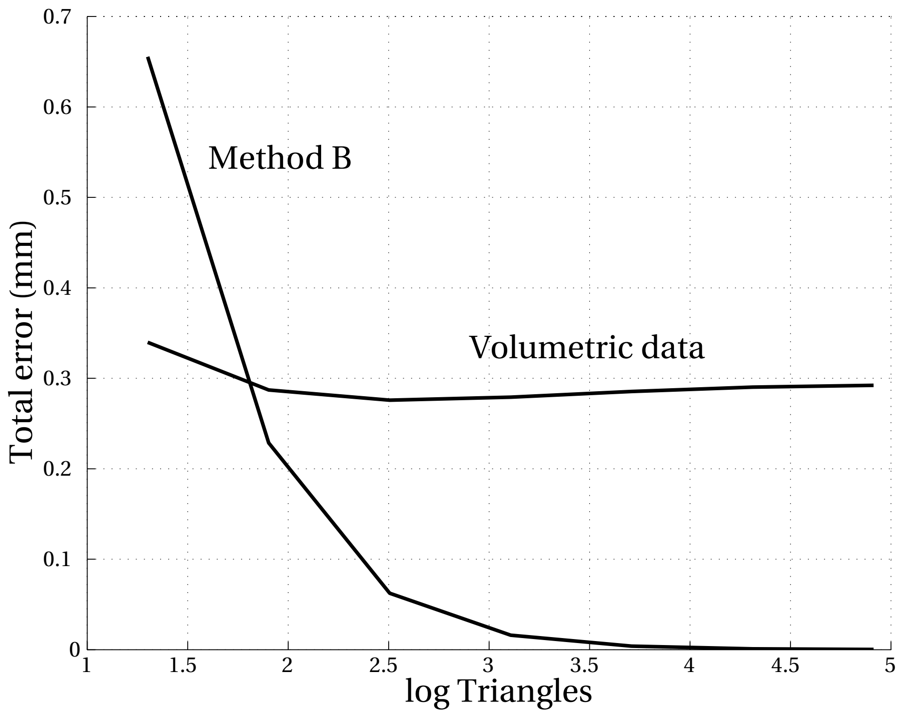


Figure 4.12: Total tetrahedral volume error (in mm) versus model complexity using particle simulation method (Method B).

Chapter 5

Evaluation

To quantitatively test the validity of the dissertation hypothesis, I needed a well-designed evaluation plan to determine (1) what the exact problem was and to what extent it needed to be solved, (2) whether any benefit could be achieved by the proposed solution, and (3) how any such benefit could be measured accurately.

Evaluation techniques can be classified broadly as either *objective* or *subjective*. Objective evaluations rely on quantitative data measurements and statistical analyses. Subjective evaluations rely more on descriptive data (e.g., free-text descriptions, visual inspections, and interviews) and qualitative analysis of such data. In general, both types of techniques are useful, although some solutions can be evaluated more easily with one technique than with the other. Because the goal of this dissertation is to perform vertebral segmentation that is suitable for both visualization as well as biomechanical simulation, I have selected a quantitative evaluation method, which I describe in detail in Section 5.1.

One of the biggest challenges in testing any hypothesis is to create a gold standard against which to compare the results of the proposed solution. Implicit in this process is the ability to accurately measure the deviation of the solution from the gold standard (i.e., error metric). Both the creation of the gold standard and the methodology used to compare it to proposed solutions are described in Sections 5.1 and 5.2. Finally, the quantitative results of the comparision are presented in Section 5.3.

5.1 Evaluation Methodology

The segmentation algorithm developed in this dissertation accepts as input a three-dimensional volume of cervical CT data. The output is a set of *classified* volumes where each vertebral segment is extracted from the original volume and labeled. Evaluating the accuracy of this algorithm amounts to comparing the extracted vertebral segments with gold standard segments (i.e., the best known answer). It is important to note that the gold standard for evaluation is not predetermined, but rather is a function of the test data and the accuracy of the human experts. Although it is reasonable to question the accuracy of the

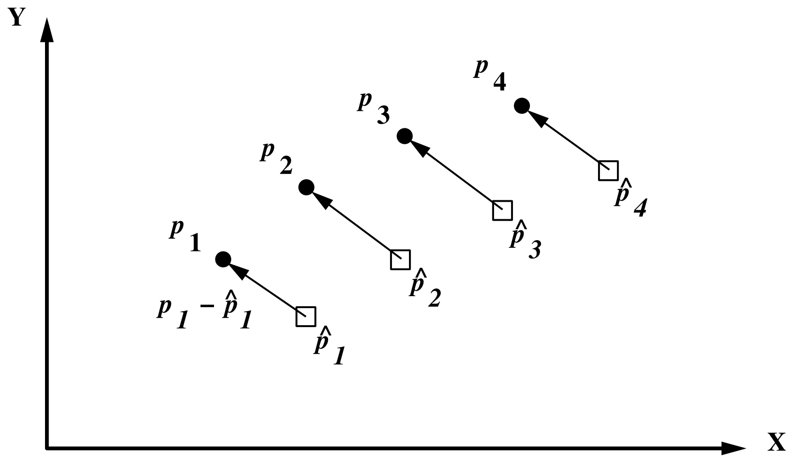


Figure 5.1: The point-based RMS error measures the average squared distance between points in the proposed solution $P = \hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_3$ and points in the correct solution $S = p_1, p_2, p_3, p_4$.

human experts, evaluating that accuracy would involve building physical models with known geometric properties and imaging characteristics. However, it is possible to assess the repeatability, or what is known as the intra- and inter-observer variability, by asking the experts to repeat the process of defining the gold-standard several times. The resulting gold standards can then be compared against one another as a way of assessing their precision.

There are two main types of methods used to evaluate the accuracy of segmentation algorithms. The first type involves the *Root Mean Squared* (**RMS**) error metric; the second type involves *Sensitivity* and *Specificity* ratios.

5.1.1 RMS Error Metric

For evaluating point-, curve-, and surface-based features, the simplest metric to use is the RMS error, which computes the average deviation between points in the proposed solution and points in the gold standard. The RMS error can be computed using a discrete error formula:

$$\text{RMS error} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|p_i - \hat{p}_i\|^2}, \tag{5.1}$$

where p_i is the correct solution and $\hat{p}_i$ is the proposed solution (see Figure 5.1). This error metric is used commonly for measuring the accuracy of a point sample distribution.

For curve- and surface-based solutions, the computation of the RMS error is slightly more complex. Curves and surfaces first must be discretized into a set of finite sample points. Then, the RMS error can be computed as the *minimum* distance between a point in the proposed solution and a point in the actual solution, as represented by finite segments or patches (see Figure 5.2). Thus, the RMS error can be defined

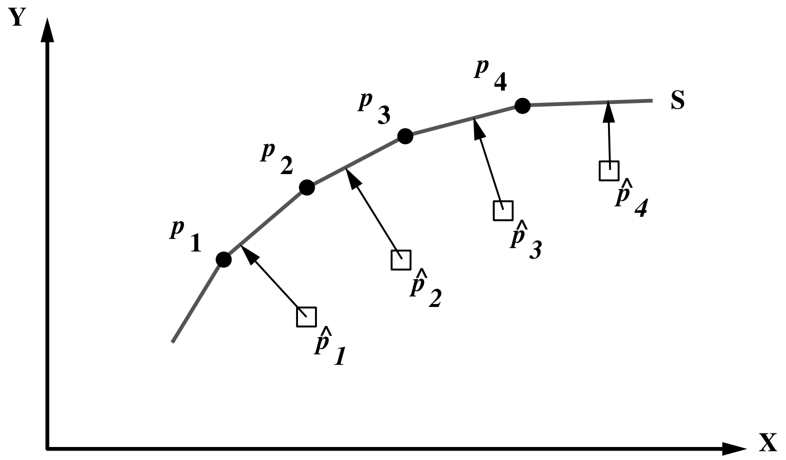


Figure 5.2: The curve- and surface-based RMS error measures the average squared distance between sample points in the proposed solution $P = \hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_3$ and the closest segment (for curves) or closest patch (for surfaces) in the correct solution $S = p_1, p_2, p_3, p_4$.

as

$$\text{RMS error} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[proj(\hat{p}_i, S)]^2}, \tag{5.2}$$

where $proj(\hat{p}_i, S)$ represents the distance of the normal vector projected from point $\hat{p}_i$ onto the correct solution. For assessing the accuracy of the cranial image fusion system, Hemler [39] uses a variant of equation 5.2 where the projection is defined as the normal vector from $\hat{p}_i$ to the closest triangular surface patch in the reference solution. Implicit in this definition is the ability to compute the projected distance in the first place (for complex geometries, this computation may not be feasible). However, if the sampling interval of p_i is sufficiently small compared to the magnitude of the projection vector, the RMS error can be approximated by using the distance between the point *closest* to the proposed solution (see Figure 5.3),

$$\text{RMS error} = \sqrt{\frac{1}{M}\sum_{i=1}^{N} min\{\|p_j - \hat{p}_i\|^2; j \in 1, 2, \ldots, N\}}. \tag{5.3}$$

Here, the quantity M refers to the number of sample points in the proposed solution and N refers to the number of sample points in the correct solution.

Special care must be used in regions where the solution is incomplete (Figure 5.4), or where the solution is superfluous (Figure 5.5). In these cases, parts of the solution are missing in either the proposed or correct solutions. The RMS error is more accurate when there is a *matched* set of points between the proposed and correct solutions, and by itself does not have any mechanism for dealing with unmatched points. The unmatched points can be forced to match the "nearest neighbor" in the gold standard, and those matching error terms that fall outside of a predetermined threshold value are ignored. The determination of the

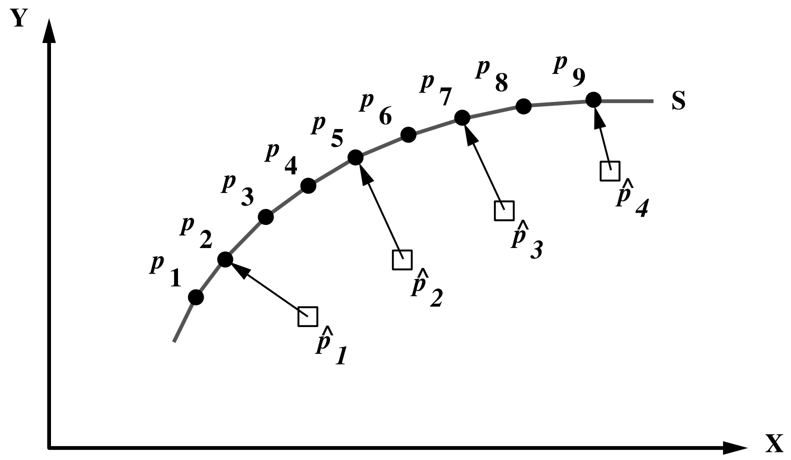


Figure 5.3: When the sampling interval of p_i is sufficiently small compared to the magnitude of the projection vector $p_i - \hat{p}_i$, the RMS error can be approximated by using the distance corresponding to the point p_i closest to the point $\hat{p}_j$.

threshold value can be based on the standard deviation of the error term distribution, which is known *a priori*. In addition, the metric now must take into account the value of N because degenerate cases such as the null solution would produce an RMS error of zero.

The main drawback with the RMS metric is that it relies on a strong isomorphic relationship between the proposed solution and the correct solution. If the proposed and correct solutions differ greatly, or contain regions that cannot be matched with one other, the RMS metric can yield results that are meaningless to the extent that they cannot be used to compare the quality of two competing solutions, nor can they be used as a basis for determining how to improve the quality of a given solution (see Figure 5.6). Figure 5.7 illustrates how the RMS metric fails to accurately measure the error due to a feature that is present in the proposed solution but is absent in the correct solution. The RMS metric can be fooled into reporting a lower error measure for either of two reasons: (1) the extraneous points on the proposed solution do not contribute any error terms (in Equation 5.3, the extraneous points are discarded by the *min* function), or (2) only a subset of the superfluous points are needed to compute the projected distance in equation 5.2.

5.1.2 Sensitivity/Specificity Metric

When dealing with closed contours or closed surfaces, it is possible to avoid some of the drawbacks of the RMS metric by using true positive and true negative statistics as described in Section 2.1.2. Here, the True Positive Rate (TPR) can be defined as

$$\text{True Positive Rate} = \frac{\text{\# of Pixels Correctly Classified as Vertebra}}{\text{Total \# of Pixels in Vertebra}}, \tag{5.4}$$

and similarly, the True Negative Rate (TNR) can be defined as

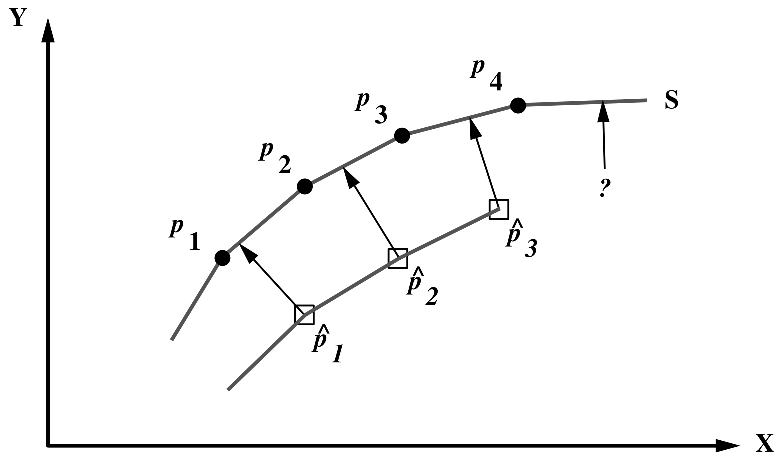


Figure 5.4: Special care must be taken when computing the RMS error where parts of the proposed solution $\hat{p}_i$ are underspecified.

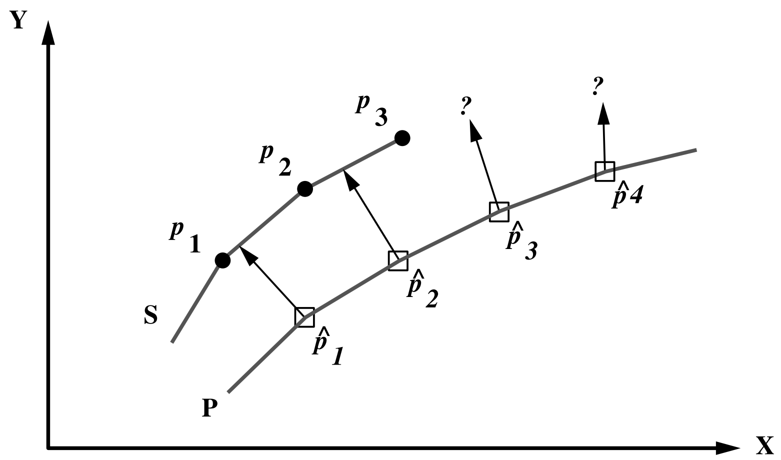


Figure 5.5: Special care must be taken when computing the RMS error where parts of the proposed solution $\hat{p}_i$ are overspecified.

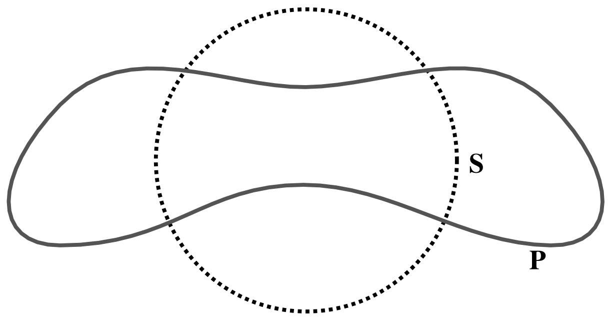


Figure 5.6: The proposed solution P is grossly deviated from the correct solution S. In such a case, the RMS metric cannot provide any meaningful information that can be used for comparing competing solutions or improving the accuracy of a given solution.

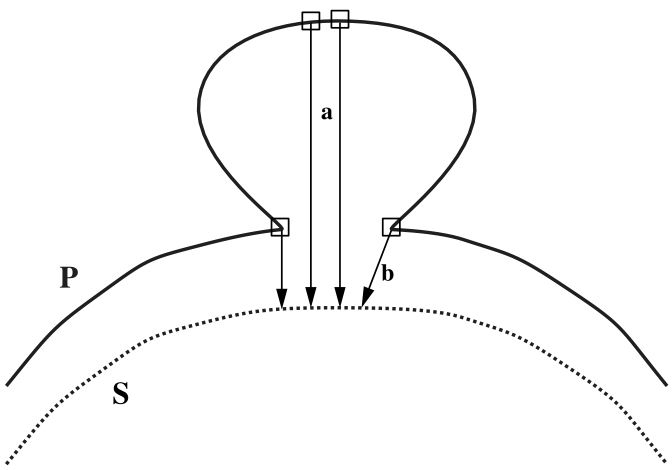


Figure 5.7: The proposed solution P deviates somewhat from the correct solution S. The RMS metric either (a) ignores the superfluous points in P or (b) projects only a subset of the superflous points from P onto S.

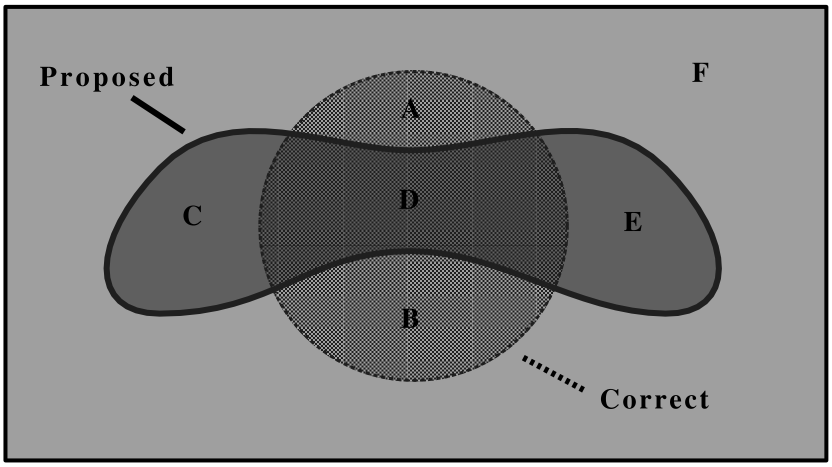


Figure 5.8: The TPR and TNR metrics avoid the problems associated with computing distances between matching features by relying solely upon the amount of overlap among two- and three-dimensional regions between the proposed and correct solutions (solid and dashed outlines respectively). Regions A, B, and D correspond to the correct solution whereas regions C, E, and F are outside the true solution. Hence, region D is a true positive and region F is a true negative, regions C and E are false positives, and regions A and B are false negatives.

$$\text{True Negative Rate} = \frac{\text{\# of Pixels Correctly Classified as } \neg \text{ Vertebra}}{\text{Total \# of Pixels outside Vertebra}}. \tag{5.5}$$

The TPR and TNR metrics avoid the problems associated with finding matching features for computing the error terms. Instead, they rely solely upon the amount of overlap among two- and three-dimensional regions between the proposed and correct solutions (see Figure 5.8). The TPR and TNR metrics are both 100 percent if and only if the two solutions concur precisely (Figure 5.9a). If the proposed solution dominates (i.e., wholly contains) the correct solution, the TNR rate is 100 percent and the TPR is < 100 percent (Figure 5.9b). If the correct solution dominates the proposed solution (Figure 5.9c), the TNR rate is <100 percent and the TPR is 100 percent. When the proposed solution overclassifies the pixels as belonging to $Vertebra$ set, the TPR decreases (Figure 5.9d). When the proposed solution overclassifies the pixels as belonging to $\neg Vertebra$ set, the TNR decreases (Figure 5.9e).

The choice of RMS versus TPR/TNR metrics depends on the type of data being validated. RMS metrics are more appropriate in situations where sample points can be easily determined and easily labeled and matched between the test and reference data. For general two- and three-dimensional region-based data, the TPR/TNR provides a more tractable and intuitive measure of segmentation error. Consider the matching problem shown in Figure 5.10, where the predicted model is designated by the solid line and the correct solution is designated by the dashed line. The predicted model has been shifted relative to the correct model by a distance W corresponding to the base width of the feature triangles. Because each model has two equally sized triangular features and the predicted model appears to overlap half of the correct model, intuition would tell us that the accuracy of this solution is roughly 50 percent (i.e, half the

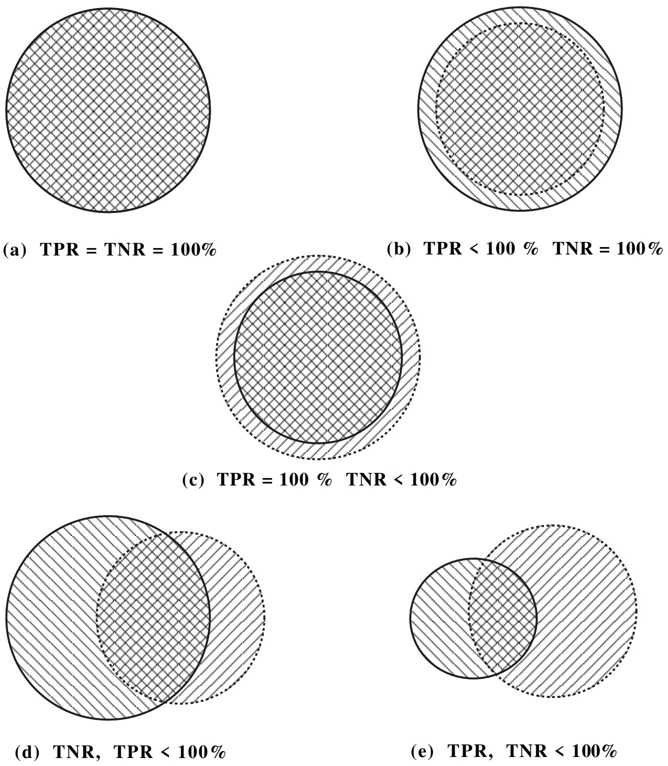


Figure 5.9: The TPR and TNR rates are 100 percent only when the proposed and correct solutions are identical, as shown in (a). If the proposed solution (solid line) is larger and dominates the correct solution (dashed line), the TPR rate will decrease (b). Conversely, if the proposed solution is smaller and is dominated by the correct solution, the TNR rate will decrease (c). If neither solution dominates the other, both the TNR and TPR rates will be less than 100 percent (d,e).

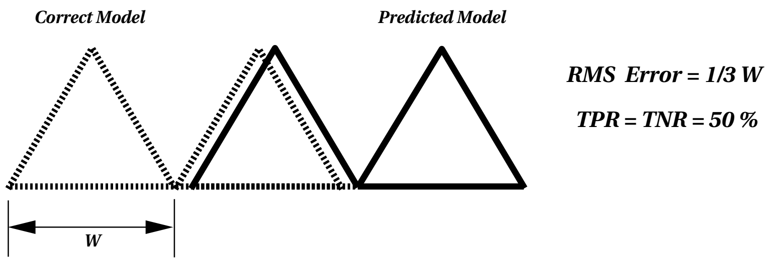


Figure 5.10: The RMS error metric can give biased results when comparing unlabeled features using the nearest neighbor method as shown above. In regions where the surfaces of the proposed and actual solutions overlap, there is no error contribution. As a result, the RMS error is a factor of three lower than the actual error (W). The TPR and TNR ratios are both 50 percent because exactly half of the interior points are true positives while half of the exterior points are false negatives.

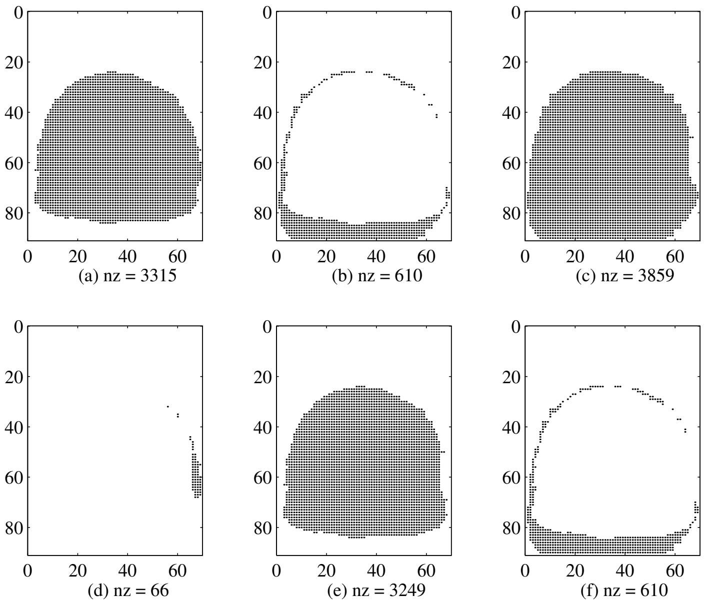


Figure 5.11: The TPR/TNR rate can be computed by first scan converting the gold standard model and the proposed model on a per-slice basis and then comparing the individual voxels in the two image spaces. Figure (a) shows the gold standard model scan converted near the region of the vertebral body. Figure (c) shows the scan conversion of the proposed model at the same location. Figures (b) and (f) show the false positive regions, figure (e) shows the true positive regions, and figure (d) shows the false negative regions.

features have been matched and the solution is shifted by one whole feature width). The TPR and TNR rates for this example are both 50 percent, which is consistent with the intuitive notion of accuracy used here. On the other hand, the RMS error metric using the "nearest neighbor" rule reports an error of $W/3$ (three of the six sides of the model overlap very closely and contribute negligible error, two of the sides have an average deviation of $W/2$, and the most distant side has an average deviation of W). Intuitively, we would expect the RMS error to be approximately W instead of $W/3$ because that is the amount of shift between the two models. In the absence of matching points, the RMS error metric will be biased toward a lower figure. Due to the number of vertices involved in the FEM models of the vertebrae, performing a labeled match is very difficult. Therefore, the best metric to use from both a feasibility and a tractibility perspective is the TPR/TNR metric. This metric can be computed easily by scan converting the interior regions of the surface models and performing a voxel-by-voxel comparison between the test and reference models (see Figure 5.11). This scan conversion process is performed on a per-slice basis for all the slices in the region of interest (typically the minimum bounding box that contains both surface models).

5.2 Evaluation Procedure

For this evaluation, the bone classifications made by a group of experts in radiology and neuroanatomy served as the gold standard. Test volumes were fed to the extraction algorithm and the resulting segmentation output was compared with the two-dimensional segmentation performed by the experts.

There were five major steps involved in the evaluation procedure: (1) acquiring suitable data, (2) outlining vertebral boundaries and landmarks, (3) building the generic model, (4) extracting vertebral boundaries from test data sets, and (5) comparing the system's results with the experts' results. These five steps were performed in two distinct phases (see Figure 5.12). Phase I (image acquisition, classification, and model construction) required assistance from radiologists so that the three-dimensional CT and MR images would be acquired in accordance with the scanning parameters set forth by the **SpIF** standards (see Appendix G for further details). After these images were acquired, the experts then identified and outlined the landmarks and vertebral segments. These outlines formed the gold standard for phase II (system segmentation and comparison) and also were used to build the generic models in phase I.

PHASE I: Acquistion, Classification, and Model Construction

Acquire $N = k \times m$ 3-D CT and MR data sets
For each data set D_n
    For each cervical segment C_j:
        Expert: Identify landmarks on C_j using D_n
        Expert: Outline vertebral boundaries on C_j using D_n
        Generate 3-D AFEM M_n of C_j using Expert boundary data

PHASE II: Segmentation and Comparison

Create k partitions each with m data sets
For each partition P_i:
    Reserve P_i for test set
    Combine all $M_n(n \ni P_i)$ models to generate test model M_i
    For each cervical segment C_j in P_i:
        System: Extract landmarks on C_j using P_i
        System: Outline vertebral boundaries on C_j using P_i
        Compare System landmarks with Expert landmarks
        Compare System vertebral boundaries with Expert landmarks
Combine RMS statistics for all partitions

Figure 5.12: Outline of evaluation procedure.

As is true for any classification method, it is important to keep the data used for training the model separate from the data used to evaluate the model. However, because of the time and cost associated in acquiring actual patient data sets, I was only able to acquire 12 cervical spine data sets (both CT and MR) over the course of the evaluation study. Because splitting this limited amount of data into separate fixed test and training categories would have resulted in an insufficient number of test and training cases, I used a *cross-validation* technique [83]. Briefly, the cross-validation method partitions a collection of D data sets into k groups each with m data sets for a total of $N = k \times m$ patient studies (see Figure 5.12). The model is trained using $k - 1$ groups but tested using the group P_i excluded from the training set. In this case, training the model involved generating a three-dimensional surface mesh for each cervical segment C_j for $j \in \{1, 2, ..., 7\}$ using the segmented boundary information from the experts. The boundaries were not used to generate the mesh directly. Instead, they were used to isolate the bony segments from one another and from the surrounding tissues. The isolated segments were then fed into the tessellation and mesh optimization routines. Each mesh was then augmented with the expert-extracted landmarks to form the complete AFEM for each C_j.

The model was retrained k times and each time a different group was reserved for testing. Finally, the results from the k different trials were combined to form an estimate of the true performance of the system. A modification of the cross-validation technique known as *boot-strapping* [83] allows the training partition to include multiple instances of the same data set. This technique can sometimes produce more accurate assessments in situations when both k and m are very small. For cross-validation, the number of training suites cannot exceed k, whereas for boot-strapping the number of suites is equal to $(N - t)^s/s!$ where s is the number of samples used to build the training set and t is the number of samples used for the test set.

By using the gold standard to build the generic model, it may appear that there is a violation of the principle of separating the training sets from the test sets. Recall that each partitioning of the data sets used to build the model M_i will, in turn, be excluded from the cross-validation test sets. That is, M_i is constructed from the set $D - P_i$. It would not be fruitful to throw away the expertise used in building the gold standard. That knowledge can be used to construct the generic model without compromising the goal of having an independent test set. This is analogous to the procedure used to perform cross-validation tests on neural networks [61]. Using anything other than the gold standard to build the generic model would amount to testing the accuracy of the non-expert model builders instead of testing the accuracy of the model-driven segmentation procedure.

In theory, for each test partition P_i, a composite model M_i could have been created using the information from the $N - t$ models excluded from P_i. But because of the difficulty in combining multiple gold-standard models into a single generic gold standard, I instead chose to create multiple generic models, one from each data set. Hence the number of partitions k equals the number of data sets D. Each generic model is then tested on the remaining $k - 1$ data sets, yielding a maximum of $k(k - 1)$ test cases. For $k = 12$, this would have resulted in a total of 132 total test cases, a prohibitively large number considering the amount of time required to perform each test. Because of the time constraints, I randomly chose 12 generic-model–test-data pairs to perform the evaluation.

5.3 Quantitative Results

Twelve CT and MR data sets were acquired for this evaluation (labeled P001 through P012). Eight of the data sets had sufficiently high resolution in the z-dimension (better than 1 mm) to be useful for generating gold-standard models (see Appendix F for full descriptions of the data set parameters). To maximize the number of test cases, all the gold-standard models were constructed from the fourth cervical segment (C4). Although it would have be useful to test the method on other segments as well, this would have required generating an inordinate number of models and test cases.

Each vertebra was decomposed into eight distinct regions (see Figure 5.13) and the matching results were reported individually for each of these regions. This enabled the extraction algorithm to be executed on a multiprocessor parallel machine to reduce the total running time for obtaining the solution. This also provided additional insight into where the extraction method worked well and where it did not. The boundaries of the zones for each vertebra are computed as a function of the landmark positions in the following manner:

$$x_l = 0.8\ AFL_x + 0.2\ AFR_x \quad (5.6)$$

$$x_u = 0.2\ AFL_x + 0.8\ AFR_x \quad (5.7)$$

$$y_l = \frac{(AFL_y + AFR_y)}{2} \quad (5.8)$$

$$y_u = 0.9\ SCT_y + 0.1\ y_l \quad (5.9)$$

$$y_m = \frac{(y_l + y_u)}{2} \quad (5.10)$$

where AFL, AFR, and SCT are the left arterial foramina, right arterial foramina, and spinal canal trough as defined in Section 3.2.3. The values x_l, x_l, and y_m partition the data space into six parallelepiped regions (zones 1 through 6). The remaining two zones (7 and 8) were formed by cutting a rectangular notch out of regions 1 and 3, where the corners of the notch are defined by AFL and AFR (see Figure 5.13). Zones 5, 7, and 8 are considered *threshold-classify* zones because the vertebrae can be extracted unambiguously using just thresholding.

The nine extraction algorithm parameters shown in the following chart were fixed for all of the test

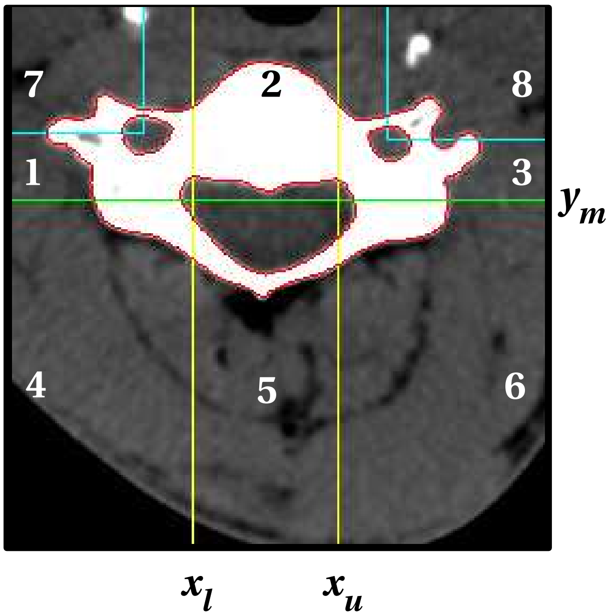


Figure 5.13: The eight classification zones are depicted above and extend in the positive and negative z-direction as well. Zones 5, 7, and 8 are threshold-classify zones because the vertebra can be extracted using simple thresholding with no ambiguity.

cases (these parameters are described fully in Chapter 3).

Parameter	**Value**	**Description**
L_s	30 mm	search bracket limit (see Sec. 3.6)
α	4.0	angular weight (see Eq. 3.65)
β	0.1	linear weight (see Eq. 3.65)
ω_T	0.9 rad	angular coherence limit (see Eq. 3.39)
ω_C	0.9 rad	curvature coherence limit (see Eq. 3.42)
d_o	3	outlier contagion depth (see Sec. 3.6.2)
$\triangle t$	0.004 sec	relaxation time step
T_R	12 sec	total relaxation time
σ_a	.1 mm	triangle altitude limit (see Section 3.4.2)

Extraction parameter values

The aggregate true positive and false negative results for each zone in the 12 cases are summarized in Table 5.1. Zones 5, 7, and 8 have no false positives or false negatives because the extracted voxels are masked with the thresholded data values. Except for these zones, the most accurate results were obtained in zones 4 and 6, corresponding to the two facet regions. Region 2, corresponding to the vertebral body, had the lowest accuracy due to the high degree of ambiguity near the disk surface and bone interface.

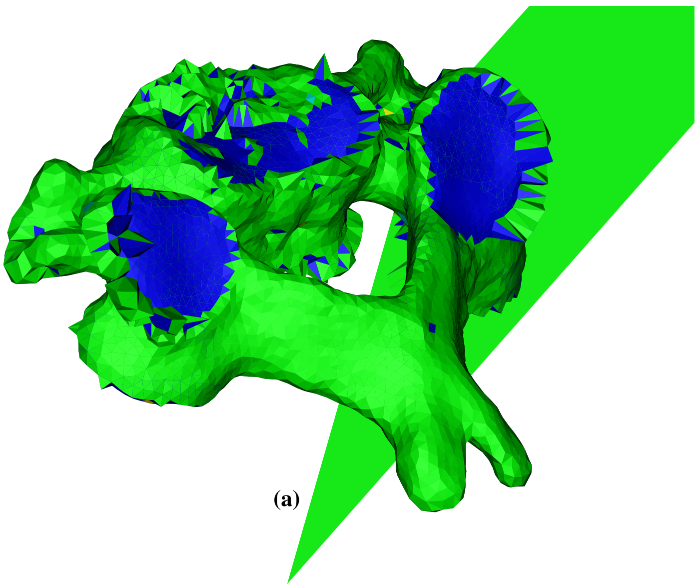
Aggregate Matching Accuracy, N = 12

Zone	TP	FN	Sensitivity %	Specificity %
1	161621	14203	92.6	99.6
2	346295	28088	91.6	99.1
3	154538	19235	94.7	99.4
4	101659	7535	95.5	99.8
5	185322	0	100	100
6	90847	13842	96.9	99.6
7	26756	0	100	100
8	26017	0	100	100
Total	1093055	82903	**93.0 %**	**99.3 %**

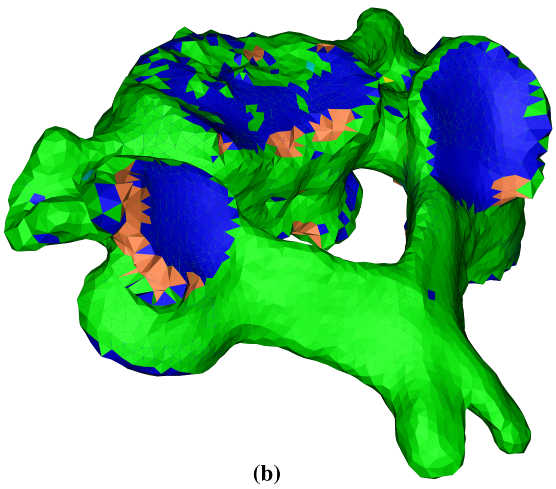
Table 5.1: Total true positive and false negative voxel counts for each of the eight zones for the twelve test match trials along with their corresponding sensitivity and specificity ratios.

The specificity of the extraction algorithm in all the zones is quite high (> 99 percent). This is due to the fact that the extraction algorithm knows from the model how to limit the classified voxels to the interior regions of the model's surface and ignores all voxels in the exterior regions. Figures 5.14 and 5.15 show typical extracted vertebra from data sets P002 and P004.

Because the matching algorithm computes a score for each match, if multiple generic models are matched to a given data set, it is possible to select those matches that yield the highest scores for a given zone. It seems reasonable to hypothesize that the accuracy of the extraction algorithm can be improved by selecting those matches with the highest scores and combining them to form a composite solution. To test this sub-hypothesis, data set P002 was matched with models derived from data sets P004, P005 and P006, while data set P006 was matched with models derived from data sets P002, P004, and P005. The results

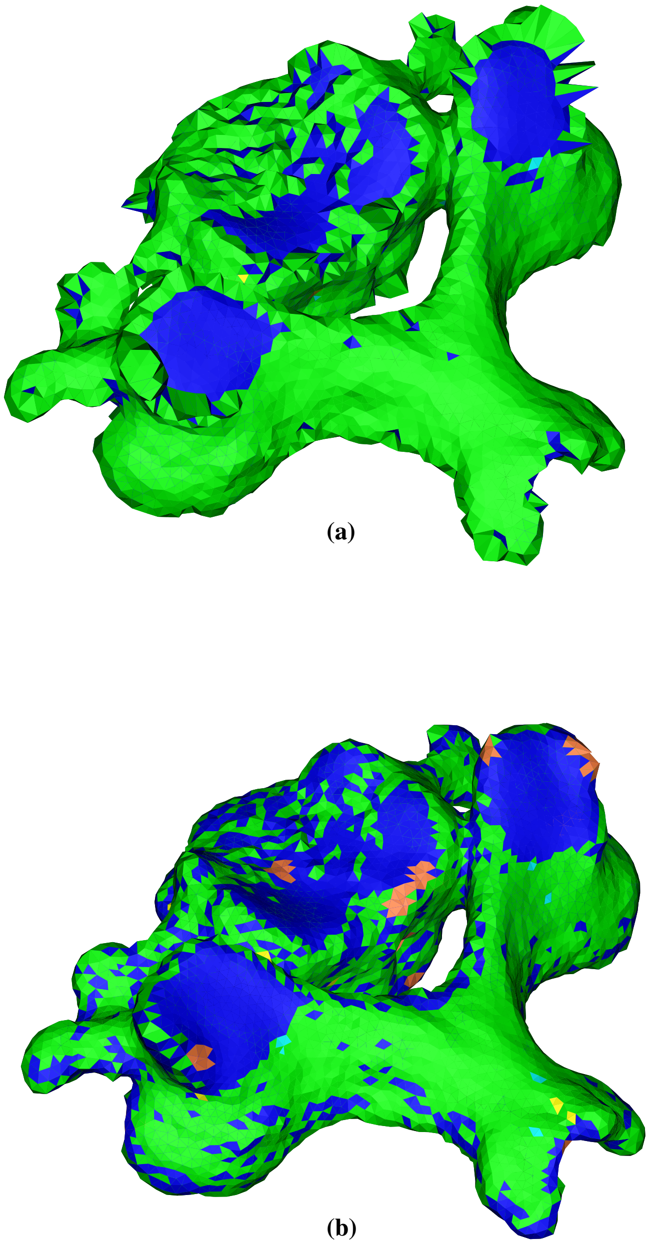


(a)



(b)

Figure 5.14: Shown above is the patient-specific model for data set P002 before the relaxation (a) and after the relaxation process (b).



(a)

(b)

Figure 5.15: Shown above is the patient-specific model for data set P005 before the relaxation (a) and after the relaxation process (b).

of these score selected matches are summarized in Table 5.2. The overall sensitivity was increased from 93 to 94.1 percent and the overall specificity was increased from 99.3 to 99.9 percent. These test cases do suggest that the accuracy of the extraction algorithm can be improved by matching multiple generic models to a given data set, and selected the matching zones that yield the highest scores. Although it is beyond the time contraints of this research, this *maximal score-selected* method merits further evaluation using a larger sample size.

To assess the precision of the gold standard, I repeated the manual extraction of the C4 vertebra from data set P004 and found the average sensitivity and specificity of the self-matched models to be 99.4 and 99.9 percent respectively. This suggests that the sensitivity of the automated segmentation algorithm is within 5 percent of the best human-expert-extracted process.

5.4 Summary

For computing the error in extracting the vertebrae from CT images, I selected and implemented an error metric based on the false positive and false negative classification rates. The alternative RMS point-based error metric was rejected because it underreports the error when using unlabeled (i.e., unmatched) data points. Because of the limited number of data sets available for the evaluation, I used a cross-validation technique to assess the performance of the model-driven vertebral-extraction procedure. I constructed eight gold-standard vertebra models, which I then applied randomly to 12 data sets. The results of this evaluation showed that the vertebrae can be extracted with an overall sensitivity of 94.1 percent and overall specificity of 99.9 percent. That is to say, within the volume of interest, 94.1 percent of the vertebra of interest can be correctly identified and 99.9 of the non-vertebra regions can be correctly identified. This compares with an inter-observability sensitivity of 98.9 percent and specificity of 99.9 percent. For practical reasons, it was easier to compute the metrics using two-dimensional cross-sectional slices rather than three-dimensional data sets. Because radiologists are trained to work with two-dimensional data, this strategy yielded more reliable comparisons.

As would be expected, the algorithm was least accurate in regions such as the vertebral end plates, where the ambiguities were greatest (approximately 92 percent sensitivity and 99 percent specificity). Further evaluations should be conducted using a larger sample size to verify the consistency of these results.

Maximal Score Selected Matching Accuracy, N = 6

Zone	TP	FN	Sensitivity %	Specificity %
1	32333	1774	87.8	99.9
2	19432	2652	92.3	99.9
3	28973	887	82.3	99.9
4	24526	5047	99.0	100
5	37052	0	100	100
6	33772	1361	96.4	99.9
7	6312	0	100	100
8	4822	0	100	100
Total	154889	9947	**94.1 %**	**99.9 %**

Table 5.2: Maximal score selected true positive and false negative voxel counts for the six test match trials in which multiple gold-standards were matched for a given data set (along with their corresponding sensitivity and specificity ratios).

Inter-observer accuracy, data set P004

Zone	TP	FN	Sensitivity %	Specificity %
1	4367	61	99.7	99.9
6	3091	732	95.2	99.9
3	5047	55	99.7	99.9
4	5168	7	99.9	99.9
5	0	0	100	100
2	11396	44	99.9	99.9
7	0	0	100	100
8	0	0	100	100
Total	150157	900	**99.4 %**	**99.9 %**

Table 5.3: Total true positive and false negative voxel counts for the inter-observability trial using the gold-standard derived from data set P004 (along with the corresponding sensitivity and specificity ratios).

Chapter 6

Summary and Future Work

In this dissertation, I have demonstrated that a set of finite-element models can be used to segment vertebrae from volumetric CT data sets of the cervical spine. Furthermore, I have demonstrated that such a FEM-based segmentation technique has a true-positive classification rate of 94 percent and a false-negative classification rate of 0.1 percent. The FEM approach offers several distinct advantages over both non-model-based methods (such as thresholding and edge detection) as well as model-based methods (such as active contours). First, the FEM incorporates high-level *a priori* information about the region to be segmented and thus serves to guide the low-level surface-detection algorithm. Second, the FEM provides information that augments and disambiguates the low-level segmentation process in those regions where the vertebral surface information is missing or open to multiple interpretations. Third, the FEM provides a geometric description of the segmented data so that more sophisticated model-based visualization methods can be used to view the patient data sets. Finally, the FEM is a starting point for building sophisticated patient-specific biomechanical models that can be used for diagnosis, treatment, preoperative planning, and outcomes simulation for various interventional options.

The matching technique developed here not only utilizes high-level information captured in the model, but it also estimates the quality of the extraction process through the computation of the matching score. This score enables the algorithm to determine how closely the patients' vertebrae match the stock of prebuilt generic models. Thus the algorithm can assess both the degree of the match as well as the type of match. As more generic models are built, the algorithm automatically increases its knowledge of both the normal as well as the abnormal variations to expect in the patient population. With a suitably large database of generic models, this matching technique could possibly serve as a diagnostic aid as well.

6.1 Research Contributions

The original contribution of this research is that it provides a highly automated method for extracting vertebral segments from CT images. Prior to this research, no such methods, either automated or semiautomated, have been developed to perform such a task.

Another contribution of this research is that it supports the overall goal of developing a technique for performing fusion of CT and MR images of the cervical spine. The significance to the medical community is that it will improve the usefulness of CT and MR imaging modalities by providing diagnostic capabilities that are not presently available in routine clinical settings.

Finally, the long-range contribution of this research is that it will put forth the first unified approach for solving the combined problems of feature extraction, biomechanical modeling, and three-dimensional visualization (by using the FEM as the underlying basis for representing structural information). Through this research, I endeavored to show that finite-element models can provide a very useful framework for synthesizing many of the existing approaches to image understanding, such as edge detection, thresholding, and region growing. Although each of these techniques has proven to be useful in certain application areas, they all suffer from the drawback that they cannot work in a highly automated fashion. The FEM-based approach will make it feasible for many of the existing methods to work together in a highly autonomous manner.

6.2 Generality of the Method

Because I wanted the segmentation technique to be easily generalized to other areas of computer vision, I made as few domain-specific assumptions as possible about the generic models or the data sets to be extracted. The key assumptions I made were that (1) most of the surfaces of the vertebrae (approximately two-thirds) can be detected either ambiguously or unambiguously using a low-level segmentation algorithm (e.g., thresholding or edge detection), (2) the variability of the actual vertebrae can be captured with a modestly sized database of generic FEM's, and (3) a minimum of four non-colinear object landmarks can be detected in the data sets either manually or in an automated fashion.

The generality of this method rests primarily upon the degree to which these three assumptions can be met. Conditions 1 and 2 are not categorical, but they affect the quality and accuracy of the segmentation results. The more closely these conditions are met, the more accurate the solutions will be. Condition 3 is the only categorical condition, which must be satisfied in order for the segmentation method to work as described in this dissertation. If this condition cannot be met, it is still possible for the method to work provided there is some other means of determining the elements of the homogeneous transformation matrix that performs the initial coarse match of the generic model to the data. However, the optimization algorithm modifies the elements of this matrix during the local surface-matching step. As a result, these elements do not have to be determined with a high degree of precision.

Keeping this in mind, it should be straightforward to apply this segmentation method to other volumetric image analysis problems where these key assumptions can be met. Needless to say, further testing of this method is warranted to assess its sensitivity to these conditions.

6.3 Future Work

As is true for almost any dissertation, there are many opportunities for extensions and refinements of the methodology presented here. The usefulness of this methodology could be greatly enhanced in the following areas:

- Improving the sensitivity of the segmentation technique by using post-processing techniques such as region growing and other morphological operators. In areas where there are good matches, a constrained region-growing operation could be used to include voxels that have been conservatively bounded by the surfaces of the model.
- Creating composite generic models out of sets of several similarly shaped models. This would reduce the total number of generic models needed but might also reduce the overall accuracy of the segmentation process.
- Incorporating more of the match parameters into the optimization loop. During the surface-matching step, only the homogeneous transformation parameters were optimized. Other parameters such as the search bracket limit, the angular coherence limit, and the curvature coherence were held constant within the optimization loop. These and other parameters described in Section 5.3 could also be viewed as variables to be optimized (albeit at a much greater computational expense).
- Evaluating the accuracy using a larger sample size. The sample size used here was limited primarily by the available time and other resource constraints. A larger number of test cases and a larger number of generic models would improve the accuracy of both the segmentation method and the evaluation process itself.
- Evaluating the performance of the segmentation method on MR data sets. Again, due to resource constraints, the evaluation was limited to spinal CT images. Evaluating the performance on MR images would also reveal the accuracy of the overall CT/MR image-fusion process.
- Incorporating soft tissues such as ligaments, muscles, and disks into the FEM. To be useful for biomechanical simulation, connective tissues and supportive elements need to be incorporated into the spine model. With a more complete spine model, it should be feasible to perform static and dynamic simulations on individual patients using their imaging data.

The extent to which these refinements will be explored is a function of the clinical applications that depend on these methods. Although sophisticated finite-element models have been used in biomechanical studies for quite some time, the addition of high-quality imaging data and increasingly powerful desktop supercomputers makes it feasible to develop computational methods that will provide greater clinical insights into the assessment and treatment of structural disorders. Hopefully, this will propel the nascent field of computational medical informatics into a much larger field of clinical applications. In addition, these new computational methods will provide support for modeling and simulation of biological systems that will be useful for a wide variety of basic research and educational endeavors.

Appendix A

Essential Spinal Anatomy

Contrary to popular perception, the spine is not simply a passive skeletal structure consisting of vertebral segments, disks and a spinal cord. But rather, it is an active, dynamic structural entity with a variety of adaptations designed for its manifold duties. It can be thought of as a complex tapestry of bones, disks, ligaments, cartilage, muscles, tendons, nerves, and blood vessels. Every element plays a critical role in the overall function of the spine. Even the shape of the spine itself is essential for motion in the upright posture. Some authors prefer to use the term “spinal engine” rather than refer to it as simply the “spine”. Although the human spine is a remarkable evolutionary adaption to the demands of upright human mobility, like any other complex system, the spine has many inherent modes of failure (pathologies). The most common pathologies are caused by traumatic injuries (e.g., sports related or motor vehicle accidents), self-inflicted abuse or neglect (e.g., inadequate exercise and nutrition, improper posture or improper lifting of heavy objects), or simply old age (osteoporosis, arthritis, etc). Because various tissues of the spine are also richly innervated - a fact which explains why most disorders of the spine involve severe or chronic pain.

Before delving into the essential anatomy of the spine, it is useful to first understand some of the important functions of spine. The following is a partial list of some of these functions (not ranked in any order of significance):

- Provides a structural underpinning for the upper body, neck and head.
- Supports the skull and provides attachment points for many muscles and ligaments needed for movement of the head and neck.
- Acts as a load-bearing shock absorber which cushions the head from the jarring accelerations caused by walking, running and jumping.
- Provides a strong, yet flexible protective enclosure for the spinal cord and the blood vessels which nourish all the elements of the spine.

- Provides attachment points for various muscles of the thorax, shoulders and abdomen. These muscles are responsible for upper body movement, balance and upright posture.
- Acts as an internal achorage point for mechanical operations such as lifting, pulling, pushing, and twisting.
- For human beings, the spine provides a stable column needed for mobility in the upright posture.

Not only must the spine perform all the functions mentioned above, but it must perform many of these functions simultaneously. For instance, the spine is capable of flexion and rotation without constriction of its blood vessels or nerves. It must also act as a strong, yet light-weight vertical column with static as well as dynamic stability. Furthermore, it must endure an endless series of shocks and stresses and still provide a lifetime of reliable service (with proper care). Like any other system in the body, awareness of the mechanical limitations of the spine and proper preventive maintenance are essential requisites for prevention of problems. Given all the functions that the spinal system is responsible for, it is little wonder that years of neglect and abuse are often repaid in the form of chronic pain.

Spinal Structure

The spine can be separated into 24 distinct vertebral segments along with five fused bony segments which form the sacrum and an additional four fused segments which form the coccyx (the tail bone).

Although the spine generally operates as an integrated unit, the vertebral segments are grouped into three distinct regions with each region displaying its own particular form as well as function. The lowest region is called the *lumbar* spine and one of its primary roles is to act as a structural base for the upper body and provide support for the upright posture of the human frame. The five segments of lumbar spine are the largest and thickest of the 24 vertebral segments and are therefore best suited for supporting the weight of the entire upper body. The middle 12 vertebral segments form the *thoracic* spine which is the least flexible of the three regions. The important roles of the thoracic spine are to support the rib cage and to dissipate much of the shock transmitted through the spine. The thoracic spine and the rib cage together form a solid, protective enclosure which helps minimize damage to the heart, lungs, major blood vessels and bronchial passages. The remaining seven vertebral segments which make up the uppermost region are collectively referred to as the *cervical* spine.

The major roles of the cervical spine are to support the skull and to provide the flexibility needed for maximum articulation of the head. The seven vertebral segments of the cervical spine are noticeably smaller and thinner than the lower 17 segments. The upper two segments of the cervical spine (known as the atlas and the axis) are even more distinctively shaped than the remaining five lower segments (C2 through C7). They are distinct enough to lead one to question the usefulness of labeling them as cervical segments. The broad and flat shape of the atlas (C1) acts a large and highly leveraged attachment base for the muscles and ligaments which connect the skull to the spine. The atlas can therefore be viewed as the interface between the skull and the spine. The many muscles that connect the atlas to the skull are highly innervated. As a result, these muscles are capable of very fine and precise movement of the head - an important mechanical property whichs lends support to the visual system. The axis (C2) acts a pivot point for the atlas and contains a cylindrical, spindle-like process which actually extends above the atlas. The remaining five cervical segments more closely resemble the thoracic and lumbar segments and are distinguished mostly by their smaller size and wider aspect ratio.

Appendix B

Image Registration

In this appendix, I will review three techniques researchers have used in the past to register CT-to-MR or CT-to-CT images. Once the matching features have been detected on a set of images, it is then possible to register these images. In order to understand the motivation for the development of these techniques, I will first describe the general characteristics of registration systems.

B.1 Image Data Transformations

Registering two image data sets always involves an operation that changes the locations of voxels in one data set to line up with corresponding voxels in the other set. The task of registration is to determine how to *transform* the first data so that it matches the second. The transformation operation can involve translations, rotations, scaling, and warping. There are three types of transformation operations (see Figure B.1) in common use:

1. *Rigid-body*. This transformation only allows translations, rotations, and possible scaling of the image sets (Figure B.1a). The objects in the image are treated as if they are made from non-pliable material such as granite. Rescaling operations are permitted if the image sets have not been calibrated in the x, y, and z dimensions, or if the calibration factors are not known.

2. *Affine*. This is similar to rigid-body transformation except that sheering operations along one or more dimensions are also permitted (Figure B.1b). Sheering can be viewed as a *non-orthonormal* rescaling operation. Both affine and rigid-body transformations can be performed using matrix multiplication operators.

3. *Elastic*. This is the most general transformation operation in which any well-behaved domain mapping function can be used (Figure B.1c). Most elastic transformations try to conform to a set of physical constraints such as continuity, smoothness, and minimal energy state.

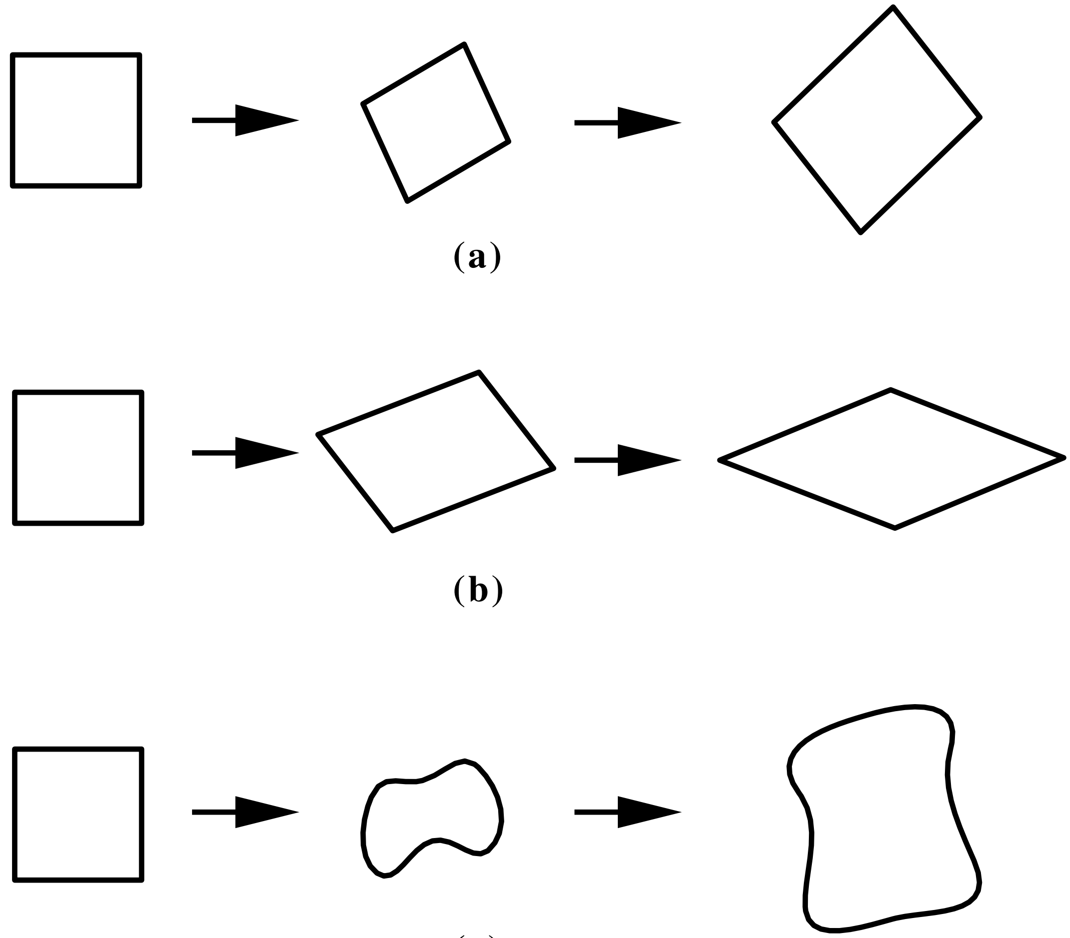


Figure B.1: Three commonly used transformations are (a) rigid-body, (b) affine, and (c) elastic.

B.2 Registration Techniques

Registration techniques can be categorized broadly as being either *prospective* or *retrospective* in nature. Prospective techniques use data sets that are acquired with the intent of being used for registration. Retrospective techniques use data sets that were not acquired with any knowledge of future registration goals. Each of these techniques has its own unique set of requirements:

Prospective registration. These techniques require a *fiducial reference system* to be attached to the patient's body (usually the head) before performing the scans. The purpose of the fiducial reference system is to provide a set of image *marker coordinates* that can be used to establish a correspondence among measurements made in the different image modalities. There are two common types of fiducial reference systems:

1. *Fiducial frame.* This system consists of a boxlike frame that is mounted to the patient's head. The frame contains thin rods that consist of materials that are either radiopaque or exhibit high-contrast resolution on the resulting images (without introducing any distortions). Fiducial frames have two major advantages: (1) the fiducial points are consistently visible on all slices, and (2) the reference coordinate system can be reconstructed more accurately because there are more reference points available. There are also disadvantages of a fiducial frame: (1) patients find it painfully uncomfortable, (2) physicians don't like to go through the process of attaching the frame, and (3) the frame is bulky and can be worn only for a limited time, usually requiring scans and operations to be performed on the same day.

2. *Fiducial markers.* This system consists of several objects (markers) that are implanted on the surface of the skull. The markers are visible as points, but only on certain slices. These slices are used to determine the reference coordinate system. Fiducial marker systems have three major advantages: (1) they are generally less bulky, less invasive, and less painful for the patient, (2) markers can be left in place for longer periods of time, and (3) they do not interfere with the patient's normal activities. Fiducial marker systems have two major disadvantages: (1) they are less accurate because only four or five markers are typically used, and (2) the slice plains have finite thickness and separation, so not all markers may image correctly and the patient may have to be rescanned.

Because the precise geometry of the fiducial reference system is known beforehand, distances between fiducial markers that appear in the scanned images can be easily calibrated. Once that has been done, the registration procedure can compute a set of transformation parameters. To determine these transformations, the fiducial markers must be distinguishable from the surrounding tissues on all the image modalities.

Restrospective registration. These techniques are used when a patient is scanned without the benefit of a fiducial reference system (either by omission or commission). Multi-modality images are registered afterward by performing transformations of the data set from one modality to match the data set of the other modality. There are three major approaches to retrospective registration:

1. *Global registration*. An affine transformation is performed on images from one modality to match images from the other modality (translation, rotation and scaling operations on the data as a whole). The advantages of global registration are (1) it be performed very quickly, (2) it provides a good visual fit of the data sets, (3) it doesn't normally require recognition of anatomical features, and (4) it can be useful for correcting gross errors in alignment, scaling, and rotation. But it suffers from two major drawbacks: (1) it doesn't necessarily provide accurate alignment throughout the image, and (2) it cannot correct for non-linear distortions present in images.

2. *Local registration*. Elastic deformation operations are performed on one image modality to make it match the the other. Elastic deformations can be both non-linear and non-global and are usually performed after the global registration step.

3. *Global plus local registration*. A global registration step is performed first on two the data sets, followed by local registrations steps applied to selected regions in the image.

B.3 Elastic Matching Techniques

Elastic matching refers to a registration technique in which one geometric model or image is warped in such a way as to match, as closely as possible, another geometric model. Such techniques can be useful for both model-based segmentation and registration procedures. Bajcsy and Kovacic have developed such a technique for matching three-dimensional images obtained from sequential CT and photographic studies of the human head [4]. Their technique is completely retrospective (i.e., it does not require the use of any registration apparatus at the time of scanning) and, because it is three-dimensional, it is independent of the slice plane orientations. The input to their algorithm consists of a pair of three- dimensional volumetric data sets obtained from the two image modalities. The algorithm then deforms the photographic data set (which they refer to as a brain atlas) in order to make it geometrically similar to the CT data set. Hence, the CT data set is treated as a reference that the brain atlas is matched against. The deformation process is analogous to stretching a piece of rubber in various directions and at various locations without tearing or folding the rubber. In fact, their algorithm models the brain atlas as an elastic body and applies external forces, that are balanced by internal forces that resist any stretching. The desired geometry is found when the cost of deformation outweighs the cost of similarity. That is to say, the desired solution is obtained by minimizing the net cost function,

$$Cost(net) = Cost(deformation) - Cost(similarity). \tag{B.1}$$

The cost of deformation is a function of the total external forces applied to the object, whereas the cost of similarity is a function of how well certain anatomical features are matched in the two image sets. This type of cost function was originally proposed by Broit [17] in his earlier work with image registration.

This image matching technique differs from previously published techniques in several ways. First, the authors draw upon principles of elastic modeling used in disciplines such as materials science, mechanical engineering, and structural engineering [73]. Second, they use a combination of global as well as local deformations. The global process minimizes any errors due to differences in scaling, translation, and rotation of the two image sets. The local process tries to minimize differences at the level of individual organs without perturbing the global matching process. Finally, their technique operates at three different levels of resolution (multiresolution). The local deformation starts from the coarsest of the three levels and proceeds to finer resolutions when the cost function becomes minimal at each level.

Global registration

The first step of the global matching process computes the centroid of the head in each modality. The centroid is basically the three-dimensional center of an object $(\bar{x}, \bar{y}, \bar{z})$ and is computed as follows:

$$\bar{x} = \tfrac{1}{N}\sum_{i=1}^{N} x_i, \quad \bar{y} = \tfrac{1}{N}\sum_{i=1}^{N} y_i, \quad \text{and} \quad \bar{z} = \tfrac{1}{N}\sum_{i=1}^{N} z_i, \tag{B.2}$$

where N refers to the number of voxels inside the head. Note that this definition of the centroid is not a weighted measurement. It gives equal weighting to all voxels, regardless of their relative importance. Once the centers of the images have been aligned, a rotational correction is performed using the covariance matrix of the voxel locations,

$$C = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}, \tag{B.3}$$

where each element is defined as the sample variance or covariance,

$$\sigma_{xy} = \tfrac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x})(y_i - \hat{y}), \quad etc. \tag{B.4}$$

Because the matrix C is symmetric $(\sigma_{ij} = \sigma_{ji})$, there exists a matrix Q that transforms C into a diagonal matrix consisting solely of its eigenvalues along the main diagonal [35]. The matrix can be computed as

$$= \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} = Q^{-1}CQ = Q^T CQ. \tag{B.5}$$

The matrix can be viewed as the principal axes of the data. Consequently, the inverse of Q will rotate the image so that its major axes are aligned with the x, y, and z image planes. Because Q is orthonormal, the rotation matrix R can be computed as

$$R = Q^T. \tag{B.6}$$

Unfortunately, the rotation matrix R is not unique (if λ is an eigenvalue, $-\lambda$ is also an eigenvalue). The authors chose the eigenvalues in order to minimize the magnitude of the rotational corrections. That is, they preferred to rotate by 90 degrees rather than by -270 degrees. Also, to avoid reflections, they preferred positive eigenvalues instead of negative ones.

Once the two images have been centered and rotated into alignment, the last step in the global registration process is to compute the scaling coefficients so that the two images (denoted by image a and image b) have the same size. The scaling matrix K can be determined by taking the ratios of the corresponding eigenvalues,

$$K = \begin{bmatrix} \frac{\lambda_{1a}}{\lambda_{1b}} & 0 & 0 \\ 0 & \frac{\lambda_{2a}}{\lambda_{2b}} & 0 \\ 0 & 0 & \frac{\lambda_{2a}}{\lambda_{2b}} \end{bmatrix}, \tag{B.7}$$

where $(\lambda_{1a}, \lambda_{2a}, \lambda_{3a})$ corresponds to the three eigenvalues of image a (and similarly for image b).

Local registration

The local registration process is performed by applying localized forces subject to the constraints of the Navier-Stokes equation [22] in each of the x, y, and z planes,

$$\mu \nabla^2 u_x + (\mu + \rho)\frac{\partial \theta}{\partial x} + F_x = 0, \tag{B.8}$$

where

$$\theta = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}. \tag{B.9}$$

Here, μ and ρ are elasticity coefficients, u is the three-dimensional displacement vector, and F is the localized force. The authors concede that the choice of the constants μ and ρ is arbitrary because the primary purpose of equation B.9 is to ensure that the elastic deformations are smooth and correspond to a physical deformation process.

In order to evaluate the goodness of fit after all the deformations have been applied, the authors define a similarity metric S between a region f in one image and a corresponding region g in the other image:

$$S(f,g) = \frac{\sum_{i=0}^{\infty} F_i G_i}{\sqrt{\sum_{i=0}^{\infty} F_i^2}\sqrt{\sum_{i=0}^{\infty} G_i^2}}, \tag{B.10}$$

where

$$F_i = \int_D f(v)h_i(v)dv, \quad \text{and} \quad G_i = \int_D g(v)h_i(v)dv. \tag{B.11}$$

The functions h_i are chosen to form an orthonormal basis for the domain D under consideration. For instance, h could be the Fourier sin and cos kernels. The authors chose to use the set of Hermite functions instead, although no theoretical justification was given for this choice. Furthermore, the authors do not describe the cost function for the deformation forces. Presumably, the deformation cost would be based on the sum of the magnitude of the forces applied throughout the region being deformed.

Using the similarity and cost functions, the algorithm then extracts various features from the two image sets and applies the forces iteratively to the brain atlas until the total cost function falls below a threshold minimum value. The authors do not describe how the features are extracted from the images nor do they mention the errors involved in the extraction process.

B.4 Results

The authors provide very few quantitative results of the performance of the matching process. Instead, they include several dozen images showing the intermediate and final stages of deformation for various organs in the brain. The outlines of these organs (as defined by the atlas) are overlayed on the deformed CT images. For subcortical regions such as the caudate nucleus, putamen, thalamus, brain stem, and cerebellum, Bajcsy and Kovacic reported registration positioning errors between an idealized brain atlas and CT images to be within 2 to 4 mm. They also reported that 82% of the image voxels could be matched by using global registration alone. Further evaluation of the accuracy of their system performed by Dann et al. corroborates the initial accuracy reported by the authors [23].

B.5 Parametric Surface-Matching

Szeliski and Lavallee have developed a non-rigid-body (elastic) technique for registration of three-dimensional facial surfaces acquired from laser range scanners [76]. They use a three-dimensional warping function to provide a smooth deformation for matching the two data sets. The surfaces to be matched are first modeled by three-dimensional multiresolution spline. The warping function $T(q_i)$, where q_i is the set of all points on the surface model spline S, is defined by the following tensor product spline:

$$T(q_i) = \sum_{j,k,l} d_{kjl} B_j(x_i) B_k(y_i) B_l(z_i), \tag{B.12}$$

where (x_i, y_i, z_i) are the coordinates of q_i and d_{kjl} are the spline coefficients for the independent B-spline basis functions $B_j(x_i)$, $B_k(y_i)$, and $B_l(z_i)$. The registration technique tries to minimize the cost function over the spline coefficient parameter space p,

$$C(p) = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} [d(r_i, S)]^2, \tag{B.13}$$

which is just the least-squared minimum distance between all the points on one image (r_i) with the surface model of the other image (S). The minimization is performed using a standard Levenberg-Marquardt algorithm.

The unique features of Szeliski's technique are that it (1) first performs a rigid-body transformation to narrow the search space for the minimization algorithm, (2) uses three-dimensional multiresolution spline functions to model the object surfaces, and (3) uses a three-dimensional spline function to perform localized warping between the two image sets.

The authors do not report any quantitative results on the accuracy of their technique. However, they do provide visual results of two experiments, the first using facial surfaces acquired from laser range scans of two different volunteers, and the second using CT images of a patient's vertebra, which were matched against a plastic phantom vertebra [76]. In both case, they demonstrate visually that their elastic matching technique produces better results than the purely rigid-body transformation technique. Whereas their technique shows promising results, it also suffers from three significant drawbacks: (1) it assumes that the object surfaces are already segmented, (2) it assumes that parametric spline models can be constructed for volumes to be matched, and (3) it assumes that the surfaces to be matched are reasonably similar. Hence, this technique is not appropriate for matching CT and MR images, since none of these assumptions can be taken for granted.

B.6 Rigid-body Registration

Gamboa-Aldeco and Fellingham have developed a retrospective surface-based method for registering CT and MR images [33]. Their method determines the optimal global affine transformation matrix for mapping one image onto the other. They have developed a system for displaying anatomical features that have been outlined on one data set in the context of the other data set. Furthermore, they can also display composite images showing the merged data with the outlines superimposed onto their respective features.

Gamboa-Aldeco et al.'s method uses an automated segmentation algorithm for extracting the two-dimensional contours of features to be matched. The segmentation algorithm uses a threshold to determine which points are inside or outside of a feature. These points are then connected using a boundary connection algorithm. The contours are extracted on all of the image planes and are stacked up to form a three-dimensional surface representation. They report that the most reliable contour (in terms of contrast resolution on both imaging modalities) is the interface between the skull and the brain. Using this interface as their target contour, they apply an indicator function to transform the continuous intensity data to binary membership data. That is, if a pixel is inside the contour, it has the value of 1; if it is outside the contour, it has the value 0. The reason for doing this is that, for the purposes of matching, they are only interested in the geometry and not the density distribution of the organ. Using the indicator data, they compute the zeroth, first-, and second-order moments for each modality. The zeroth moment is a measure of the volume of the data and is used to determine the scale factor for the transformation (since two identical images must have isovolumetric features when properly scaled). The first-order moment is a measure of the center of mass and is used to determine the offset value for the transform. The second-order moments are an indicator of the principal axes of inertia for the three-dimensional volume. By computing the eigenvectors of the second-order moments, they can determine the rotation matrix for aligning the orientation of the two data sets.

The authors do not present any data on the absolute accuracy of their matching technique using actual image data. Instead, they analyze and measure the effects of various parameters such as slice spacing, range of the scans (i.e., total scan height), and the thresholds used to determine the surfaces. They created a CT reference scan of a plastic vertebral segment using the highest scan resolution available (1.5 mm). Using this same model, they performed scans at different orientations in order to generate their test data. After running their registration algorithm on the phantom data, they determined that the mismatch error between predetermined point landmarks was 2.3 mm $\pm$ 0.7 mm. Then they performed the matching operation by varying the slice separation, range, and threshold. The slice-separation parameter was varied by interleaving the slices (for example, by using every other slice, or every third slice). The range was varied by successively eliminating the top and bottom slices for each iteration.

According to their tests, the biggest error contribution was due to the threshold parameter used in the segmentation process. The number of slices and the range of the slices did not affect the accuracy as much since the overall symmetry and volumes of the object were still similar. However, the threshold selection had a significant influence on the size and shape of extracted features. This, in turn, had a significant effect on the zeroth and second moment of the indicator functions.

Gamboa-Aldeco et al. have implemented their registration technique in a commercialized system for merging CT and MR data. Their results suggest that radiologists should not need to acquire large numbers of scans with small slice separations in order to accurately merge the CT and MR images. This allows radiologists to minimize the patient's exposure to harmful ionizing radiation. The authors' results underscore the need for developing better segmentation algorithms for accurately extracting the surfaces of organs or features used in determining the transformation matrix.

B.7 Correlation-based Registration

Van den Elsen has developed a restrospective technique for registering CT and MR images of the head [81] [82]. Her technique performs registration by determining the optimal rigid-body transformation matrix needed to match computed features in both image modalities. The features to be matched are computed using mathematical functions that combine differential geometry operators with Gaussian blurring [79] [80]. Rather than use sparse image features such as points, edges, and other selective landmarks, van den Elsen employs a fully automated hierachical feature-correlation technique to match images in two and three dimensions. The features are generated by a *ridge* operator, which, in two dimensions, is defined as the second derivative of the image values along the direction normal to local peak gradients. In three dimensions, the ridge operator is defined as the minimized second derivative of the image values taken over all directions normal to the local peak gradients (for gradient troughs, the second derivative is maximized) [81].

Once the ridge operators have been applied to the CT and MR images, the resulting values show enough similarity that fully automated correlation methods can be used for registration. The optimal rigid-body transformation T for an image g can be found by maximizing the following correlation function:

$$C_T = \sum_{x=0}^{p-1} \sum_{y=0}^{q-1} \sum_{z=0}^{r-1} (f(x,y,z) \cdot g(T(x,y,z))), \tag{B.14}$$

where $f(x,y,z)$ is the value of the image feature voxel at location (x,y,z). The parameters p, q, and r are the dimensions of image feature space of f. One of the noteworthy properties of this correlation function is that it considers only those features that are present in both images. Features that are present in one image and absent in the other contribute zero terms to the summation. Hence, this function is robust as long as there are a sufficient number of matching features to be correlated. This built-in robustness makes the function well suited for automated registration.

The optimal transformation T is found by exhaustively searching a small region in the parameter space close to the starting conditions. The parameter space of T consists of the three translational and three rotational parameters in R^3 (two translational and two rotational parameters in R^2). To reduce the computation time, a multiresolution search is performed, starting at the coarsest resolution and progressively moving to finer resolutions. Once the optimal T is found at resolution level n, the search is repeated at resolution level $n+1$ using the results from level n as the initial conditions. Furthermore, the size of the search region is progressively narrowed as the level of resolution is increased. Hence, the search region is largest at the coarest resolution and smallest at the finest resolution level. To reduce the computation time even further, the CT and MR images are scaled to have identical pixel/voxel dimensions (since the pixel sizes are known at the time of scanning).

van den Elsen compared the accuracy of this registration technique with that of a skin-surface fiducial-marker registration technique. She acquired high-resolution CT and MR data sets from two different patients. Skin fiducial markers were attached to both patients and used for acquiring images in both modalities. For two-dimensional matching, van den Elsen reported a typical deviation of 0.75 pixels for the translation parameters and 0.75 degrees for the rotation parameters between the fiducial-based registration and the correlation-based registration results. For three-dimensional matching, van den Elsen reported that the two schemes “differed somewhat.” After inspecting the results of the two different techniques, she determined that the correlation-based technique was more accurate than the marker-based technique.

Currently, van den Elsen is extending the correlation-matching technique to perform registration of the spine. Because the spine cannot be treated as a rigid body, this matching technique must be applied locally to individual vertebral bodies. This requires the vertebral bodies to be isolated prior to matching in both the CT and MR images, and then segmented prior to image fusion. A fully automated registration technique for spinal images would require higher-level control mechanisms for initializing and interpreting the correlation and segmentation phases.

B.8 Other Methods

Many other researchers have developed techniques for two- and three-dimensional medical image registration. A complete review of all the techniques is impossible given the limited scope of this dissertation. In this section, I will briefly summarize two of these techniques and highlight their unique characteristics.

Variation of intensities

Hill and others [41] [40] have developed an automated rigid-body correlation-based technique for registering CT and MR cranial images. Instead of iteratively solving for the transformation parameters that maximize a correlation function, Hill's technique precomputes a set of *feature space dispersion* functions, which describe how certain features vary as a function of the degree of misregistration. The underlying assumption is that a properly defined feature-based correlation function will have a strong peak when the registration between two images is perfect. In addition, this function should monotonically decrease as the misregistration is gradually increased. In effect, they solve an inverse correlation problem. Rather than searching for a global correlation extremum, they know beforehand (from training sets) how the correlation function varies with the degree of misregistration. The goal is to use this knowledge to rapidly compute the transformation that corrects for the misregistration.

After experimenting with different feature definitions, they found that the variance of intensity ratio (VIR) exhibits the best dispersion characteristics. The VIR is an extension of a feature space developed by Woods and Cherry [86] [85] in which the ratio of intensity values between corresponding voxels in the two images is computed. If the images to be registered are acquired using the same modality, the ratio of intensity values will be constant throughout the entire volume and the variance should be close to zero. If the two images are acquired using different modalities, then the ratio will be a function of the tissue type with a known *a priori* correlation coefficient. When the two images are grossly misaligned, the total correlation of voxel intensity values will be near zero (and hence, the variance of the intensity ratios will be maximized). Hill has gathered empirical data to show that the cumulative VIR measure monotonically approaches a minimal value as the misregistration approaches zero. Hill has reported results based on CT and MR test images from three patients. The first patient was used as the training example to compute the VIR dispersion function. Using this data, Hill reports that the CT and MR images of the other two patients were aligned to an accuracy of 2 mm or better. This accuracy was determined by measuring the mismatch between intrinsic landmarks in the registered image pairs [42]. Although this registration technique seems to be feasible for cranial images, it would require modifications to work with spinal images because of non-rigid-body assumptions.

Surface-matching

One of the earliest and most successful surface-based CT/MR/PET cranial image registration techniques was developed by Pelizzari and his colleagues [66] [50]. This technique can be described using a "hat to head fitting" analogy: The three-dimensional surface of the first image set (called the *head*) is modeled as a stacked set of closed contours that are extracted by a semiautomated edge-detection algorithm (minor editing is required around the ears, nose, and oropharynx). The authors do no mention how the contour extraction algorithm works, nor do they say how accurate it is. A similar edge-detection algorithm is applied to the second image set, but only a subset of the points along the contours are used. The latter set of points form the *hat* model of the three-dimensional surface. Typically, the head model is derived from the higher-resolution data set (usually CT) and the hat model is derived from the lower-resolution data set (usually MR). The registration procedure then finds the optimal affine transformation that minimizes the sum of the squared distances between the hat points to the head surface. The distance measurements are computed by drawing a line from hat points to the centroid of the head. The distance is defined by the point of intersection of this line with a linearly interpolated surface boundary.

The accuracy of this technique was first verified using a polymer cast phantom containing a human skull and sectioned into transaxial slabs with a 2.5 cm thickness. In addition, 5 mm holes were drilled into the slabs on a 3 cm x 3 cm square grid. For PET imaging, the holes were filled with tubes containing an acidic solution. For MR imaging, the entire phantom was submersed in water (since polymer has little signal in MR). For CT imaging, the holes appear as a void and the polymer appears gray. To test the basic accuracy of their technique, the authors performed a CT-to-CT registration in which the second data set was acquired by scanning the phantom with a 15 degree gantry tilt. The average RMS error (i.e., distances) for the CT-to-CT test set was 0.73 mm. Similarly , the average RMS error was 1.16 mm for CT-to-MR registration, 1.80 mm for CT-to-PET registration, and 1.83 mm for MR-to-PET registration. The gantry tilt for the CT images in the CT-to-PET registration was first set to zero, and then 14 degrees. The authors do not mention the tilt for the CT-to-MR registration tests. The pixel sizes were 1.15 mm, 1.95 mm, and 2.50 mm for the CT, MR, and PET images, respectively. All of the image slices were in axial plane.

Although Pelizzari's technique performs adequately for cranial images, it cannot be applied directly to spinal images for two reasons: (1) the rigid-body assumption does not hold in the spine, and (2) the technique relies on accurate segmentation of the two objects that are to be registered. However, this technique is noteworthy not only because of its intrinsic accuracy, but because of the sophistication of the evaluation methods used to measure the system's accuracy. The authors have report successful results in actual clinical settings involving patients with a variety of neurological disorders. Although these results do not constitute a formal clinical evaluation of this technique, they do demonstrate the feasibility of the surface-based registration strategy.

Appendix C

Data Model Definition Language

Over the past few decades, the number of imaging modalities used in clinical and research applications has increased rapidly. These imaging modalities include color photography, plain-film X rays, computed tomography (CT), and magnetic resonance (MR). Today, images are no longer restricted to the simple two dimensional spatial formats characteristic of the earliest scanners. Instead, they can include three-dimensional reconstructions of CT and MR serial studies, or complex two-, three-, and four-dimensional spectroscopic studies. In this paper, we introduce a unified representation scheme for describing images from a wide variety of domains. We have abstracted the common features of image data and captured these features in a formal semantic framework. This framework simplifies the tasks of storage, communication, and interpretation of image data. We refer to this framework as the flexible image representation. Images can be stored and retrieved by using a simple yet powerful protocol called the flexible data format. Images can be transmitted by serializing the data and the metadata that represent the images. Images can be interpreted by using a dynamically typed data model description language which is syntactically similar to the C language but includes symbolic manipulation capabilities. This representation scheme is not intended to replace the joint American College of Radiology/National Electrical Manufacturers (ACR/NEMA) standard for digital image exchange. Instead, we hope that this new representation scheme will coexist and extend the capabilities of the ACR/NEMA standard.

C.1 The Challenge

The problem of how to represent digital images has been a challenge for researchers ever since the first telemetry data were received from satellites in the early 1960s. Today, imaging technologies are used in diverse disciplines such as medicine, meteorology, geophysics, astronomy, and astrophysics. The need to share and communicate images will increase as imaging technology become more widely accepted. This need will place greater demands on the capabilities of software applications that store, retrieve and process images. Standardized image representations will be required so that diverse applications can create and share data in a networked environment. In the area of clinical imaging, standards created by the joint American College of Radiology/National Equipment Manufacturers Association (ACR/NEMA)[2] have made it easier to exchange data among various application platforms. The future holds even more interesting challenges in the area of image standardization. For instance, technologies such as high-definition television and high speed fiber-optic networks will become so commonplace that image processing in the future will be as useful and easy to use as word processoring is today.

Despite the efforts of software designers to standardize the representation of images, there has still been a proliferation of imaging standards. The reasons for this are twofold: (1) Most standards committees have focused their efforts on limited applications areas. As a result, most standards are very difficult to adapt to more general applications. (2) Software designers cannot easily and elegantly extend the expressive powers of most existing standards. As a result, many designers have resorted to creating their own standards in order to overcome the limitations in existing standards.

In this section, we will describe the flexible image representation (FIR) scheme that we have developed in order to overcome the limitations of present day standards. We will first describe the properties of existing standards that software engineers find difficult to conform to. Next, we will describe the ey ideas behind the FIR that enabled us to describe images in a more abstract language. We will then introduce a model description language (MDL) and a flexible data format (FDF) that we have developed in order to turn our abstract image representations into concrete objects that can be readily stored, retrieved, transmitted, and intrepreted. Finally, we will describe the performance of the FIR-MDL scheme used in two different MR image processing applications.

C.2 The MDL Approach

Although the ACR/NEMA standard has been successful in the domain of clinical imaging, it also contains many features that makes it unsuitable for general use. First, it was primarily designed to handle the imaging needs of the medical community. Secondly, ...

What makes MR imaging unique is the vast qualitative differences in the various types of image sequences. For instance, an image can consist of any one of the following entities:

- One-dimensional non-localized spectral densities
- One-dimensional single-voxel spectral densities
- Two-dimensional image planes representing proton densities
- Three-dimensional volume of proton densities
- Two-dimensional image plane consisting of one-dimensional spectral densities
- Three-dimensional volume consisting of one dimensional spectral densities

C.3 Model Description Language

One of our goals in developing the FIR scheme was to provide a means for capturing not only the image data, but also a means for capturing the qualitative and quantitative descriptions associated with the data. These descriptions typically consist of the following entities and parameters:

1. Type of data
2. Rank of the data (one-dimensional, two-dimensional, etc.)
3. Dimensionality of the data (e.g., 256 512 16 elements)
4. Data word type and size (e.g., 64-bit complex or 12-bit short)
5. Abscissa and ordinate types (e.g., parts-per-million, density)
6. Real-world and logical coordinates of the data
7. Geometric orientation
8. Regions of interest
9. Physical parameters (e.g., nuclei, nuclear frequencies)
10. Color maps and pseudo-color information
11. Correction tables and enhancement profiles
12. Overlay information
13. Patient information
14. Free-text descriptions and annotations
15. Graphical annotations and highlighting
16. Other information related to the context or use of the image data

The last parameter presented a big challenge to us while we were developing a standardized image representation. As a result, another goal we had in creating FIR was to allow application writers to incorporate unanticipated parameters without sacrificing the simplicity and efficiency of static representations. In other words, it should be easy for end users to extract only the information they need without incurring a high overhead cost for dealing with a flexible representation. To achieve these goals, we designed the MDL to be flexible enough to handle both the abstract image parameters as well as the domainand application-specific parameters. The specialized parameters can be integrated into the data description in precisely the same manner as can the abstract parameters. The semantics of the MDL allow for a simple, yet flexible and extensible, representation for image data. Hence, the MDL scheme allows application writers to deal with the content of the data independent of the form of the data. The standardization problem then becomes one of agreeing on a vocabulary to describe image parameters rather than agreeing on what the data should look like. Furthermore, the symbolic nature of the MDL lets application writers manipulate data using a declarative programming style as opposed to a procedural programming style.

The MDL is similar in behavior to dynamically typed languages such as Lisp[3] but has a syntax similar to the C programming language[4]. In fact, we designed the syntax of the MDL to be a proper subset of the C language. All parameters are specified by C-style variable declarations. However, unlike the C language, the types of the parameters are determined by the application at runtime - there is no need to compile the MDL file. This allows the programmer to create complicated data objects without having to first specify, and hence restricting, the types of those objects. By using dynamic objects, the programmer can also access data elements symbolically[5]. As long as the vocabulary of the image parameters is known to the programmer, the MDL representations can avoid backward compatibility problems that are characteristic of static representations. As a result, programmers can share data among diverse applications without being concerned about the form of the data.

There are four data types supported by the FIR-MDL at this time:

1. Float
2. String
3. Array
4. Structure

The following are simple floating-point definitions:

```
float pi = 3.14159;
float aa = 1;
```

The following are definitions of string variables:

```
char* storage = "float";
char fid[] = "T1";
```

The following are definitions of arrays:

```
float b[] = {2.718, 1.572, 3.14159};
char ordinate[] = {"intensity", "intensity"};
float roi[] = {NaN, NaN, NaN, NaN};
```

The following are structure definitions:

```
struct {
  float x;
  float y;
  } point1 = { 1.2, 3.4 };

struct {
  float x;
  float y;
  float y;
  char* text;
  } label = { 0, 0, 0, "origin"};
```

A useful syntactic feature of the MDL is that arrays are allowed to be nested. This feature enables users to constructs arbitrary list sequences similar to those found in Lisp:

```
float d[] = {
  1.1, 2.2,
  {3.3, 4.4, 5.5},
  3, 4,
  {{0, 1, 2, 3},
    {4, 5, 6}},9.9
  };
```

Structures can also be nested:

```
struct {
  float e;
  char* f;

  struct {
    float c;
    char* g;
  } h;

} i = { 2.718, "force", {2.99, "null"}};
```

Since the MDL is really a dynamically typed language, type definitions are essentially ignored. If the programmer leaves out any type specification, a C compiler will, by default, interpret such variables as type int. Although the MDL will accept meaningless type declarations, programmers should, for the sake of clarity, use correct type declarations. By using type declarations (when meaningful), programmers can provide heuristics hints to other application that will later use the data. In light of the dynamic typing feature, the following are also syntactically (but not necessarily semantically) valid declarations:

```
j = 3.33;           /* okay, int = float */
k = 36;             /* okay */
i = {1.1, 2.3};     /* int = array, okay for MDL, but not C */
l = "a string?";    /* int = string, okay for MDL, but not C */

m[] = 0;            /* int array = int, okay for MDL, but not C */

char *s = 3;        /* string = int, okay for MDL, but not C */

struct {
  float a;
  } b = {"string"};  /* float = string, okay for MDL, but not C */
```

As a result of the dynamic typing of the MDL, arrays and structures can have mixed data types:

```
d = {
  1, 2,
  {"hydrogen", "nitrogen", "carbon"},
  3,  4,
  { {"nuclear", "frequency"}, "door",0.0},
  "please",
  9.9};

struct {
  float n[];
  int o[];

  struct {
    int p[];
    }q[];
```

```
} r[] = { 1.0, {0, 1}, {{"min"}, {"max"}} };
```

A formal definition of the MDL syntax is shown in Figure C.1. At this time, only data declarations are specified. We are expanding the MDL syntax to include procedure definitions similar to those in the C language.
The following are reserved variable names (keywords) in the MDL:

```
float, int, char, struct, NaN
```

C.4 The Flexible Data Format

Images represented by the FIR-MDL can be turned into file system objects by using the FDF convention. This convention specifies how a programmer or application must construct an FIR-MDL image file. By adhering to the FDF convention, programmers will be able to create, access, and share images, in their entirety, regardless of the origin or destination of these images.

We have divided the FDF file into three parts. The first part is the magic string, which distinguishes the application class of the file. The following is an example of a magic string:

```
#!/usr/local/Apps/data
```

The magic string must appear as the first line of the FDF file. Although there is no limit on the length of the lines in the file, the magic string must only appear on the first line and must include the # symbol as its first character. The magic string can be followed by optional magic tokens to provide further information about the image file:

```
#!/usr/local/Apps/data image 34.5 54.0 3.14159
```

Neither the magic string nor the magic tokens have any particular syntactic or semantic significance, other than to specify the image file properties to an external program that might be reading it. Readers who are familiar with the UNIX operating system[6] will recognize the our magic string was chosen so that FDF files are self-executing. That is, UNIX-based end users can launch their imaging applications by double clicking on the FDF file icons or by typing the name of the file into the UNIX shell. We think this was a reasonable design feature because there was no other major operating system that interpreted the first character string of an executable file as a system command.

The second part of the FDF consists of the MDL code specification corresponding to the image metadata. The specification of the metadata has been covered in Section 3. The magic string, the magic tokens, and

```
mdlspec -> statements;
statements -> statement; < statement; ...>

statement -> assigment
statement -> declaration
statement -> expression

assignment -> VARIABLE = expression

declaration -> <typename> VARIABLE <[]> <= expression>

typename -> float <*>
typename -> int <*>
typename -> char <*>
typename -> VARIABLE <*>

expression -> <+| -> NUMBER
expression -> VARIABLE
expression -> array
expression -> assignment
expression -> struct { statements } VARIABLE <[]> <=> array
expression -> STRING
expression -> ( expression )

array -> { expr <, expr ...> }

string -> "list of characters enclosed within double quotes"

comment -> C-style comment
```

Figure C.1: The formal grammar specification for the MDL. Optional items are enclosed by < ... > wherease alternations are delimited by '| '. For example, <+| -> NUMBER means either +NUMBER, -NUMBER, or just NUMBER.

the MDL code, taken as a whole, constitute a single string. This string is terminated by a null character (ASCII character zero).

The final part of the FDF file is the actual image data in binary form. In order to visually separate the metadata from the binary image data, we recommended that the entire MDL code section consist purely of printable text characters. Furthermore, we require that the binary image data start on an eight-byte block boundary. This will ensure that any FDF file that is read into an application's memory will have the correct word alignment. Applications that require a block boundard other than eight bytes should explicitly define a parameter called word_boundary in the MDL code:

```
int word_boundary = 128; /* Alignment width of data in bits */
```

If an application encounters a word boundary that is not compatible with its machine architecture, then that application is required to realign the image data before using the data. We included the word boundary requirement for the sake of efficiency: Applications that adhere to the default word boundary will not have to spend time realigning the image data after reading the FDF file into core memory. Because of the word boundary requirement, there may be a gap in the FDF file between the MDL code and the binary data. This gap can be closed by padding the end of the MDL code that precedes the null byte, with spaces or newline characters. Even though the end-user applications will always ignore the pad characters, programmers should not leave these characters in an undefined state. Undefined pad characters will make the MDL section of the FDF file visually-unfriendly to other application writers or end users.

We will now present an example of a file that illustrates all of the key elements of the FDF convention. The following FDF file represents a twodimensional chemical-shift image:

```
#!/usr/local/fdf/csi

/* Abstract parameters sections */

int rank = 3;
char* storage = "float";
int bits = 32;
char* type = "complex";

int matrix[] = {1024, 16, 16};
char* abscissa[] = {"ppm1", "spatial", "spatial"};
char* ordinate = {"intensity", "intensity", "intensity"};
```

```
float span[] = {2000.0, 256.8, 256.8};
float origin[] = {0.0, 0.0, 0.0};
float location[] = {0.0, 0.0, 0.0};
float orientation[] =
        { 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0 };
float roi[] = {8.0, 8.0};

/* Application specific section */

char* nucleus[] = {"H1"};
float nucfreq = 85.5;
char* acquisition_type = "half echo";
float shift_factor[] = {0.5, 0.5, 0.5};
int rotation_number = 0;
char* data_swap = "noswap";
float acquisition_delay = 15.0;

float const_phase[] = {45, 45, 44, 46, 48 };
float linear_phase[] = {730, 730, 735, 735 };

/* Clinical section */

struct { name; age; id; } patient ={
  "John Q. Subject",
  45,3235131 };

char* study = "head";

/* Miscellany */

int checksum = 07739202;


<padding>

<null byte>
```

<image data>

The actual parameters in this example are not relevant to this discussion; they are merely used to illustrate the organization of the FDF file. Note that the first line of this file contains the magic string and magic tokens. The MDL code constitutes the lines following the magic string and extending to the null byte. The remainder of the file contains the image data.

C.5 Evaluation

We have used the FIR-MDL and FDF techniques in two different commercial applications targeted for the research community. Both applications are currently being developed by Spectroscopy Imaging Systems Corporation (SISCO). Using these applications, users can view, manipulate, and analyze two-, three and four-dimensional MR data sets acquired from SISCO scanners. The first application is a general purpose image browser (viewer) and image quantitation system used for two and three-dimensional MR data sets. This application also has the ability to reconstruct and display animations of time series acquisitions. The second application is an image processing system for viewing and analyzing chemical shift images. Initially, both of these projects were hampered by the lack of a comprehensive image representation strategy. The work we have described in this paper was a direct result of our need for a better method for representing the complex characteristics of spectroscopic MR images.

In addition, we have used the FDF scheme to describe non-image-based data, such as histograms, tables and two-dimensional cine lists. We have written a set of library routines in C++, to provide an object-oriented toolkit for creating, accessing, modifying, and interpreting MDL objects. We tested the performance of the MDL toolkit by measuring the the amount of time needed to retrieve images and parameters from FDF files. We compared these measurements with the amount of time needed to retrieve the same images and parameters that were stored in the original static image representation (our applications are required to be backward-compatible with our previous image format). The results of these measurements are shown in Table C.1.

The results of these timing tests indicate that MDL scheme imposes a higher overhead cost than the static scheme (568 milliseconds versus 301). However, once the application has incurred the overhead cost, the time to read subsequent images, under the MDL scheme, is reduced by one half (146 milliseconds versus 308). For our applications, which deal with multiple, simultaneous images, the subsequent image access time is more important than the initial access time.

C.6 Summary

We have introduced a new technique for representing images from a wide variety of domains. We developed this technique by first abstracting common characteristics shared by all images. Next, we developed the MDL mechanism to capture both the abstract, application-independent properties, as well as the

Measuremed Times (ms)	Static	MDL
Initial Parameter Read time	147	120
Initial Data Allocation time	154	448
Total Initial time	301	568
Subsequent Parameter Read time	156	20
Subsequent Data Allocation time	152	126
Total Subsequent time	308	146

Table C.1: Comparison of parameter access times and image data allocation times for the static and MDL image representation schemes (all measurements are in milliseconds). The initial parameter read time refers to the time needed to read the first image in a sequence of images. Likewise, the initial data allocation time refers to the time needed to allocate the memory for the first image in a sequence of images. All images consisted of 256 by 256 floating-point values.

application-dependent properties of images. Using the MDL syntax, the application writer can use a powerful, yet simple, mechanism for capturing and expressing the complex parameter sets used in various imaging modalities. The MDL mechanism draws much of its power from its symbolic representation of data objects. Internally, these objects are organized and manipulated hierachically, much as are s-expressions in Lisp[8]. S-expressions are well known for their ability to represent arbitrarily complicated code and data structures[9].

We do not claim to have a solution for representing images from every possible domain. We do think, however, that the work we have done in the field of MR image representation can offer some suggestions for the development of broad based standards for the future. We do not intend the FIR-MDL representation scheme to replace the currently accepted ACR/NEMA standard. The two representations deal with different layers of information encoding (although there is a great degree of overlap between the two approaches). Instead, FIR-MDL scheme will hopefully co-exist, and eventually be integrated into, the ACR/NEMA standard as a future extension.

Appendix D

DDL Object Library

The DDL parser has been designed as a stand-alone library package to read and manipulate the headers for binary data files. In order to use the package, you must include the header file <ddllib.h> in your code and and link the code with the library libddl.a. The file "ddllib.h" describes all of the methods that are available to end user applications to read and manipulate data file headers (please refer to the file "ddl-parser.h" for a description of the class hierarchies created by the library).

The DDL functionality is implemented by the DDLSymbolTable class. This class is responsible for opening, reading and parsing a data file written in the standard SISCO format. Upon successful parsing of the data header file, this object will contain a dictionary of symbol names and their associated values defined by the DDL header file.

The first form of the constructor accepts a string containing the unopened data file to be parsed and a string which contains the correct matching magic "number", This is the magic string designating the SISCO data format. To ignore the magic string and read files with any magic string, the user should supply an empty string "". The second form of the object constructor accepts a reference to an ifstream object which describes the file to be read. Note that in the first form of the constructor, an ifstream object will be created and stored as a public member of the DDLSymbolTable object.

If the file is successfully opened and parsed, this procedure returns a symbol table containing all the information that has been parsed from the header file. The symbol table also contains an element which is an ifstream object that manipulates the same data file. This ifstream object can then be used to read any additional data following the header (it is identical to the ifstream infile parameter in the second calling instance shown above).

This section describes the public interface to the DDLSymbolTable object:

```
enum MapType { DATA_IS_MALLOCED, DATA_IS_MMAPPED, DATA_IS_SHMEM } ;
```

MapType is an enumerated type that describes how the data will be accessed. DATA_IS_MALLOCED means that a block of memory identical in size to the binary data segment from the input file will be allocated from heap area of the calling process. The binary data segment will then be copied from the file into the memory block and used for subsequent computations. Any modifications to the data segment in the memory block will not modify the external data of the input file unless the SaveData method is invoked (refer to the section on saving data). DATA_IS_MMAPPED means that the binary data from the file will be memory mapped into the address space of the calling process. Modifications to the memory segment will not modify the external data unless the mapping protection flag has been set to MAP_SHARED (refer to the section of mapping data). DATA_IS_SHMEM means that the binary data segment will point to a block of shared memory. The shared memory block must be preallocated before this map type can be used.

```
class DDLSymbolTable : public DDLNode {
public:
```

DDLSymbolTable Constructors

```
DDLSymbolTable(char* filename, MapType = DATA_IS_MMAPPED);
```

This constructor template will read the symbol table from the header portion of the filename. The default MapType will be used unless overridden by the calling procedure. If this constructor cannot successfully open the file specified by "filename", then internal error flag will be set.

```
DDLSymbolTable();
```

This constructor template creates a symbol table with no file association. It can be used to create a parameter set with or without a binary data segment. The symbol table can then be treated as if it were read in from an external file. None of the constructors will read the data segments from an external file unless the AttachData() method is invoked. This constructor will always succeed and no error flags will be set.

Data manipulation methods

```
int AttachData();
```

AttachData() will attach the data segment associated with the input file. The method for attaching the data will depend on the value of MapType. The method will return TRUE if the data segment is successfully attached and FALSE otherwise.

```
int DetachData(); // removes the data (if possible)
```

DetachData() will remove data that was previously associated with the symbol table via the AttachData() call. This method will return TRUE if the data was successfully detached and FALSE otherwise.

```
void SetData(char* data, int data_length); // Redefines the data
```

SetData() will redefine the binary data segment to the new value specified by the argument called data. The data length parameter must be supplied since it determines how many bytes of data will be written by the output methods. The original data will be either freed or unmapped depending on the allocation strategy used.

```
char *GetData();
```

GetData() returns a pointer to the binary data segment. This method will return NULL if AttachData() or SetData() has not been called previously.

```
int DataLength();
```

DataLength() returns length of the binary data segment. This method will return 0 if AttachData() or SetData() has not been called previously.

Symbol table manipulation methods

```
int IsDefined(char *name);
int NotDefined(char *name);
```

IsDefined() will return TRUE iff the parameter specified in the character string "name" is defined. NotDefined() will return TRUE iff the parameter specified in "name" is undefined.

Scalar value query methods:

```
int GetValue(char *name, int& value);
int GetValue(char *name, double& value);
int GetValue(char *name, char*& value);
```

These methods will query the integer, double precision floating point, and string values respectively, of the parameter specified in the character string "name". If the parameter is defined in the symbol table, then the value of parameter value will be passed to the reference argument called "value" and the method will return TRUE. If the parameter is undefined, the variable called "value" will not be changed and the method will return FALSE. If the type of the return argument does not match the type of the parameter in the internal symbol table, then the return value will be indeterminate. These methods will return FALSE if the parameter specified by "name" does not exist, and TRUE otherwise.

Vector query methods

```
int GetValue(char *name, int value[], int index);
int GetValue(char *name, double value[], int index);
int GetValue(char *name, char* value[], int index);
```

These methods will query the elements of one dimensional array parameters (i.e., vectors) in the symbol table. The value of the symbol table vector parameter specified by "name" at the index location specified by "index" will be assigned to the vector argument called "value". That is, value[index] <- parameter "name"[index]. If the type of the value array does not match the type of the symbol table parameter, then value[index] will be indeterminate. These methods will return FALSE if the parameter specified by "name" does not exist, and TRUE otherwise.

Array query methods

```
int GetValue(char *name, double& value, int v, int u);
int GetValue(char *name, double& value, int w, int v, int u);
```

These methods will query the elements of two- and three-dimensional, double precision array parameters specified by the argument called "name". The first method will access the value of parameter "name"[v][u] and the second method will access the value of parameter "name"[w][v][u]. These methods are similar to the vector query methods described in the previous section with the exception that they will only operate on double precision arrays.

Vector to scalar query methods

```
int GetValue(char *name, int& value, int index);
int GetValue(char *name, double& value, int index);
int GetValue(char *name, char*& value, int index);
```

These methods will query the elements of one dimensional array parameters (i.e., vectors) in the symbol table. The value of the symbol table vector parameter specified by "name" at the index location specified by "index" will be assigned to the scalar argument called "value". That is, value <- parameter "name"[index].

This differs from the vector query functions described earlier since they expect a scalar reference argument rather than a vector reference argument. This can be useful in cases where it is inconvenient to define an array just to query one or two elements. If the type of the value array does not match the type of the symbol table parameter, then value[index] will be indeterminate. These methods will return FALSE if the parameter specified by "name" does not exist, and TRUE otherwise.

Scalar setting methods:

```
int SetValue(char *name, double value);
int SetValue(char *name, char* value);
```

These methods will set the parameter specified by "name" to the value specified by the argument "value". If the parameter does not exist, this function will create an entry in the symbol table before updating the parameter value. Hence this function can be used for both creating and updating scalar parameter values. Furthermore, updating the value of parameter will redefine the type of the parameter if the new type differs from the old type.

Vector setting methods:

```
int SetValue(char *name, double value, int index);
int SetValue(char *name, char* value, int index);
```

These methods will set the vector parameter element specified by "name" at the location specified by "index" to the value specified by the argument "value". If the index is out of range for the parameter, or if the parameter does not exist in the symbol table, this method will do nothing and return a value of FALSE. If the parameter exists and the index is valid, updating the parameter element will redefine the type of the element if the new type differs from the old type.

Array manipulation methods

```
int *GetDimensions(char *name, int& dimensionality);
```

GetDimensions() creates an integer array which contains the number of of dimensions along each axis of the named array. The number of axes is returned in the variable "dimensionality".

```
double *GetDoubleVector(char *name, int& rows);
```

This instance of the method GetDoubleVector() will return a pointer to a vector of double precision numbers representing the elements of the parameter specified by "name". This method is useful for functions that need to access the symbol table parameter as a C-style vector (one-dimensional array). Since this method will create a new instance of the vector at each invocation, the calling procedure must deallocate the vector when it is no longer needed. The length of the vector will be passed to the reference argument "rows". If the array does not exist in the symbol table, this method will return NULL and no value will be passed to argument "rows".

double **GetDouble2DArray(char *name, int& rows, int& cols);

This instance of GetDoubleVector() will return a pointer to an array of pointers to vectors of double precision numbers representing the elements of the parameter specified by "name". This method is useful for functions that need to access the symbol table parameter as a two-dimensional C-style array. Since this method will create a new instance of the C-style array at each invocation, the calling procedure must deallocate the array when it is no longer needed. The number of rows and columns of the parameter array will be passed to the reference arguments "rows" and "cols". If the array does not exist in the symbol table, this method will return NULL and no value will be passed to argument "rows" and "cols".

int LengthOf(char* name);

LengthOf() will return the number elements in the largest dimension of the parameter array specified by "name". For instance, LengthOf() for 10 x 20 x 30 element array will return 10.

Pointer array methods

Methods to create pointers to pointers to the data heap:

void *Create2DArray(int cols, int rows, int element_size)

Create2DArray() will return a pointer to a double-precision, two-dimensional, floating point C-style array (i.e., an array of pointers to vectors of type double). This two-dimensional array can be used to access and modify the values of the binary data segment. This is analogous to the method GetDouble2DArray() which is used to access values of symbol table array parameters. Since the object does not know the dimensions of the binary data segment array, the arguments "cols" and "rows" must be specified by the calling procedure. Similarly, the word size of the binary data must be specified in the argument "element_size". If the product of cols, rows, and element_size exceeds the size of the binary data segment, then the return array will contain invalid addresses at locations that are out of range. Any attempt to access or modify these locations will most likely result in a system error. This method will call AttachData() if the data segment has not been attached already.

```
void *Create3DArray(int cols, int rows, int planes, int element_size)
void *Create4DArray(int cols, int rows, int planes, int superplanes,
        int element_size)
```

Create3DArray() will return a pointer to a double-precision, three-dimensional, floating point C-style array (i.e., an array of pointers to arrays of pointers to vectors of type double). Create4DArray() will return a pointer to a double-precision, four-dimensional, floating-point C-style array. These three- and four-dimensional array can be used to access and modify the values of the binary data segment. Since the object does not know the dimensions of the binary data segment array, the arguments "cols", "rows", "planes", and "superplanes" (for Create4DArray) must be specified by the calling procedure. Similarly, the word size of the binary data must be specified in the argument "element_size". If the product of cols, rows, planes, superplanes (for Create4DArray), and element_size exceeds the size of the binary data segment, then the return array will contain invalid addresses at locations that are out of range. Any attempt to access or modify these locations will most likely result in a system error. These methods will call AttachData() if the data segment has not been attached already.

```
void *CreateNDArray(int dimensions[], int element_size) {
return ::CreateNDArray(data, dimensions, element_size);
}
```

CreateNDArray() is a generalized array method that will return a pointer to a double-precision, n-dimensional array. The dimensions of the array are specified in the integer array argument called "dimensions". The "dimensions" argument is a zero terminated list of dimensions that the returned array should have (starting with the largest dimension). For example, to create a 10 x 20 x 30 x 40 x 50 dimension array, the value of "dimensions" would be set to { 10, 20, 30, 40, 50, 0 } . The n-dimensional array that is returned can be used to access and modify the values of the binary data segment. Since the object does not know the dimensions of the binary data segment array, the "dimensions" argument must be specified by the calling procedure. Similarly, the word size of the binary data must be specified in the argument "element_size". If the product of the array dimensions and element_size exceeds the size of the binary data segment, then the return array will contain invalid addresses at locations that are out of range. Any attempt to access or modify these locations will most likely result in a system error. This method will call AttachData() if the data segment has not been attached already.

Multi-dimensional arrays methods

```
DDLNode *CreateArray(char *name);
DDLNode *CreateArray(char *name, DDLNode *ddlarray);
DDLNode *AppendElement(char *name, double value);
```

```
DDLNode *AppendElement(char *name, char *value);
DDLNode *AppendElement(char *name, ArrayData *value);
```

Helper methods for creating 2D arrays from either a single heap or from array of pointers to pointers:

```
DDLNode *CreateDouble2DArray(char *name, double *array, int rows, int cols);
DDLNode *CreateDouble2DArray(char *name, double **array, int rows, int cols);
```

File info methods

```
int HeaderLength();
int FileLength();
```

Input/ Output methods

```
void PrintSymbols(ostream& os = cout);
void PrintSymbolsAll(ostream& os = cout);
void SaveSymbolsAndData(ostream& os, char *magic, int do_checksum = 0);
void SaveSymbolsAndData(char* filename, char *magic, int do_checksum = 0);
void SaveData(char* filename);
void SaveData(ostream& os = cout);
void SaveSymbols(ostream& os, char *magic_string, int do_checksum = 0);
void SaveSymbols(char* filename, char *magic_string, int do_checksum = 0);
```

Internal node manipulation methods

```
DDLNode *Lookup(char*);
DDLNode *Install(char*, int);
DDLNode *Install(char*, int, double);
DDLNode *Install(char*, int, char*);
void Remove(char* name);
```

Memory mapping methods

```
void SetMMapFlags(int flags) { mmap_flags = flags; }
void SetMMapProt(int prot) { mmap_prot = prot; }
void SetMMapAddr(int addr) { mmap_addr = addr; }
void SetMMapOffset(int offset) { mmap_off = offset; }
```

```
void SetMapType(MapType m) { map_type = m; }
int GetMMapFlags() { return mmap_flags; }
int GetMMapProt() { return mmap_prot; }
int GetMMapAddr() { return mmap_addr; }
int GetMMapOffset() { return mmap_off; }
MapType GetMapType() { return map_type; }
```

Error state methods

```
int operator!(); // returns true if error flag set
```

Miscellaneous

```
char *infile;
ifstream *ddlin;
ofstream debug;
protected:
int syserror; // 0 => no errors
char *data;
int header_length; // number of bytes in the header
int data_length; // total number of bytes in the infile
int file_length; // total number of bytes in the data file
MapType map_type;
int mmap_prot;
int mmap_flags;
int mmap_addr;
int mmap_off;
} ;
```

Appendix E

Datamatrix Object Library

The datamatrix class library defines an n-dimensional matrix object that can be used for image processing applications that need to perform the following operations:

Threshold
Convolution
Derivatives
Edge detection
Linear algebra: addition, subtraction, eigenvalues, least squares, etc
Statistics
Region of Interest (ROI) manipulation
Input/Output

A datamatrix is an object that encapsulates the form and functionality of an n-dimension array of elements. This object contains methods to perform computations that are useful for image processing applications. The array elements have a single homogeneous type which is specified at the time the object is created. However, the array element type, size, and data can be redefined at any time. *MatrixDataType* is an enumerated type definition contains the set of all valid types recognized by the datamatrix object:

```
typedef enum {
BYTE_MATRIX_DATA,
UNSIGNED_BYTE_MATRIX_DATA,
SHORT_MATRIX_DATA,
```

UNSIGNED_SHORT_MATRIX_DATA,
INT_MATRIX_DATA,
UNSIGNED_INT_MATRIX_DATA,
FLOAT_MATRIX_DATA,
DOUBLE_MATRIX_DATA,
LONGDOUBLE_MATRIX_DATA,
UNKNOWN_MATRIX_DATA,
NODATA_MATRIX_DATA
} MatrixDataType;

Before data can be associated with a datamatrix object, the user must determine whether the data is to be allocated and owned by the object, or the data is to be shared with an external source. The enumerated type *MatrixDataSrcType* indicates whether the matrix object has a private copy of its data or whether it is sharing data. An object with PRIVATE data will free its data upon object deletion whereas an object with SHARED data will not attempt to free its data:

typedef enum {
MATRIX_DATA_SRC_PRIVATE,
MATRIX_DATA_SRC_SHARED
} MatrixDataSrcType;

The choice of MatrixDataSrcType will affect the behavior of the object destructor. If MATRIX_DATA_SRC_PRIVATE is chosen, the object will delete its internal data heap when the destructor is invoked. However, if MATRIX_DATA_SRC_SHARED is chosen, the object will not delete the data heap since it is not the owner of the data.

Generally speaking, arrays contained data acquired over a uniform coordinate axes. For example, a three-dimensional image will contains several planes of two-dimensional information where the planes are sampled at a fixed interval. However, there are cases where the data planes may not be contiguous (for example, a random archive of two-dimensional images). In order to handle both of these scenarios, an enumerated type called *MatrixConnectivityType* indicates whether the information in adjacent image planes are sequentially and isotropically connected:

typedef enum {
MATRIX_CONNECTIVITY_FALSE,
MATRIX_CONNECTIVITY_TRUE
} MatrixConnectivityType;

The Public Interface

This section describes the publicly accessible members and methods of the datamatrix class:

```
class DataMatrix {
public:
```

DataMatrix Constructors

```
DataMatrix();
```

This is the default constructor and is mainly provided as a convenience for subclasses of this object whose form may not be known at creating time.

```
DataMatrix(char *data, int dimensions[], MatrixDataType datatype,
        MatrixDataSrcType srctype);
```

This form requires caller to pre-allocate data and supply all the information needed to characterize the data matrix.

```
DataMatrix(int dimensions[], MatrixDataType datatype);
```

This form will allocate its own PRIVATE data storage based on the dimensions and datatype.

```
DataMatrix::DataMatrix(int planes, int rows, int columns,
                MatrixDataType datatype, const char *data_filename);
```

This for will allocate its own PRIVATE data storage based on the dimensions and datatype. It will read the data_file and load it into the data field.

```
DataMatrix(char *data, int columns, MatrixDataType datatype);
DataMatrix(char *data, int rows, int columns, MatrixDataType datatype);
DataMatrix(char *data, int planes, int rows, int columns,
        MatrixDataType datatype);
```

These three forms are convenience constructors for creating a one-, two-, or three-dimensional data matrix with external, shared data.

```
DataMatrix(int columns, MatrixDataType datatype);
DataMatrix(int rows, int columns, MatrixDataType datatype);
DataMatrix(int planes, int rows, int columns, MatrixDataType datatype);
```

Array pointers

The following are convenience functions for C-style multi-dimensional array manipulation:

```
void *CreateNDArray(void *data, int dimensions[], int esize);
void *getNDArray(int N);
void *DataMatrix::get2DArray() { return getNDArray(2); }
void *DataMatrix::get3DArray() { return getNDArray(3); }
void *DataMatrix::get4DArray() { return getNDArray(4); }
void *DataMatrix::get5DArray() { return getNDArray(5); }
```

Conversion methods

The following functions will convert the type of the data elements. If the size of the new type is different from the original type, the old data will be freed and a new data block will be allocated for the converted data:

```
DataMatrix& ConvertDataToDouble();
DataMatrix& ConvertDataToFloat();
DataMatrix& ConvertDataToInt();
DataMatrix& ConvertDataToUnsignedInt();
DataMatrix& ConvertDataToShort();
DataMatrix& ConvertDataToUnsignedShort();
DataMatrix& ConvertDataToByte();
DataMatrix& ConvertDataToUnsignedByte();
DataMatrix& ConvertDataToType(MatrixDataType datatype);
DataMatrix& ConvertDataToType(DataMatrix& source);
DataMatrix& CopyDataTo(DataMatrix &dest);
```

Operators

```
DataMatrix& Threshold(int threshold);
DataMatrix& Convolve(DataMatrix& kernel, DataMatrix& data);
DataMatrix& Derivative(...);
DataMatrix& EdgeDetect(...);
DataMatrix& Moment(int moment);
DataMatrix& SVD();
DataMatrix& operator=(DataMatrix& rvalue);
DataMatrix& operator+(DataMatrix& increment);
DataMatrix& operator-(DataMatrix& increment);
```

```
DataMatrix& operator*(DataMatrix& increment);
DataMatrix& operator*=(float scale);
```

Accesser functions

```
void *getData() { return data; }
DataMatrix& setData(char *data, MatrixDataSrcType srctype);
MatrixDataType getMatrixDataType() { return datatype; }
MatrixConnectivityType getConnectivityType() { return conntype; }
MatrixDataSrcType getMatrixSrcType() { return srctype; }
int *getDimensions() { return dimensions; }
int getMatrixDataLength() { return dataheapsize; }
int getMatrixDataWordSize() { return datawordsize; }
int getDimensionality() { return dimensionality; }
int getErrorState() { return error_state; }
int resetErrorState() { error_state = 0; }
```

Modifier functions

```
DataMatrix& setConnectivityType(MatrixConnectivityType conntype);
```

Input/ Output functions

```
DataMatrix& attachFile(const char *filename);
DataMatrix& detachFile();
DataMatrix& rewindFile();
DataMatrix& loadRawFile(int length, int byte_offset);
DataMatrix& loadRawFile(int length, int byte_offset, MatrixDataType in_type);
DataMatrix& saveRawFile(const char *filename, int mode);
void writePGMFile(const char *filename);
friend ostream& operator<<(ostream&, DataMatrix&);
```

Destructor

```
 DataMatrix();
```

Internals

```
private:
MatrixDataType datatype;
```

datatype stores the current data type information.

```
int datawordsize;
```

datawordsize stores the size, in number of bytes, of the array element. It is determined by the value of the datatype member variable.

```
int dataheapsize;
```

dataheapsize stores the size, in number of bytes, of the data array heap.

```
MatrixConnectivityType conntype;
```

conntype stores the current data connectivity type information.

```
MatrixDataSrcType srctype;
```

srctype stores the current data source type information.

```
ifstream *infile;
```

Infile is used by methods that perform input operations.

```
ostream *outfile;
```

Outfile is used by methods that perform output operations.

```
int *dimensions;
```

The "dimensions" vector stores the dimensionality of the data matrix. E.g., a 512x256x10 volume implies dimensions = { 10, 256, 512, 0 }.

```
int dimensionality;
```

The dimensionality member stores the number of dimensions in the data matrix.

```
void *data;
```

Data is a pointer to the data matrix heap that stores the actual data. The matrix is stored in a single monolithic array, i.e., a sequence of bytes.

```
int error_state;
```

error_state is set whenever an object method encounters an error that prevents the method from completing its operation.

```
int setSizes(int dimensions[], MatrixDataType datatype);
```

setSizes() will inform the object the dimensions of the data array have changed. It is merely and advisory method and will not change the data array or any C-style pointer arrays built from the array. However, any subsequent calls to methods such as getNDArray() will contain proper indices for the redimensioned data array.

```
protected:
void initDataMatrix(int dims[], MatrixDataType datatype,
            const char *filename = NULL);
```

initDataMatrix() is a common initialization method that overloaded constructors can use to set up the default values of the object member variables. It can also be invoked by object methods and by derived classes to perform object initialization at arbitrary times.

```
virtual void DataMatrix::display();
```

display() is a virtual function that derived classes can override to provide device specify methods for displaying the data array.

```
void DataMatrix::writePNMData(const char *datafile,
                    void *datap, int width, int height);
```

writePNMData() will save the data array as a PNM image file. The calling procedure must specify the width and height for the output image and must also provide the pointer to the starting point of the data segment to be written (datap).

Appendix F

Data Set Parameters

The scanning parameters used to acquire the twelve CT cervical data sets are shown in Tables F through F. The descriptions of these parameters are given briefly in Table F below:

Parameter	Description
data element size	number of bytes in each data element
data element type	"unsigned", "signed", "char", "short", "int", "long", "float", "double"
slice width	number of pixels in x-dimension
slice height	number of pixels in y-dimension
number of slices	total number of axial slices in data set
x pixel resolution	in mm
y pixel resolution	in mm
z pixel resolution	in mm
x fov	width of image in mm
y fov	height of image in mm
z fov	depth of image sequence in mm
slice thickness	acquisition thickness of each slice in mm
slice spacing	distance between slice centers
gantry tilt	in degrees
bone threshold	in CT units
raw data filename	name of file containing raw data segment
checksum program	command used to compute data checksums
image plane orientation	"orthogonal axial" or "axial"

Table F.1: **Description of data set parameters.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	256
slice height	256
number of slices	99
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.5
x fov	160
y fov	160
z fov	115
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p1.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.2: **Data set P001.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	256
slice height	256
number of slices	230
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.5
x fov	160
y fov	160
z fov	115
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p2.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.3: **Data set P002.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	256
slice height	256
number of slices	62
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.5
x fov	160
y fov	160
z fov	115
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p3.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.4: **Data set P003.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	299
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.5
x fov	160
y fov	160
z fov	115
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p4.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.5: **Data set P004.**

Parameter	Value
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	181
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.5
x fov	160
y fov	160
z fov	65.5
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p5.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.6: **Data set P005.**

Parameter	Value
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	58
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.62212
x fov	160
y fov	160
z fov	135
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p6.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.7: **Data set P006.**

Parameter	Value
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	39
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	3.00
x fov	160
y fov	160
z fov	114
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p7.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.8: **Data set P007.**

Parameter	Value
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	179
x pixel resolution	0.3125
y pixel resolution	0.3125
z pixel resolution	0.5
x fov	160
y fov	160
z fov	89.5
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p8.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.9: **Data set P008.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	52
x pixel resolution	0.6739
y pixel resolution	0.6739
z pixel resolution	1.5
x fov	345
y fov	345
z fov	89.0
slice thickness	1.50
slice spacing	1.745
gantry tilt	0.00
bone threshold	1230
raw data filename	"p9.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.10: **Data set P009.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	256
slice height	256
number of slices	119
x pixel resolution	0.703125
y pixel resolution	0.703125
z pixel resolution	1.5
x fov	180
y fov	180
z fov	59.0
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p10.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.11: **Data set P010.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	30
x pixel resolution	0.292969
y pixel resolution	0.292969
z pixel resolution	1.5
x fov	150
y fov	150
z fov	135.0
slice thickness	5.00
slice spacing	5.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p11.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.12: **Data set P011.**

Parameter	**Value**
data element size	2
data element type	"unsigned short"
slice width	512
slice height	512
number of slices	275
x pixel resolution	0.292969
y pixel resolution	0.292969
z pixel resolution	1.5
x fov	150
y fov	150
z fov	128.1
slice thickness	1.00
slice spacing	1.00
gantry tilt	0.00
bone threshold	1230
raw data filename	"p12.data"
checksum program	"brik -Gbq"
image plane orientation	"orthogonal axial"

Table F.13: **Data set P012.**

Appendix G

Cervical Scan Protocols

CT Protocol for Cervical Spine

Spinal Fusion Group
Lucas MRS Center, Dept of Radiology
Stanford University

1/5/93

Dr. John Adler (b3490)
Dr. John Drace (b3336)

Spiral CT (Use **GE Scanners** since we need 4 back−to−back spirals), Cover C1−C7
Tilt neck back to get dental work out of C1−C7 as much as possible.
Lateral and AP Scout. Reposition neck as above if necessary and scout again.

Assume superior C1 is at location Ix and inferior C7 is at location Iy.
For 4 sequential spirals:

- 2mm overlap between spirals
- 120kV
- max mA optimized for all spirals, but not below 150 mA
- slice thickness (collimator width): 1mm
- table speed: 1 mm/s
- pitch: 1.0mm

	Start location	End location	No. images	Group delay (secs)
Spiral 1:	Ix	Ix + 34	35	0
Spiral 2:	Ix + 32	Ix + 66	35	60
Spiral 3:	Ix + 64	Ix + 98	35	60
Spiral 4:	Ix + 96	Iy	*	60**

* No of images for spiral 4 should be <= 40
**If mA falls below 150 during optimization, can make this longer

Prospective recon: Center **160mm FOV** using LAT and AP scout, standard algorithm.
Retrospective recon: Same as above, but increment every **0.5mm**.
Please save the raw data to DAT tape.
Retrospective recons to IC for transfer over network to Lucas Center
Film Prospective Recons ONLY, and only the ODD numbered images.

Scans must be performed by the technologists who are experienced with this protocol.

Check List

1. Keep the neck extended so that tooth–fillings do not interfere with the scans.

2. ***Warning***: avoid spiral start/end errors:
   If you scan from S to I:
   Eg 1:
   1st spiral start loc: S169
   1st spiral end loc: S135 (x)
   2nd spiral start loc: S137 (not S133) (x+2)
   Eg 2:
   1st spiral start loc: I100
   1st spiral end loc: I134 (x)
   2nd spiral start loc: I132 (not I136) (x–2)

   If you scan from I to S, above scenerio is ***reversed***.

2. 60 sec. cooling time between spirals.

3. 0.5 mm retrospective recon: center coordinates of all images should be the same (specify recon center for each spiral to be the same).

4. 150mA min.

MR Protocol for Cervical Spine

Spinal Fusion Group
Lucas MRS Center, Dept of Radiology
Stanford University

1/5/93

Dr. John Adler (b3490)
Dr. John Drace (b3336)

Important: ***Posterior Neck Coil must be used*** *!! (Medical Advances Coil may also be good)*

Ser	PSD	Plane	TR/TE	BW/ET	Flip	Gate	PC	RC	VB	NP	Sat	Options	FOV	Thk/Sp	Matrix	NEX	Notes
1	GRE 2D	C	33/min TE	+ or − 16	20°	–	yes	–	–	–	–	Seq FC	24	5/0 P10−A40	256x128	1	Localizer
2	3DFGRE type−in PSD	S TE	Min Full	+ or −16	15°	–	–	–	–	–	–	Fast GRx EDR	20	1.2/60*	256x256	2 or 3	CV: sl_order = 1 Fast GRASS Fallback selected 8.34 mins at 2 NEX 12.5 mins at 3 NEX
3	Product [SPGR, FAST] (fgre3d)	S TE	Min Full	+ or −16	15°	–	–	–	–	–	–	Fast GRx	20	1.2/60*	256x224	2 or 3	Fast GRASS Fallback selected 8.34 mins at 2 NEX 12.5 mins at 3 NEX

***Increase slice thickness if necessary to cover the cervical spine from left to right.**

To get the B0−maps:

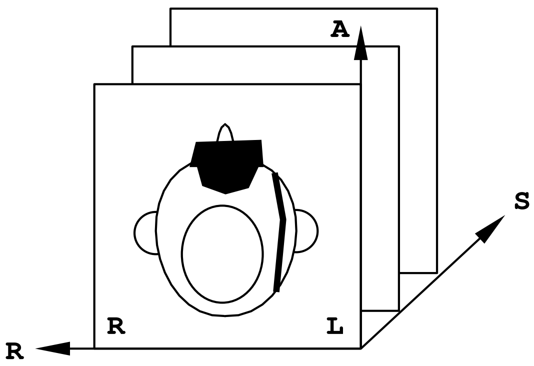
Scan 1:
1. rhformat = 5, rhrcctrl = 19
2. act_te = TE + 4878 (take down the TE value (*x*))
3. act_tr = new value (take this down too (*y*))

(typically TE = 6984, TE + 4878 = 11862, TR = 23600)

Scan 2:
1. rhformat = 5, rhrcctrl = 19
2. act_te = *x*
3. act_tr = *y*
4. No prescan

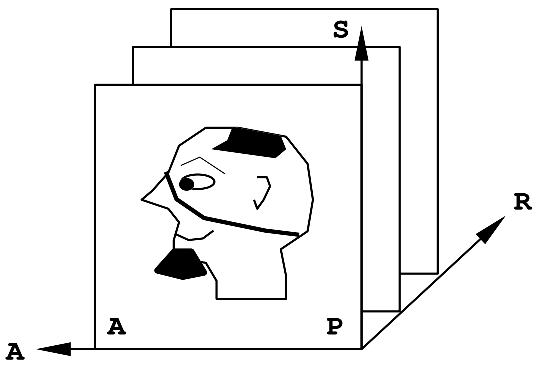
RAS Coordinate Reference Standard

Axials



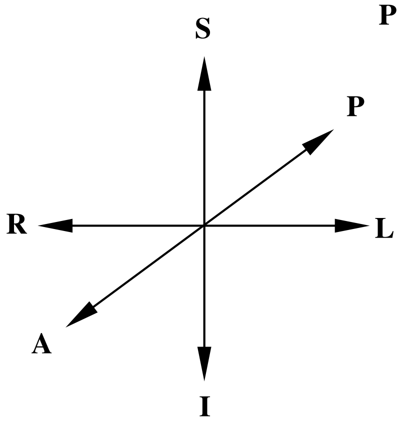
1. Indices increase from I to S.
2. Images go into the screen as S increases
3. View from I to S..

Sagittals

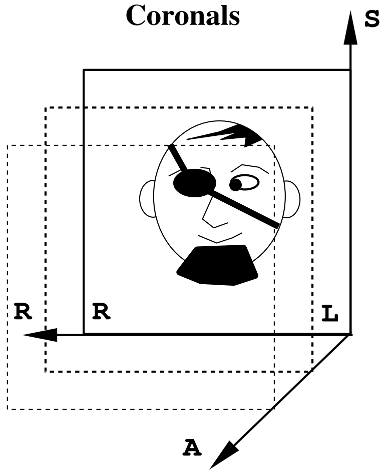


1. Indices increase from L to R.
2. Images go into the screen as R increases.
3. View from L to R.

S = Superior
I = Inferior
R = Right
L = Left
A = Anterior
P = Posterior



Coronals



1. Indices increase from P to A.
2. Images come out of the screen as A increases.
3. View from A to P.

Appendix H

Glossary of Terms

Contrast Resolution also **Dynamic Range :** The minimal possible difference in intensity level variation. This is primarily influenced by the number of significant bits in the sampled image data and for practical purposes, is also influenced by the signal-to-noise ratio of the image.

Contrast-to-Noise Ratio (CNR): The minimal possible difference of signal-to-noise ratios within the image.

Feature: A characteristic (either qualitative or quantitative) that is associated with an anatomical element or structure. For example, the surface of a disk can have the features of density, curvature, volume, angle, etc., associated with it.

Feature Space: A set of features used to distinguish or identify image elements. For example, the probability of a region belonging to a disk can be computed using the feature space consisting of image intensity, curvature, volume, and angle.

Landmark: An anatomical structure used as a point of reference for orientation or identification of images or other structure within an image. A landmark will usually have a unique location (origin or center) associated with it. Complex landmarks can have other related parameters such as length, area, volume, curvature, axes of orientation, etc. Examples of landmarks include outlines of fiducial systems, outlines of the skull or other bony processes.

Registration: The process of aligning two or more images in order to make them appear to be precisely overlapping. Registration can be performed in either two or three dimensions. Registration usually involves translation, scaling and rotation of the images with respect to one another.

Segmentation: The process of breaking up an image into its constituent pieces or distinct regions corresponding to different tissue types or anatomical regions.

Signal-to-Noise Ratio (SNR): The ratio of the standard deviation of the signal to the standard deviation of the noise. In practice, the standard deviation of the signal cannot be determined precisely but it can be estimated by using statistical methods.

Spatial Resolution: The distance between adjacent sample points in the image space. This is primarily limited by the number of samples taken in each dimension and the field of view of the image. Indicates the ability to resolve spatial features in the image.

Structure: An aggregated collection of elements such as points, lines, surfaces or volumes with some defined relationship among all the elements. The collection of elements and their relationships in turn defines an anatomical entity.

Bibliography

[1] John R. Adler. Professor of Neurosurgery, Stanford University, Department of Neurologic Surgery, personal interview, 1993.

[2] Russ B. Altman. A probabilistic algorithm for calculating structure: Borrowing from simulated annealing. Technical report, Stanford University, Stanford California, 1990.

[3] Russ B. Altman and James F. Brinkley. Probabilistic constraint satisfaction with structural models. In *Proceedings of the Symposium on Computer Applications in Medical Care*, 1993.

[4] Ruzena Bajcsy and S. Kovacic. Multiresolution elastic matching. *Computer Vision, Graphics, and Image Processing*, 1(46):1–21, apr 1989.

[5] S. Barclay. Hugin runtime (software package). *Journal of Behavioral Decision Making*, 6(1), March 1993.

[6] Marie-Odile Berger. Towards autonomy in active contour models. *Proceedings of the 10th International Conference On Pattern Recognition*, 1:847–51, June 1990.

[7] Thomas O. Binford. Image understanding: intelligent systems. In *Image Understanding Workshop Proceedings*, volume 1, pages 18–31, Los Altos, California, February 1987. Morgan Kaufmann.

[8] Thomas O. Binford, Tod S. Levitt, and Wally B. Mann. Bayesian inference in model-based machine vision. In Tod S. Levitt, L.N. Kanal, and J.F Lemmer, editors, *Uncertainty in Artificial In Intelligence.* North Holland, 1989.

[9] Jules Bloomenthal. An implicit surface polygonizer. In Paul S. Heckbert, editor, *Graphics Gems IV*, Graphics Gems Series, pages 324–349. Academic Press, Cambridge, Massachusetts, 1994.

[10] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, Massachusetts, 1967.

[11] Ronald Bracewell. *The Fourier Transform and its Applications, second edition*. McGraw-Hill, New York, 1986.

[12] Ronald Bracewell. *The Hartley Transform*. Clarendon Press, Oxford, 1986.

[13] C. Breau et al. Reconstruction of a human ligamentous lumbar spine using ct images - a three-dimensional finite element mesh generation. *Annals of Biomedical Engineering*, 19:291–302, 1991.

[14] Oran Brigham. *The Fast Fourier Transform and its Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[15] James F. Brinkley. Hierarchical geometric constraint networks as a representation for spatial structural knowledge. In *16th Annual Symposium on Computer Applications in Medical Care*, pages 140–144, 1992.

[16] James F. Brinkley. A flexible, generic model for anatomic shape: application to interactive two-dimensional medical image segmentation and matching. *Computers and Biomedical Research*, 26(2), April 1993.

[17] C. Broit. *Optimal Registration of Deformed Images*. PhD thesis, University of Pennsylvania, Philadelphia, 1981.

[18] C. K. Chow and T. Kaneko. Automatic boundary detection of the left ventricle from cineangiograms. *Computers and Biomedical Research*, 5(4):388–410, 1972.

[19] Harvey E. Cline et al. 3d surface rendered mr images of the brain and its vasculature. *Journal of Computer Assisted Tomography*, 15(2):344–351, March/April 1991.

[20] James M. Coggins. A statistical approach to multiscale, medial vision. Research report, Department of Computer Science, University of North Carolina at Chapel Hill, 1992.

[21] James M. Coggins. Statistical investigations of multiscale image structure. In *Proceedings of the Second International Conference on Visualization in Biomedical Computing*, Chapel Hill, North Carolina, October 1992.

[22] R. Courant and D. Hilbert. *Methods of Mathematical Physics*. Interscience, New York, 1966.

[23] R. Dann, J. Hoford, S. Kovacic, M. Reivich, and R. Bajcsy. Preliminary clinical evaluation of multi-resolution elastic matching software. Technical report, University of Pennsylvania, 1988. Technical Report MS-CIS-88-35.

[24] Viewpoint Datalabs. Dataset catalog, 1995.

[25] Chris A. Davatzikos and Jerry L. Prince. Convergence analysis of the active contour model with applications to medical images. *Visual Communications and Image Processing*, 1818:1244–1255, 1992.

[26] M. Dietrich, K. Kedzior, and T. Zagrajek. A biomechanical model of the human spinal system. *Journal of Engineering in Medicine*, 205:19–26, 1991.

[27] Richard C. Dorf. *Modern Control Systems*. Addison Wesley, Reading Massachusetts, 1980.

[28] John Drace. Professor of Radiology, Stanford University, Department of Radiology, personal interview, 1993.

[29] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.

[30] Dieter Enzmann. Professor of Radiology, Stanford University, Department of Radiology, personal interview, 1993.

[31] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics*. Addison Wesley, Reading Massachusetts, 1990.

[32] J.F. Frymoyer and S. Gordon. *American Academy of Orthopedic Surgery*, chapter Prospectives in Low Back Pain. Park Ridge, 1988.

[33] Arturo Gambo-Aldeco and Linda Fellingham. Correlation of 3d surfaces from multiple modalities in medical imaging. In *Medical Imaging, Processing, and Display and Picture Archiving and Communication Systems for Medical Applications*, Newport Beach, California, February 1986. SPIE. Medicine XIV/PACS IV.

[34] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming. Technical Report SOL 86-2, Stanford University Systems Optimization Lab, Stanford, California, January 1986.

[35] Gene H Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, 1989.

[36] Serge Gracovetsky. *The spinal engine*. Springer-Verlag, New York, 1988.

[37] K.B. Heithoff and R.J. Herzog. *The Adult Spine*, chapter Computed tomography (CT) and enhanced CT of the spine. Raven Press, 1991. J.W. Frymoye, editor.

[38] Paul Hemler, Thilaka Sumanaweera, Ramani Pichumani, Petra A. van den Elsen, Sandy Napel, John Drace, and John Adler. A system for multimodality image fusion of the spine. In *Applications of Computer Vision in Medical Image Processing*, Spring Symposium Series, Stanford, California, March 1994. AAAI.

[39] Paul F. Hemler, Sandy Napel, Thilaka S. Sumanaweera, Ramani Pichumani, Petra van den Elsen, David Martin, John Drace, Inder Perkash, and John R. Adler. Registration error quantification of a

surface-based multimodality image fusion system. Technical Report KSL-94-66, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, 1994.

[40] Derek L.G. Hill and David J. Hawkes. Medical image registration using voxel similarity measures. In *Applications of Computer Vision in Medical Image Processing*, pages 34–37, Stanford, California, March 1994. AAAI Spring Symposium Series.

[41] Derek L.G. Hill, David J. Hawkes, Neil A. Harrison, and Cliff F. Ruff. A strategy for automated multimodality image registration incorporating anatomical knowledge and imager characteristics. In H.H. Barrett and A.F. Gmitro, editors, *Proceedings of the 13th International Conference on Information Processing in Medical Images*, pages 182–196, 1993.

[42] D.L.G. Hill, D.J. Hawkes, Z. Hussain, S.E.M. Green, C.R. Ruff, and G.P. Robinson. Accurate combination of ct and mr data of the head: Validation and applications in surgical and therapy plannin. In C. Roux, G.T. Herman, and R. Collerec, editors, *3D Advanced Image Processing in Medicine*, pages 79–84. IEEE Engineering in Medicine and Biology Society, 1992.

[43] Ronald A. Howard and James E. Matheson. *Principles and Applications of Decision Analysis, Volume I*. Strategic Decisions Group, Menlo Park, California, 1990.

[44] Ronald A. Howard and James E. Matheson. *Principles and Applications of Decision Analysis, Volume II*. Strategic Decisions Group, Menlo Park, California, 1990.

[45] Thomas S. Curry III, James E. Dowdey, and Robert C. Murry Jr. *Christensen's Physics of Diagnostic Radiology*. Lea and Febiger, Malvern, Pennsylvania, 1990.

[46] R. E. Kalman and R. Bucy. New results in linear filtering and prediction. *Journal of Basic Engineering*, 83(D):95–108, 1961.

[47] Michael Kass et al. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1987.

[48] H.J. Kim and H.S. Yang. A neural network capable of learning and inference for visual pattern recognition. *Pattern Recognition*, 27(10):1291–1302, October 1994.

[49] Andrew Laine. *Wavelet theory and application*. Kluwer Academic, Boston, Massachusetts, 1993.

[50] David N. Levin, Charles A. Pelizzari, George T.Y. Chen, Chin-Tu Chen, and Malcolm D. Cooper. Retrospective geometric correlation of mr, ct, and pet images. *Radiology*, 169:817–823, 1988.

[51] Tod S. Levitt, Marcus W. Hedgcock, John W. Dye, Scott E. Johnston, Vera M. Shadle, and Dmitry Vosky. Bayesian inference for model-based segmentation of computed radiographs of the hand. *Artificial Intelligence in Medicine*, 5(4):365–387, August 1993.

[52] F. Leymarie and M.D. Levine. New method for shape description based on active contour model. In *Proceedings of the SPIE - The International Society for Optical Engineering*, volume 1192 of *Intelligent Robots and Computer Vision VIII: Algorithms and Techniques*. SPIE, November 1990.

[53] Frederic Leymarie and Martin D. Levine. Tracking deformable objects in the plane using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 617–634, 1993.

[54] Wallace B. Mann. *Three Dimensional Object Interpretation of Monocular Gray-Scale Images*. PhD thesis, Stanford University, Stanford, California, 1994.

[55] Dave Martin. Professor of Neurosurgery, Stanford University, Department of Neurologic Surgery, personal interview, 1993.

[56] J. McCarthy. Recursive functions of symbolic expressions. *Communications of the ACM*, 3(4):184–195, 1960.

[57] Sylvie Menet, Philippe Sait-Marc, and Gerard Medioni. Active contour models: Overview, implementation and applications. In *IEEE International Conference On Systems, Man And Cybernetics Conference Proceedings*, pages 194–9, Los Angeles, California, November 1990. IEEE.

[58] Yael Moses and Shimon Ullman. Limitations of non model-based recognition. In G. Sandini, editor, *Computer Vision - ECCV '92. Second European Conference on Computer Vision Proceedings*, pages 820–8, Berlin, Germany, May 1992. Springer-Verlag.

[59] G. Noone, J. Mazumdar, and D. Ghista. Continuous model of the human scolotic spine. *Journal of Biomedical Engineering*, 13:473–480, November 1991.

[60] R. Ohlander, K. Price, and D. R. Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing*, 8(3), December 1979.

[61] Lucila Ohno-Machado. *Hierarchical Neural Networks: Applications in Medical Care*. PhD thesis, Stanford University, Stanford, California, to appear in 1995.

[62] Jean Oliver and Alison Middleditch. *Functional Anatomy of the Spine*. Butterworth Heinemann, Oxford, 1991.

[63] Joseph O'Rourke. *Computation Geometry in C*. Cambridge University Press, Cambridge, 1994.

[64] Kirill K. Pankratov. SaGA: Spatial and geometric analysis toolbox (unpublished). Department of Earth, Atmospheric and Planetary Sciences, Massachusetts Institute of Technology, 1995.

[65] Judea Pearl. Fusion, propagation, and structuring in bayesian networks. *Artificial Intelligence*, 1986.

[66] Charles A. Pelizzari, George T. Y. Chen, Danny R. Spelbring, Ralph R. Weichselbaum, and Chin-Tu Chen. Accurate three-dimensional registration of ct, pet, and/or mr images of the brain. *Journal of Computer Assisted Tomography*, 13(1):20–26, January/February 1989.

[67] Jean Ponce and David J. Kriegman. On recognizing and positioning curved 3d objects from image contours. In *Proceedings of the Image Understanding Workshop*. DARPA, 1989.

[68] J. M. S. Prewitt. Object enhancement and extraction. In B.S. Lipkin and A. Rosenfeld, editors, *Picture Processing and Psychopictorics*. Academic Press, New York, 1970.

[69] L. G. Roberts. Machine perception of three-dimensional solids. In J. P. Tippett, editor, *Optical and Electro-optical Information Processing*. MIT Press, Cambridge, Massachussets, 1965.

[70] Francis W. Sears, Mark W. Zemansky, and Hugh D. Young. *University Physics*. Addison-Wesley, Reading, Massachusetts, 1987.

[71] Ross D. Shachter. Evaluating influence diagrams. *Journal of Operations Research*, 34:871–882, 1986.

[72] Irwin Sobel. Camera models and machine perception. Technical Report AIM-21, Stanford University Artifical Intelligence Lab, Stanford, California, May 1970.

[73] I. S. Sokolnikoff. *Mathematical Theory of Elasticity*. McGraw-Hill, New York, 1956.

[74] H. C. Sox, M. A. Blatt, M. C. Higgins, and K. I. Marton. *Medical Decision Making*. Butterworths, Boston, Massachusetts, 1988.

[75] J.S.B. Stopford. The arteries of the pons and medulla oblongota: part 1. *Journal of Anatomy*, page 131, 1916.

[76] Richard Szeliski and Stephane Lavallee. Matching 3-d anatomical surfaces with non-rigid volumetric deformations. In *Applications of Computer Vision in Medical Image Processing*, Spring Symposium Series, Stanford, California, March 1994. AAAI.

[77] P.G. Tahoces, J. Correa, M. Soutos, L. Gomez, and J.J. Vidal. Computer-assisted diagnosis: the classification of mammographic breast parenchymal patterns. *Physics in Medicine and Biology*, 40(1):103–117, January 1995.

[78] C. Taitz, H. Nathan, and B. Arensburg. Anatomical observations of the foramina transversaria. *Journal of Neurolog,y Neurosurgery and Psychiatry*, page 170, 1978.

[79] B.M. ter Haar, L.M.J. Florack, J.J. Koenderink, and M.A. Viergever. Scale-space: its natural operators and differential invariants. In A.C.F. Colchester and D.J. Hawkes, editors, *Proceedings of the 12th International Conference on Information Processing in Medical Images*, pages 239–253, 1991.

[80] Bart M. ter Haar Romeny et al. Higher order differential structure of images. In H.H. Barrett and A.F. Gmitro, editors, *Proceedings of the 13th International Conference on Information Processing in Medical Images*, pages 77–93, 1993.

[81] Petra A. van den Elsen. *Multimodality Matching of Brain Images*. PhD thesis, Utrecht University, The Netherlands, 1992.

[82] Petra A. van den Elsen. Retrospective fusion of ct and mr brain images using mathematical operators. In *Applications of Computer Vision in Medical Image Processing*, Spring Symposium Series. AAAI, March 1994.

[83] Michael G. Walker. *Probability Estimation for Classification Trees and DNA Sequence Analysis*. PhD thesis, Stanford University, Stanford, California, March 1992.

[84] Merriam Webster. *Webster's New Collegiate Dictionary*. G. & C. Merriam Company, Springfield, Massachusetts., 1993.

[85] R.P. Woods, S.R. Cherry, and J.C. Mazziotta. Mri-pet registration with automated algorithm. *Journal of Computer Assisted Tomography*, 17(4):536–46, 1992.

[86] R.P. Woods, S.R. Cherry, and J.C. Mazziotta. A rapid automated algorithm for accurately aligning and reslicing pet images. *Journal of Computer Assisted Tomography*, 16(4):620–633, 1992.

Colophon

This manuscript was set in 12-point New Century Schoolbook font using the LaTeX document formatting language. Final PostScript output was generated using the **dvips** conversion program. All source material creation (figures, tables, diagrams, and text) as well as source code development, debugging, and testing were performed on a Silicon Graphics Indigo2 workstation. Larger-scale computations were performed on a Silicon Graphics Power Challenger parallel processing computer.