Seeing Structure:
Using Knowledge to Reconstruct and Illustrate Anatomy

Kevin P. Hinshaw

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2000

Program Authorized to Offer Degree: Dept. of Computer Science & Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Kevin P. Hinshaw

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

James F. Brinkley

Reading Committee:

James F. Brinkley

Linda Shapiro

Steven Tanimoto

Date:

Signature____________________________

Date_______________________________

University of Washington

Abstract

Seeing Structure:
Using Knowledge to Reconstruct and Illustrate Anatomy

by Kevin P. Hinshaw

Chair of Supervisory Committee

Professor James F. Brinkley
Dept. of Biological Structure

Current medical imaging technology makes it possible to gather remarkably detailed three-dimensional data about an individual's anatomy. In domains ranging from education to clinical medicine, a common desire is the ability to examine selected structures from such volume datasets. This dissertation describes tools for performing the two key tasks in that process: reconstructing (or *segmenting*) specific structures from volume data and illustrating them in meaningful ways.

On the reconstruction side, this work offers new, in-depth analysis of two previously proposed methods for using shape knowledge to guide image segmentation. The ideas are generalized to create a 3D shape model, which is used as part of a novel algorithm for semi-automatic segmentation of the brain. Unlike other methods, this approach offers intuitive user controls and explicitly addresses the removal of the skull and other surrounding structures. This method is incorporated into a working, interactive system for recording and studying functional data from the human brain. On the illustration side, several real-world situations are used to demonstrate how non-standard rendering methods can enhance the clarity of anatomical illustrations. The lessons learned from these examples lead to requirements for and a prototype of a medical illustration system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AXIAL: parallel to a plane separating the upper and lower halves of the body. An axial slice is sometimes called a horizontal or transverse slice.

COMPUTED TOMOGRAPHY (CT): an imaging method in which cross-sectional images are computed mathematically after measuring the body's absorption of X-rays from different angles.

CORONAL: parallel to a plane separating the front and back halves of the body.

CORTICAL STIMULATION MAPPING (CSM): an invasive method for identifying specific sites in the brain where language function occurs. This method is sometimes used to plan removal of tumors or epileptic foci.

FUNCTIONAL MAGNETIC RESONANCE IMAGING (fMRI): an *MRI* method which records physiological activity, typically by measuring changes in blood oxygenation over time.

GYRUS (*pl.* GYRI): a crest or ridge on the surface of the brain.

HOOK FUNCTION: a programming language mechanism that lets the programmer modify the behavior of a function or object, but without having to change its internal code.

MAGNETIC RESONANCE IMAGING (MRI): an imaging method which uses a strong magnetic field to align the atomic nuclei within a patient's body, then measures their responses to a radio-frequency pulse. The nuclei behave differently depending on their types and densities, and these differences are mapped to grayscale intensities to create cross-sectional images of the body.

POSITRON EMISSION TOMOGRAPHY (PET): an imaging method which records physiological activity by measuring the decay of radio-isotopes injected into the body.

PIXEL: the sampling unit for digital images, typically storing one grayscale or color value which represents the light visible over some small area of a given scene.

RECONSTRUCTION: the process in which a geometric model of an object is derived from images or volumes which show the object.

REGION GROWING: a *segmentation* method which starts with one or more single-pixel regions, then expands them by successively adding adjacent pixels that satisfy some measure of similarity.

RENDERING: the process of creating a picture of a scene, given descriptions of the geometrical and material properties of the objects it contains, the lights, and the camera.

SAGITTAL: parallel to a plane separating the left and right halves of the body.

SEGMENTATION: the process of dividing an image or volume into distinct regions according to some definition of similarity.

SULCUS (*pl.* SULCI): a fissure separating two adjacent *gyri* on the surface of the brain.

THRESHOLDING: a *segmentation* method in which pixels are divided into two groups, depending on whether they fall above or below a given intensity threshold.

TRAINING DATA: instances that are used as representative examples of some larger class of objects.

VOXEL: the 3D equivalent of a *pixel*. Volume data is represented as a three-dimensional array of voxels.

ACKNOWLEDGMENTS

My journey at the University of Washington has flown by, thanks in large part to the terrific people I have met along the way. This dissertation would not exist without their help and support.

First and foremost I must thank my advisor, Jim Brinkley, for his ability to balance the roles of teacher, cheerleader, mentor, and psychiatrist. Cornelius Rosse, Linda Shapiro, and Steve Tanimoto also provided guidance at critical times, and I appreciate their help. On a daily basis, I have learned much from the talented and supportive members of UW's Structural Informatics Group: August Agoncillo, Evan Albright, Bill Barker, Bill Lober, Richard Martin, Onard Mejino, Bharath Modayur, Kate Mulligan, Jeff Prothero, Andrew Poliakov, Darren Stalder, John Sundsten, and Ben Wong. Rex Jakobovits deserves special mention, both for paving the road to graduation by example and for keeping the beeline in my bonnet. The results shown here owe much to the excellent MRI images provided by Ken Maravilla and his colleagues in Radiology, and also to Dave Conley for his 3D models of organs in the thorax. My work was made possible by funding from the National Cancer Institute, the National Library of Medicine, Achievement Rewards for College Scientists, and the National Institute on Deafness and Other Communication Disorders.

I am also grateful to those who made my time *away* from school so enjoyable. I found some of the best housemates ever with Ruth Anderson, Sung-Eun Choi, and Tina Vlasity. I will miss our shared decompression sessions, but look forward to many years of Blue House Reunions. To Tim Salzman and the UW wind ensemble: thank you for keeping me sane by welcoming me into your fold. You brought me dear friends in the forms of Pam Farmer, Conny Chen and Nina Shimabukuro, and my life is forever enriched by the music we shared. My time at UW would not have been nearly as fun without Brad Chamberlain, Jim Fix, TJ Goan, Kurt Partridge, Mike Perkowitz, Geoff Voelker, and Alec Wolman. I owe much

to my alter ego, Steve Hackstadt, for countless long-distance gripe sessions and for always pushing me to do my best (even if he didn't realize he was doing so). Most importantly, I am deeply grateful to my family—Dad, Mom, and Jennifer—for always being there for me and never losing faith in my ability to finish. (Or if they did lose faith, for keeping quiet about it.)

for Leroy, Dolores, Mary, and Paul

Chapter 1

INTRODUCTION

Having just the vision's no solution,
Everything depends on execution:
Putting it together—
That's what counts.

Stephen Sondheim,
Sunday in the Park with George

This dissertation describes the design and development of a working software system which uses prior knowledge to help reconstruct and illustrate anatomy.

1.1 Motivation

Over the past two decades, great strides have been made in the field of medical imaging technology. The advent of methods such as magnetic resonance imaging (MRI) and computed tomography (CT) has provided physicians with powerful, non-invasive ways for studying the internal structure and behavior of the human body.

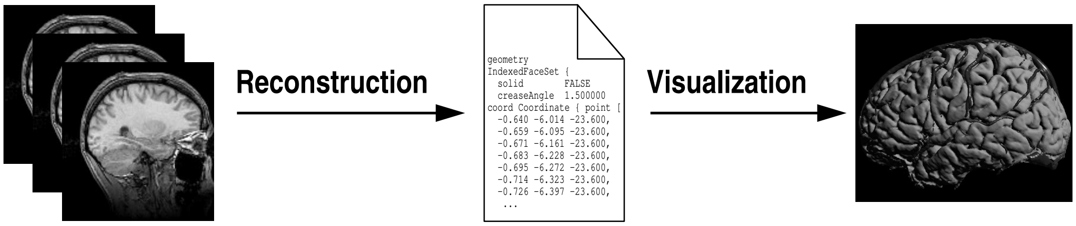
Although they are presented as pictures, medical images differ significantly from traditional photographs. A photograph captures how objects reflect and transmit light. By contrast, an MR image is computed mathematically. By placing the body in a strong magnetic field and measuring how different regions respond to a radio-frequency pulse, an MRI scanner can generate an image which differentiates between tissue types. And unlike X-ray imaging, which produces a projected image of the body, MRI and CT methods create cross-

Figure 1.1: An MR image of the head (sagittal view).

sectional images of the body. (See figure 1.1.) When stacked, these images form a *volume dataset*—a dense, three-dimensional grid of sample points which captures not just the visible outer surface of the body, but its internal structures as well. Current MRI technology offers sub-millimeter accuracy, and it will undoubtedly improve with time.

While volume data can be useful on its own, it often provides a mere starting point for other types of analysis. A common desire in biomedical disciplines is the ability to examine selected structures from such volume datasets. Sometimes the goal is computational: calculating the size of a tumor, or predicting how much damage the liver would sustain from a proposed radiation treatment plan. In other cases, the intent is more qualitative, such as showing a medical student how the major abdominal organs are arranged in a real patient. Either way, the task generally can be divided into two main stages: *reconstruction* and *visualization*. (See figure 1.2.)

The goal of reconstruction is to identify and isolate significant objects in a volume dataset and describe them as 3D geometric models. In other words, the computer must



```
geometry
IndexedFaceSet {
  solid        FALSE
  creaseAngle  1.500000
coord Coordinate { point [
  -0.640 -6.014 -23.600,
  -0.659 -6.095 -23.600,
  -0.671 -6.161 -23.600,
  -0.683 -6.228 -23.600,
  -0.695 -6.272 -23.600,
  -0.714 -6.323 -23.600,
  -0.726 -6.397 -23.600,
   ...
```

Figure 1.2: High-level stages of volume data analysis. From volume data, structures are isolated and converted into 3D models. These models are then manipulated and transformed to create a 2D picture.

"see" the volume data not as a giant array of intensity values, but in terms of individual structures. Despite what simplified anatomy illustrations suggest, organs do not float freely in the human body. The connective tissues that hold them together are often ignored for the purposes of high-level illustration, but they still show up in MRI scans. Somehow the computer must be able to separate the MRI data into distinct regions, identify the relevant ones, and generate a meaningful geometric description for each of them.

Given 3D reconstructions, the goal of visualization is to turn the geometry into pictures. In some cases, the intent may be images that closely resemble photographs of real organs. In others, the intent may be more abstract, calling for illustrations of structural relationships that cannot be expressed well with photorealistic methods. This process of converting 3D models into 2D pictures is also called *rendering*.

In a sense, both stages help a certain audience to *see structure* in the volume data. Reconstruction helps the computer see it, and visualization helps humans see it. Neither task is trivial. Indeed, each of them has spawned its own subfield of computer science research; the first is the domain of computer vision, while the second is the domain of computer graphics.

1.2 Problems and Insights

This dissertation focuses on two particular problems that materialize when trying to examine structures from volume data.

Problem 1: Complete organ boundaries are difficult to find.

Unfortunately, full organ boundaries in CT and MR images are often difficult to detect with standard vision techniques, because adjacent soft tissues with similar densities get mapped to nearly equal intensity values. Boundaries may be clearly defined in some regions, but there are often large sections where they are not.

Insight: Human observers can cope with such challenges. They can “fill in the blanks” mentally in low-contrast regions by using visual cues from nearby edges and knowledge of an object’s shape. If an image segmentation system were provided with the right kind of information, it might be able to do the same. Shape knowledge could be used to guide the search for edges which are clear and infer boundaries in regions where the image contrast is too low.

Problem 2: Standard rendering tools are inadequate for anatomy illustration.

Rendering is frequently viewed as a solved problem, thanks to the standardized graphics pipeline for transforming 3D models into 2D images. This standard approach to rendering developed from efforts to imitate optics and generate photorealistic images. However, medical illustration offers a prime example of a domain in which photorealistic rendering methods fall short. The human body is a complex, densely layered machine. Clearly depicting the structural relationships among its parts can be a daunting task, one which requires considerable anatomical understanding on the part of the artist.

Insight: Anatomy textbooks rely on drawings much more heavily than on photographs. This suggests that photorealistic rendering techniques are not the right approach. While the task may be difficult, medical illustrators have developed an array of techniques for depicting the complex spatial organization of the human body. By adopting such techniques, the quality of anatomical renderings could be improved.

1.3 A Possible Solution

This dissertation presents tools for handling the complete transformation from volume data to illustrations of specific anatomical structures. To deal with the problems outlined above, it uses shape models of organs to guide the segmentation process, and it explores non-photorealistic rendering techniques to enhance the quality of the illustrations.

The guiding principle of this work is that the tools should be useful and practical, rather than mere research prototypes. Therefore they must be interactive, with the user's role kept a primary consideration in all design decisions. Prior segmentation research is notorious for developing automated methods in which the user has no means for correcting errors made by the system. Consequently, this work generally opts for semi-automatic methods, which offer the user more opportunity to deal with mistakes.

The scope of this dissertation is unusually broad. By tackling the full problem, it sheds light on some of the technical details that are often left unaddressed. I argue that this approach has led to more robust tools which stand a better chance of being useful in real-world situations.

1.4 Chapter Overview

Chapter 2 presents background information about the fields of vision and graphics, describing the relevant issues and ways in which other researchers have approached them. Chapter 3 presents two previously proposed methods for representing shape and analyzes their suitability for 2D image segmentation. Chapter 4 extends the 2D methods to three dimensions and shows how the *radial surface model* can be used to reconstruct a low-resolution version of the human brain. Chapter 5 describes the *Brain Mapper*, a working software system for reconstruction and mapping of the human brain, which puts the radial shape model to use. Chapter 6 addresses illustration. Examples show how non-standard rendering methods can enhance anatomical images, leading to proposed requirements for and an early prototype of a system for medical illustration. Chapter 7 summarizes this work and outlines possible avenues for future exploration.

Readers interested primarily in reconstruction should read sections 2.1–2.3 of the background chapter, along with chapters 3–5. Readers interested in visualization and illustration should focus on section 2.4 and chapter 6.

In truth, visualization issues permeate the entire process of transforming volume data into a 2D illustration. In an interactive system, the user needs constant feedback about the current state, and this feedback generally takes the form of pictures. Thus long before the "final" illustration is created, a variety of other pictures will have been employed to show the user what is happening. For this reason, visualization issues will provide a recurring theme throughout this dissertation, in addition to receiving special focus in chapter 6.

1.5 Contributions

This dissertation synthesizes work from many research areas. Its primary contributions are:

- an in-depth analysis of two methods that use shape knowledge to guide 2D image segmentation
- a novel method for 3D brain segmentation, which offers intuitive user controls and explicitly addresses the removal of the skull and other surrounding structures
- a working, interactive software system for reconstructing, mapping, and visualizing the human brain
- a list of requirements for and a prototype of a medical illustration system

Chapter 2

BACKGROUND

The goal of examining organs from medical images is not a new one. On the contrary, much research has been devoted to this task over the years. This chapter lays the groundwork for the rest of the dissertation by highlighting the main issues involved with reconstruction and illustration, and by describing how other researchers have approached them.

One of the fundamental concerns in any computing domain is data representation. Therefore this chapter begins by discussing some of the standard methods for representing real-world objects. This is followed by descriptions of major methods for medical image segmentation, beginning with the most common low-level segmentation strategies and moving on to more complex methods such as deformable models. The final section addresses illustration issues, describing prior work in non-photorealistic rendering and illustration systems.

2.1 Object Representation

A complete analysis of the various methods for representing objects is beyond the scope of this work, as it has been known to fill multiple book chapters[26]. Therefore the discussion will be limited to a brief comparison of two broad classes of representation: *region-based methods*, which describe the complete contents of an object; and *boundary-based methods*, which describe its perimeter or surface only. This section will also present a third method, known as a *modal representation*, which combines aspects of the other two and has been gaining popularity in computer vision research.

2.1.1 Region-based representations

Region-based methods describe an object as a whole, rather than just its surface. Examples include implicit surfaces, constructive solid geometry, and binary images and volumes. One appealing feature of most region representations (with the exception of volume datasets) is that they can be stated compactly. In addition, they tend to be unique. The major drawback with region-based methods is that they must be converted into triangulated surfaces to take advantage of the hardware rendering support provided by current graphics workstations.

2.1.2 Boundary-based representations

Boundary-based methods describe the surface of an object without explicitly describing its interior. Examples include polygonal meshes and spline patches. An advantage of surface representations is their flexibility; they can be used to describe a wide variety of surfaces. One major problem with boundary-based methods, though, is that they may be ill-formed, either because they contain self-intersections or because they do not describe a closed object. Even when a surface corresponds to a well-defined object, computations that involve its implicit interior (collision detection, for instance) can be time-consuming. Boundary-based representations also tend not to be unique, meaning that there may be many ways to represent the same object. This makes it difficult to perform shape comparisons between objects, for example, because two similar surfaces may be described very differently.

Of the various methods for surface representation, triangle meshes are the most widely used. Since this is the input format for most current graphics workstations, it works well when fast rendering of 3D scenes is needed. On the other hand, planar triangles can only approximate the surface of a curved object; hundreds of thousands of tiny triangles may be needed to achieve a reasonably close match. Since objects made from triangle meshes do not have a very compact representation, they can also be difficult to edit.

2.1.3 Modal representations

In an attempt to join the advantages of the region- and boundary-based methods, Pentland and Sclaroff developed *modal representations*[52]. With their approach, the finite element method (FEM)[7] is used to represent an object as a deformation of some prototype shape. A set of orthogonal parameters is obtained by computing the eigenvectors of the FEM equations. (These eigenvectors are also known as the object's free vibration or deformation *modes.*) When an object is represented in terms of these modes, its description will be unique and compact, much like the volume-based representations. This makes modal representations an attractive alternative for shape matching. In addition, the FEM maintains the expressiveness of boundary-based methods.

2.2 Volume Rendering

Before discussing segmentation issues, it is necessary to mention that an image-to-model-to-image approach is not the only alternative for viewing structures from volume data. For many years, *volume rendering* has been used for visualizing complex 3D datasets, especially medical imagery. This method bypasses any intermediate 3D object models, and instead uses information such as tissue density to render directly from volume data[24].

One advantage of volume rendering is that it works without explicit segmentation of objects. It can also reproduce detail that would be difficult to model, such as the porous internal structure of bones[28]. However, the lack of explicit models makes direct manipulation of volume-rendered objects impossible. In addition, volume-rendered images often look out-of-focus and indistinct, making them less suitable for some visualization purposes than those generated from surface-based models.

2.3 Tools for Image Segmentation

Each time an image segmentation system is built, the techniques used are influenced by the particular problem that is being solved. Consequently a wide array of segmentation tools have been developed over the years. A few basic questions can be used to build a taxonomy of the available methods:

- Is it 2D or 3D? In other words, does it deal with a single cross-sectional image or with a volume of data?

- Is it boundary- or region-based?
- Is the method generic, or tailored to find a specific structure?
- Does it use prior knowledge?

Despite this diversity, there are certain basic methods that get frequent use. This section reviews some of the most common low-level tools, along with more sophisticated techniques that can compensate for the shortcomings of the low-level methods.

2.3.1 Low-level feature detectors

One class of low-level feature detectors comprises edge-based methods, which are used to identify object boundaries. Typically, these methods locate edges that correspond to local peaks in the intensity gradient of an image. Such a strategy works well, for example, when trying to find the boundary between two differently colored objects. Edge detection is also fast, because the decision made at each pixel depends only on local information. However, this locality trait can lead both to spurious boundaries in highly textured areas and to missed boundaries where adjacent objects are not well-delineated.

Region-based methods offer an alternative approach to low-level image segmentation. Rather than looking for boundaries between objects, they break an image into "homogeneous" regions of pixels, where homogeneity is usually defined according to a cost function based on the intensity of a pixel relative to its neighbors. One example of a simple criterion would be: "all pixels in a region must have the same intensity value." One of the simplest and most common region-based methods is *thresholding*, which divides pixels into two classes, depending on whether their intensities are above or below a specified threshold value. A slightly better method is *region growing*, which begins with a set of single-pixel regions and progressively enlarges each one by adding neighboring pixels that satisfy the cost function. However, thresholding and region growing suffer from the same locality problems as edge-based methods. For example, if the two similarly-colored structures share a small boundary, region growing methods will often merge them into a single region.

More sophisticated region-based segmentation can be achieved using *clustering.* This approach attempts to find the "natural subregions" of the image and to label each pixel with its appropriate group. Depending on the method used, pixel labels may be crisp (specifying membership in exactly one region), probabilistic (specifying the likelihood of membership in each region), or fuzzy (specifying an "affinity" for each region). The most common approach to clustering is the fuzzy c-means algorithm[9]. After the user chooses the number of subregions, c, the algorithm finds the image clusters—typically by minimizing a sum-of-squared-errors objective function—and colors the image according to the computed pixel labels. The user then attaches logical labels to each colored region (*e.g.* "yellow is bone"). Promising results have been achieved with cluster-based segmentation, especially for MRI analysis[8]. The computational complexity of clustering methods, however, makes them slow in practice. In addition, the amount of user interaction required to choose the number of clusters and label the results has been seen as a detriment.

2.3.2 Knowledge-based methods

One way to improve the accuracy of low-level segmentation methods is to incorporate knowledge about the shapes of the objects being sought. Such a strategy relies on the availability of *a priori* information about the types of structures that will appear in the image. While this assumption would not be appropriate for computer vision tasks involving arbitrary scene compositions, it is perfectly acceptable for the domain of medical image analysis, because the basic shape and arrangement of anatomical structures will be similar from person to person.

Shape knowledge can be applied to segmentation in a number of ways. It can guide the search for edges by specifying a region of interest where a particular object will most likely be found. It can compensate for limitations of the imaging technology, for example, by completing an object's boundary where no edge information is available. It can also be used as a diagnosis tool, by identifying places where expected shape relationships are not met.

A basic image processing tool that uses shape information is *mathematical morphology.*

Morphology is commonly used on binary images or volumes to clean up an object's boundary by adding or removing portions that have a particular shape characteristic. The technique works by comparing neighborhoods in the data to a small *structuring element* that encodes the desired shape feature. Morphology is rarely used by itself for segmentation, but it does provide a simple method for fixing local mistakes made by other low-level feature detectors.

A higher-level method for incorporating shape knowledge is *Bayesian inference.* In the context of image segmentation, Bayesian inference is a tool for determining the likelihood that a particular unknown object $\mathbf{u}$ is present in a scene, given that sensor data $\mathbf{d}$ (*i.e.* the image) was observed. In Bayesian terms, this likelihood is called the *posterior model*, $P(\mathbf{u}|\mathbf{d})$, and it can be computed using Bayes' Law:

$$P(\mathbf{u}|\mathbf{d}) \propto P(\mathbf{d}|\mathbf{u})P(\mathbf{u}) \tag{2.1}$$

where $P(\mathbf{u})$ is a *prior model*, which expresses the probability that object $\mathbf{u}$ will be present (regardless of the sensor data), and $P(\mathbf{d}|\mathbf{u})$ is a *sensor model*, which expresses the probability that the sensor will generate data $\mathbf{d}$ given that $\mathbf{u}$ is present. Typically, these prior and sensor models are computed by statistically analyzing a set of training images.

An example of a system that uses shape knowledge is the work of Kobashi and Shapiro, which segments several structures in CT images of the abdomen[38]. In addition to checking traits such as region size and location, the system employs negative shape constraints to rule out regions with unacceptable shape properties. (For example, very elongated regions would not correspond to the aorta, which tends to be circular in axial CT images.)

Explicit geometric models offer another way to represent shape knowledge. Rigid 2D and 3D models have worked well for machine vision applications such as part inspection, where images are searched for instances of a small number of objects whose shapes are exactly known beforehand. In the more general case, geometric models may be parameterized to describe a class of objects, rather than a single specific instance. For example, a rectangular model with parameters l and w for its length and width will match more objects than a model of a square with sides of $l = w = 1$. Yet even parameterized rigid models will have limited use for the medical domain, where flexible structures are the norm. For that reason, many researchers have turned to *deformable models* for solutions.

2.3.3 Deformable models

Deformable models offer an alternative to rigid geometric models. They treat the surface of an object as an elastic sheet that will stretch and deform when forces are applied. Such models are often called *physically-based*, because their behavior is designed to mimic the physical laws that govern real-world objects. Deformable models were first introduced as a tool for image segmentation by Kass *et al.*[37]. In their formulation, known as *active contour models* or *snakes*, image segmentation is posed as an energy minimization problem.

The 2D problem can be stated as follows, adopting the notation used by Szeliski and Terzopoulos[61]. Given a contour $\mathbf{v}(s) = (x(s), y(s))$, with parameter $s \in \Omega = [0, 1]$, an energy function for the contour can be stated as

$$\mathcal{E}(\mathbf{v}) = \mathcal{E}_s(\mathbf{v}) + \mathcal{P}(\mathbf{v}) \tag{2.2}$$

where $\mathcal{E}_s$ is the contour's internal energy, and $\mathcal{P}$ is an external potential that acts on the contour. The contour will be attracted to local minima of the potential function, so $\mathcal{P}$ usually involves the inverse gradient of the image, thus pulling the contour toward strong edges. The internal energy of the contour is specified by

$$\mathcal{E}_s(\mathbf{v}) = \int_\Omega w_1(s)|\mathbf{v}_s|^2 + w_2(s)|\mathbf{v}_{ss}|^2 ds \tag{2.3}$$

where subscripts on $\mathbf{v}$ denote differentiation, and w_1 and w_2 are weighting functions. The first term of this integral measures the contour's ability to stretch (*i.e.* behave like an elastic membrane), and the second term measures its ability to bend (*i.e.* behave like a thin plate). The contributions of these two forces, relative to each other and to the influence of the image through $\mathcal{P}$, can be changed by adjusting the weighting functions w_1 and w_2. In practice, these weights are usually assumed to be constant over the length of the contour.

To simulate deformable models with a computer, the formulation above must be discretized. The discrete version can be thought of as a mesh of masses connected by springs. If the continuous contour $\mathbf{v}$ is approximated with node variables $\mathbf{u}_i$, the quadratic energy in (2.2) can be written as

$$E(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} + P(\mathbf{u}) \tag{2.4}$$

where $\mathbf{K}$ is a stiffness matrix that expresses how the springs can stretch and bend, and $P(\mathbf{u})$ is the discrete version of the external potential $\mathcal{P}$. This energy will be minimized where its gradient is zero; this location can be found by solving

$$\mathbf{Ku} = -\nabla P = \mathbf{g} \tag{2.5}$$

where $\mathbf{g}$ is a generalized force vector that expresses the loads impinging at each node.

If the system is changing over time (for example, if the potential function is changing), then Lagrangian dynamics can be used to express how potential energy in the contour changes to kinetic energy. In this case, the energy minimum can be found by solving the following system of second-order differential equations:

$$\mathbf{M\ddot{u}} + \mathbf{C\dot{u}} + \mathbf{Ku} = \mathbf{g} \tag{2.6}$$

where $\dot{\mathbf{u}}$ and $\ddot{\mathbf{u}}$ are $\mathbf{u}$'s first and second derivatives with respect to time, $\mathbf{M}$ is a mass matrix that expresses the masses of the nodes, and $\mathbf{C}$ is a damping matrix that expresses how rapidly energy in the system dissipates.

When an active contour model is placed on an image, solving this set of equations yields positions along a curve that minimize the energy function. Many sections of the curve will follow edges in the image, but the internal energy constraints will also keep the curve relatively smooth. It is important to note that an active contour model performs local energy minimization only, so it will converge to the correct solution only if its initial position is close to the desired one.

2.3.4 Specialized deformable models

At their simplest, deformable models employ very little in the way of domain-specific knowledge. The internal energy term of the cost function merely imposes a uniform preference for smooth boundaries. However, by adding other terms to the cost function, this generic method can be tailored to achieve better performance for specific applications.

An example of such specialization can be found in the work of Davatzikos and Bryan, who developed a deformable surface that is tailored for finding the surface of the brain in MR volume data[21]. The convoluted nature of the cortical surface provides a significant challenge for deformable models. Throughout the cortex, there are regions where the tissue has folded over upon itself to form deep grooves, or *sulci*, that descend toward the center of the brain. But the tops of the adjacent ridges, or *gyri*, are usually so close together that the valleys are nearly invisible at the surface level; the surface appears to continue straight across, with only a slight indentation where the valleys occur. Therefore Davatzikos and Bryan propose a model with two components: (1) a standard, mesh-based deformable model that captures the detail present at the surface, and (2) a depth map that measures distances from this surface mesh to the bottoms of the sulci.

After isolating the brain's gray matter using a combination of region growing, morphology and classification methods, a deformable surface is fitted to volume data. The gray matter is basically a distorted sheet of tissue with nearly uniform thickness. Therefore, the cost function of their specialized model includes two extra external forces that take this property into account. The first force has the effect of shrink-wrapping an elastic surface around the brain; points that are far outside the gray matter are pushed inward along the model's surface normal. When they get close enough to the cortex, a second force takes effect and pulls the points toward the center of the gray matter sheet. In addition, the internal energy forces of the model have been modified to omit the standard bending energy term. This component penalizes surfaces with lots of kinks, so including it would hinder the surface from conforming to the natural curves of the cortex.

Another specialized approach is the *active shape model*. Developed by Cootes *et al.*, the active shape model permits only those deformations that are characteristic of the object it represents[18]. The model's constraints are derived from training data. Example images of the object of interest are segmented by manually marking the locations of predetermined reference points around the object's boundary. The n vertices on each contour, each with x and y coordinates, can be thought of as a defining a single point in $2n$-dimensional ($2n$-D) space. Thus the training set forms a $2n$-D point cloud, and points within the cloud correspond to other contours with roughly the same shape. Assuming this cloud is approximately ellipsoidal, its eigenvectors can be computed to identify its principal axes of variation. The eigenvectors form a basis for the $2n$-D space, so they provide a set of parameters that are uncorrelated over the training set. For each parameter, a mean and variance can be computed to find the range over which it tends to vary in the example shapes. The important point is that since the parameters are orthogonal, they may be freely varied in their prescribed ranges, and the corresponding contour will still maintain the shape characteristics of the training set.

Once a model has been computed from a training set, it can be used to segment a new image by applying the same idea used with active contour models. Given an initial curve that is in the neighborhood of the correct object, a force is applied to pull it toward edges in the image. To maintain the model's parameter constraints, the displacements computed at each step are transformed into model parameter space, and truncated to stay within the parameter bounds. The resulting solution is equivalent to computing a least squares approximation of the external forces and shape constraints. Later work combines this approach with a finite element model to handle situations when there is not enough training data available to build a good statistical shape model[17].

2.3.5 Atlas-based methods

The method just described is boundary-based; it attempts to deform a prototypical contour to match the perimeter of an object. But the same principle can be applied in a region-based approach. Suppose that instead of representing the object boundary as a chain of points, the entire *image* were represented as a grid of points connected by springs. By applying the same energy minimization technique, the whole image would be warped to match the test data.

This is the strategy used by a fairly recent and promising approach known as *atlas-based segmentation*[27, 63]. This method warps a pre-segmented image (the *atlas*) to match a target image. If the warp is done correctly, then the target image will be segmented automatically; any structures which have been delineated in the atlas will be mapped onto the target. This makes it possible to do one very detailed segmentation of one image by hand, then apply the results to many other images.

2.4 Illustration

2.4.1 Non-photorealistic rendering

While the computer vision community attempts to extract geometric models from images, the computer graphics community works in the other direction, making images from geometric models. During the past two decades, the Holy Grail of the graphics community has been *photorealism*—generating images that are indistinguishable from photographs. Research papers with this goal frequently show side-by-side comparisons of a photograph and a computer-generated rendering of the same scene. Efforts to simulate the camera's optics have even extended to the point of replicating photographic artifacts like lens flare. By the mid-1990s, films such as *Jurassic Park* and *Forrest Gump* made it clear that computer-generated imagery could be integrated seamlessly with live-action footage. In this context at least, the goal of photorealism had been achieved.

At about the same time, graphics researchers began turning their attention to topics that had been ignored in the push for photorealism. After all, the camera provides merely one method for capturing an image of the world, and a relatively recent one at that. Artists and illustrators have been doing the same thing for centuries, using an array of painting and drawing techniques that had been mostly neglected by the graphics community.

Collectively, these various areas of inquiry have come to be known as *non-photorealistic rendering (NPR)*.[1] Most early NPR work attempted to imitate other traditional styles of image production, such as painting. One of the first was Paul Haeberli's Impressionist, which rendered pixels not as dots, but as short brush strokes to create the effect of a painted image[31]. Others have expanded upon this idea to create a variety of "painterly" rendering styles[33, 39, 46]. Researchers have also designed methods for imitating pen-and-ink illustration[55, 68].

More recent work has achieved convincing images through physical simulation of traditional media. For example, Curtis *et al.* create impressive watercolor effects by modeling

---

[1] "Non-photorealistic rendering" is an unfortunate label for this area of research, as it describes merely what the work is *not* trying to achieve. Such naming can be difficult to avoid, given the prevalence of terms like "post-modern." The author hopes a more flattering label—perhaps "artistic rendering," "illustrative rendering," or even just "illustration"—will be attached to this body of work in future years.

the interaction of water, pigment, and paper[20]. Similar work by Sousa and Buchanan generates realistic pencil drawings[59].

Methods from technical illustration have also received limited attention. For example, Feiner and Seligmann investigated methods such as cutaways and ghosting to reveal important objects that otherwise would be hidden in a traditional rendering[25]. Gooch *et al.* showed how to modify the standard lighting model to implement a "cool-to-warm shift"—a method used frequently by technical illustrators to limit the tonal range of an object without losing important shape cues[30].

2.4.2 Illustration systems

Although individual illustration techniques have received attention recently, relatively little work has been done in the area of illustration systems. More is implied by the term "illustration system" than a simple drawing application. Rather, it refers to a system that can generate illustrations dynamically from 3D models. One of the most well-known is the Intent-Based Illustration System (IBIS), which generates technical illustrations on-the-fly, based on high-level design goals specified by the user[57]. Another system, the E^3 Project, designed interactive illustrations for a "how things work" manual of everyday objects like refrigerators[5].

The work of Dooley and Cohen[22, 23] provides some of the most promising ideas for an useful illustration system. To support automated illustration, they turn to the techniques used by traditional illustrators. They propose the use of new display primitives to help clarify structure, including methods for the user to specify the importance of various model components. Many of their ideas will be echoed in the work presented in chapter 6.

Chapter 3

2D SHAPE MODELS

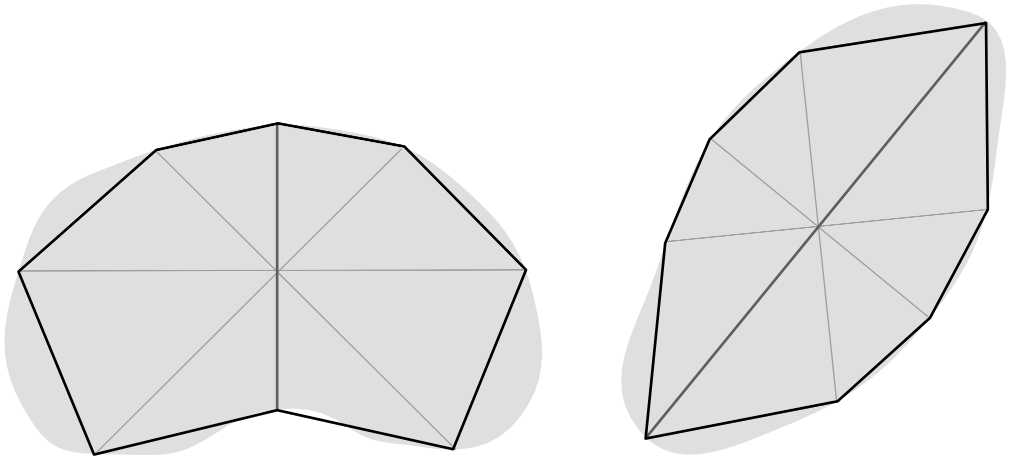
This chapter lays the groundwork for 3D segmentation by first considering the 2D problem of finding a structure in a single medical image. Since the goal is to use shape knowledge to guide this search, the issues involved when representing shape and shape knowledge are presented. With those issues in mind, two previously proposed methods for modeling shape are tested on a variety of anatomical structures. The results will show that a relatively simple network of interrelated shape constraints can significantly enhance an image segmentation tool. A semi-automatic system that uses this approach requires two-thirds less work than a fully manual one.

3.1 Representing Shape

As shown in chapter 2, there are many ways to represent the shape of an object. However, the representation used will affect the ease with which different types of information can be computed from it. For example, it is easier to compute a shape's area from a region-based representation than from a boundary-based one. Choosing a good shape representation inevitably depends on the definition of "good," and that definition will vary from problem to problem.

Consider the task of comparing shapes of objects extracted from images. When searching for an object in a collection of images, that object will rarely occur in the same position every time. It is also unlikely that it will always be the same size, or even that it will have the same orientation. This is especially true with medical images taken from different individuals. Consequently, three highly desirable features in this problem are invariance to position, scale, and orientation.

An example of a representation that does *not* have these invariance features would be a list of (x, y) coordinates that form a shape's boundary. Suppose you had two such lists,



$r = [1.9\ 2.4\ 3.3\ 3.3\ 1.9\ 3.4\ 3.5\ 2.3]$ $r = [3.6\ 2.2\ 1.9\ 2.4\ 3.6\ 2.0\ 1.9\ 2.4]$

Figure 3.1: Radial contour approximations of two shapes. Each of these examples uses eight radials spaced at 45° intervals, with the darker line denoting the reference axis. The numeric encoding shows the length of each radial, starting with the upper half of the reference axis and proceeding clockwise.

describing two objects extracted from two different images—in other words, two connect-the-dots pictures in which the dots had not yet been connected. Do the objects have the same shape? Are they even remotely similar? These questions would be difficult to answer (though not impossible), because the coordinate-list representation does not lend itself to such a comparison.

The methods described in this chapter use a representation called a *radial contour*. Each contour is a closed polygon stored in polar coordinates with the parameterization $r(\theta)$. In other words, the points forming the contour can be thought of as the tips of spokes, or *radials*, emanating from a central point inside the contour. Two values are therefore associated with each radial: an angle (the direction in which it leaves the center point) and a length (the distance from the center point to the object's boundary). In theory a radial contour is continuous, but for practical use it must be represented with discrete samples measured at particular angles. Two examples of radial contours with eight samples each are shown in figure 3.1.

The exact position and orientation of these radials are determined by a *reference axis*

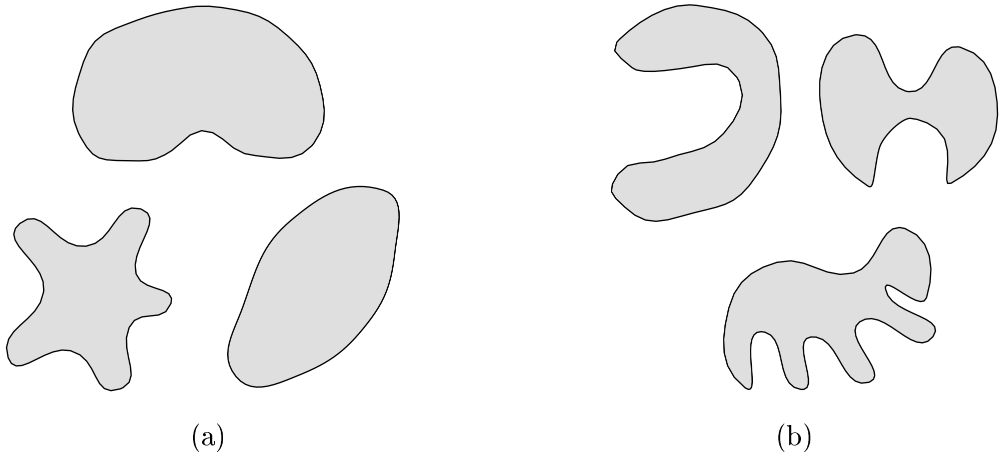

(a) (b)

Figure 3.2: Examples of (a) star-shaped and (b) non-star-shaped contours.

for the contour. This axis can be chosen arbitrarily, but is typically aligned either to an axis of symmetry or to the direction in which a shape is most elongated. The endpoints of the reference axis touch the object's boundary, and its midpoint provides the central point from which other radials extend. (Note that the reference axis itself actually provides two radials for the radial contour.) Since radials are specified relative to it, the reference axis provides a local coordinate system for the radial contour.

Some additional limits increase the usefulness of the radial contour representation. Radials are restricted to be evenly spaced, which means the angle between any two adjacent radials in a contour is the same. It is also assumed that each radial will intersect the contour exactly once; contours meeting this constraint are sometimes called *star-shaped*. These restrictions eliminate self-intersections. In addition, the connectivity of the polygon becomes implicit, with its edges formed by proceeding sequentially along the endpoints of the radials. The primary disadvantage of these simplifications is that they restrict the range of shapes that the radial contour can represent. As figure 3.2 shows, many objects are not star-shaped. Yet for domains such as radiation treatment planning, star-shaped approximations of the organs of interest will frequently suffice.

But by sacrificing the ability to represent every possible shape, we gain the critical ability to compare multiple shapes more easily. By using a predetermined layout of radials relative to the reference axis, this representation provides automatic registration between radial contours. In effect, the reference axis factors out issues of a structure's location and orientation, leaving only issues of scale and shape variation to be accounted for. Once the reference axis has been determined for two structures, the correspondences between all of their other boundary points will also be known. It therefore becomes possible to make *comparisons* between corresponding sections of their boundaries.

This ability to compare corresponding boundary regions is the key to extracting shape knowledge from a group of similar structures.

3.2 Representing Shape Knowledge

More interesting than representing a single shape is the issue of representing shape knowledge. In other words, how can the notion of objects with "similar shapes" be quantified? With a single object, the goal is generally to capture as many significant details as possible. With a group of objects, the goal is usually twofold: (1) getting a sense for what makes all of the objects similar, and (2) capturing the ways in which individual cases deviate from the norm.

Given a set of radial contours as training data, shape information can be extracted and used to build a model describing the set. By gathering data from a collection of similarly-shaped radial contours, a *radial contour model* can be built to describe them. From a segmentation standpoint, the intent is that this model will be able to guide the search for structures which share this shape. During segmentation, boundary features extracted from the image can be used to focus attention on the range of shapes in the generic model which are also consistent with the contour in the image.

Deciding which information to extract—and abstract—becomes the challenge. This section presents two previously proposed approaches to representing shape knowledge; one derives its constraints from ratios of radial lengths, while the other uses probabilistic methods.

3.2.1 The min/max shape model

The first type of model under consideration is Brinkley's *min/max shape model* (or *M-model*)[11]. The M-model uses ratios of radial lengths as local shape features. For a pair of radials i and j with lengths r_i and r_j, the ratio is simply $s_{ij} = r_i/r_j$. Radials for which a length ratio is computed are referred to as *neighbors*.

Once shape features have been measured for a set of contours, they can be used to derive constraints for an M-model. For each pair of radial neighbors i and j, the M-model stores a lower bound, M^-_{ij}, and an upper bound, M^+_{ij}, which correspond to the minimum and maximum values observed in the training set for ratio s_{ij}. These bounds comprise the model's shape constraints.

The total number of constraints in the M-model depends on how neighbors are defined. Brinkley's original work tested two variations: a local version, with constraints between adjacent radials only, and a maximal version, in which constraints were used between every pair of radials. This work uses the maximal version of the M-model, because it performs slightly better than the local version.

The constraints of the M-model capture information about relationships between radials, rather than information about the radials in isolation. This makes it difficult to draw a picture of the constraints. An instantiated M-model, shown in figure 3.3, is easier to visualize. An instance of an M-model can be derived from a reference axis; the two halves of the axis provide lengths for two of the contour's radials. These can be used in conjunction with the M-model's shape constraints to derive *uncertainty bounds* for the remaining radials. These bounds delineate the range of lengths that each radial can have while still satisfying the shape constraints of the M-model.

Suppose that radial j is initialized to have length r_j. For each of its neighbors i, the constraint $M^-_{ij} \leq s_{ij} \leq M^+_{ij}$ can be used to infer that $M^-_{ij} r_j \leq r_i \leq M^+_{ij} r_j$. The neighbors of j are now bounded, because the constraints have been used to rule out values that were not observed in the model's training set. The updated neighbors can be used in turn to bound *their* neighbors, creating a wave of updates that propagates through the entire contour. Figure 3.4 shows pseudocode for the propagation algorithm.

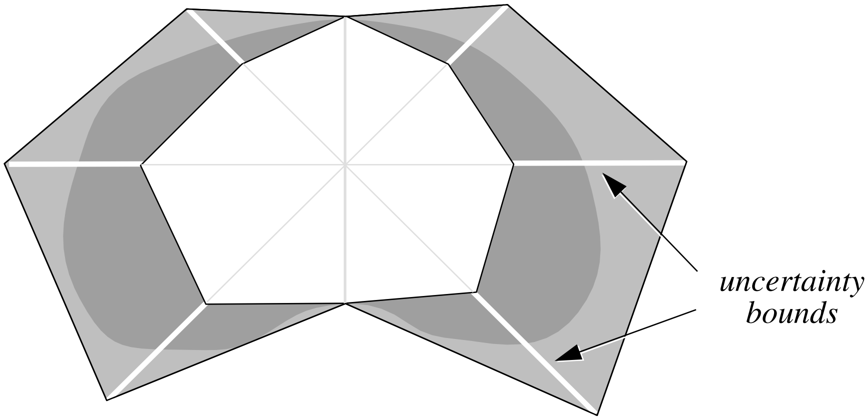


Figure 3.3: An instantiated M-model. The model's shape constraints generate uncertainty bounds, which can be connected to define a region of interest in which the object's boundary is expected to lie.

One appealing feature of the M-model's ratio constraints is that they are automatically scale-invariant, because they depend on relative lengths of radials rather than absolute lengths.

Complexity analysis

Previous work has demonstrated that in the M-model, each constraint propagation operation has complexity $O(n)$, where n is the number of radials[11]. If edges are found for the majority of the radials, then the total complexity is $O(n^2)$.

3.2.2 The probabilistic shape model

An alternate approach to representing shape constraints, which was proposed by Altman and Brinkley[3], is the probabilistic shape model (or *P-model*). Like the M-model, the P-model derives its shape constraints from training data. However, these constraints are based not on ratios of radial lengths, bur rather on covariance between radial lengths. Furthermore, the P-model relies on a Bayesian technique known as Kalman filtering[36] to incrementally update its estimate of the contour location as edge information is gathered. The notation used here is adopted from Welch and Bishop's excellent introduction to Kalman filtering[66].

```
procedure propagate(float length, int k, Model M, Contour C) {
    Q = new(Queue)
    C[k].lo = C[k].hi = length
    enqueue(Q,k)
    while notEmpty(Q)
        j = dequeue(Q)
        for each i in neighbors(j)
            d.lo = C[j].lo * M[i][j].lo
            d.hi = C[j].hi * M[i][j].hi
            e = intersect(C[i], d)
            if null(e)
                exit()
            else if subset(e, C[i])
                C[i] = e
                enqueue(Q,i)
}
```

Figure 3.4: The M-model's propagation algorithm. Given a measurement `length` for radial `k`, the algorithm uses the shape constraints of model `M` to tighten the uncertainty bounds for contour `C`.

Consider a radial contour with n radials expressed as a state vector, $x \in \Re^n$, where each component of x corresponds to the length of one radial. The goal of segmentation is to estimate x for a particular organ, based on information gathered from a noisy image by a less-than-perfect edge detector. The Kalman filter is an algorithm for estimating such a state vector, one which can iteratively refine the estimate as new sensor data is acquired and take into account uncertainty in sensor measurements.

To make the mathematics of the problem tractable, three assumptions are required. First, the components of x must have Gaussian distributions. This means that over a set similarly-shaped contours, the lengths associated with a particular radial can be described with a "bell curve"—a reasonable assumption for biological structures. Second, the sensor's noise must be Gaussian and "white" (*i.e.* not correlated in time). Third, the model of how x changes over time must be linear. For the problem at hand, this condition is met because the true contour position is not changing over time, yielding a trivial linear relationship between time steps k and $k+1$:

$$x_{k+1} = x_k. \tag{3.1}$$

Sensor measurements are represented by $z \in \Re^m$ and modeled as

$$z_k = H_k x_k + v_k \tag{3.2}$$

where m is the number of measurements taken, H_k is an $m \times n$ matrix that predicts measurements based on the current state, and v_k represents the sensor noise that causes the measured values to differ from the predicted values. Since the contour x is not changing over time, H_k acts as a binary mask, filtering out those components for which no measurement was taken at step k. For example, if an edge were found for the third radial in a 4-radial contour, H_k would be defined as $[0\ 0\ 1\ 0]$.

Let $\hat{x}_k^- \in \Re^n$ be the *a priori* estimate of the contour location at step k, and $\hat{x}_k \in \Re^n$ be the *a posteriori* estimate given measurement z_k. For each of these estimates, an $n \times n$ covariance matrix of the estimate error can be computed as

$$P_k^- = E\left[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T\right] \tag{3.3}$$

$$P_k = E\left[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T\right] \tag{3.4}$$

where E is the expected value function. The Kalman filter is designed to compute $\hat{x}_k$, given the *a priori* estimate $\hat{x}_k^-$ and measurement z_k, such that P_k will be minimized. This estimate is given by

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k\hat{x}_k^-) \tag{3.5}$$

where the $n \times m$ matrix K is defined as

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{3.6}$$

and R_k represents the covariance matrix for the sensor's noise (v_k). The *a posteriori* estimate error covariance P_k can be computed as

$$P_k = (I - K_k H_k) P_k^- . \tag{3.7}$$

The derivation of these equations is beyond the scope of this work, but can be found elsewhere[45]. Even without the details, equation 3.5 makes some intuitive sense. The new contour estimate ($\hat{x}_k$) is computed by first finding how much the measured radial length differs from its predicted value ($z_k - H_k\hat{x}_k^-$), then weighting that difference by some factor (K_k) and adding it to the previous estimate ($\hat{x}_k^-$). Since K is derived from covariance information, the estimated lengths of radials which are strongly correlated with the measured radial will therefore be affected by this new information. In this sense, the covariance information in the P-model corresponds to the interval constraints in the M-model.

With this formulation, a P-model consists of initial estimates for the contour position ($\hat{x}_0$) and error covariance (P_0). These estimates are found by computing mean radial lengths and the covariance matrix for a set of hand-drawn training examples. As mentioned earlier, a useful property for shape models is scale invariance. To maintain this property when building a P-model, each contour in the training set is normalized relative to its reference axis, prior to the computation of $\hat{x}_0$ and P_0.

Complexity analysis

The computational bottleneck of the probabilistic model is updating the covariance matrix P_k. If edges are introduced one at a time, then equation 3.7 requires multiplying a dense

$n \times n$ matrix with a sparse one; this operation will be $O(n^2)$. If edges are found for the majority of the radials, then the total complexity is therefore $O(n^3)$.

3.2.3 Model comparisons

One drawback of the P-model is that its order of complexity is higher than that of the M-model. This means that as the number of radials used in a model increases, the constraint propagation in the P-model will become disproportionately slower than the M-model.

On the other hand, the P-model has an advantage in terms of how its radial lengths are set. In the M-model, a radial's length stays fixed once a value has been assigned to it. But in the P-model, setting a radial's "length" actually corresponds to specifying a distribution (generally a very narrow one) where the boundary is expected to lie. Unless that distribution has no variation at all, this means that future iterations of the Kalman filter can still influence the radial's length. When the system is using an edge detector to generate edge locations, it inevitably will make mistakes. If the shape model being used is a good one, this additional flexibility in the probabilistic model may make it possible to overcome some errors in edge detection.

3.3 Implementation: The Scanner System

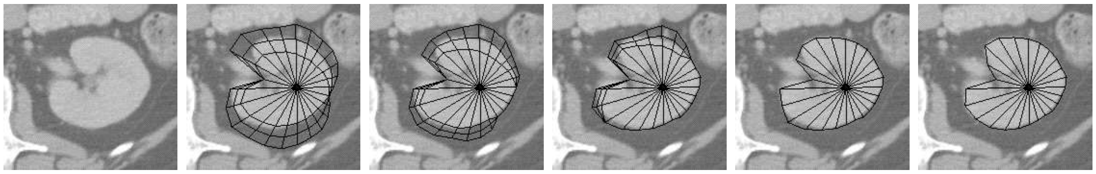
The min/max and probabilistic shape models have been incorporated into an interactive segmentation system called Scanner. This system is based on a previous implementation used by Brinkley to test the feasibility of the M-model[11]. Prior work with the P-model had been simulated only, so this work constitutes the first implementation of the P-model in a working segmentation system.

Figure 3.5 shows the interface for Scanner. Menus at the far right of the window allow the user to manipulate models, contours, and images. The main graphics window shows an abdominal CT image and a radial contour of a kidney. Toggle switches provide control over the display of the radials and contour boundaries. In addition, the user can adjust parameters controlling the automated part of the segmentation process.

Model: "3m[illegible]kidney2.lsp"
Model type: MINMAX
Comment: "shape model"
Radial angle: 15
Contours in Model: 8

Image: "k095.tiff"

Figure 3.5: A screen shot of Scanner. The user has loaded a CT image of the abdomen and has instantiated a radial contour model of the kidney. Hierarchical menus for controlling the segmentation algorithms and manipulating models, contours, and images appear at the far right.



(a) (b) (c) (d) (e) (f)

Figure 3.6: Segmenting the kidney with the radial contour model. (a) Close-up of the kidney region. (b) The instantiated kidney model, derived from the reference axis drawn by the user. (c-d) Intermediate stages of the model during the search-and-propagate loop. (e) The system's best estimate of the kidney's location, after the search-and-propagate loop has completed. (f) The final contour, after user corrections.

3.3.1 Segmenting an image

Figure 3.6 demonstrates Scanner's 5-step process for segmenting an image:

1. The user loads an image (figure 3.6a).
2. From a library of available shape models, the user selects one that matches the object to be segmented (in this case, the kidney).
3. The user instantiates the shape model by drawing the structure's reference axis. By combining the radial lengths from the axis with the model's shape constraints, the system derives uncertainty bounds for the remaining radials (figure 3.6b).
4. Scanner uses a *search-and-propagate* loop to fit the shape model to the image (figures 3.6c-e.)
5. The user corrects errors by clicking on radial endpoints and adjusting them as necessary (figure 3.6f).

The search-and-propagate loop of Step 4 is where most of the work happens. During the "search" phase, Scanner looks along radials to find strong edges that may correspond to the object's boundary. (The first time through, this boundary information actually comes from the user, in the form of the reference axis.) A one-dimensional edge detector examines each radial and flags places where the intensity gradient exceeds some threshold. The most promising of these is selected, and the corresponding radial is set to that length.

At this point, the "propagate" phase of the strategy activates. The edge detector has provided the system with a new radial length. This new information can therefore be used with the model's shape constraints to tighten the uncertainty bounds for neighboring radials. In the case of the M-model, this means applying the constraint propagation algorithm that was outlined in figure 3.4; for the P-model, it means applying the Kalman filter. Either way, the result will be smaller uncertainty bounds for some of the radials in the contour. The system then performs another search, then another propagation, and so on until no more strong edges are found.

The intent of this search-and-propagate strategy is twofold. First, the model's shape information guides the search for strong edges. In any medical image, an edge detector will find an enormous number of edges; the trick is finding the *right* ones. The initial uncertainty bounds limit the search space and therefore increase the system's chances of finding the true boundary. Second, as the clearer parts of the boundary are found, the propagation phase narrows the search space further. Eventually, the inner and outer bounds should converge to the correct boundary, even in places where no edge is visible.

Several possibilities exist for deciding which edge to choose during the search phase. One method is to search all radials and use the edge with the steepest gradient. Another option is to choose the edge that falls closest to the center of its radial's uncertainty interval. The current system provides a menu of edge selection strategies from which the user can choose.

3.3.2 Building shape models

Shape models for several organs are already available in Scanner. However, the system can also be used to build new models. The process is nearly the same as for segmentation, but omits the automated search-and-propagate step. First, the user creates a new and "empty" shape model of the desired type. The user then loads an image containing the desired object and outlines it by hand with a radial contour. A menu option lets the user add these contours to the new shape model, prompting the system to update the model's shape constraints. When finished, the user can add the model to the shape model library for future use within Scanner.

3.3.3 Implementation details

Originally written in Objective C for NeXT computers[11], Scanner was redesigned as a module in Skandha4. Developed primarily by Jeff Prothero, Skandha4 is a hybrid programming environment that provides a Lisp-like scripting language over a foundation of C functions[13]. In addition to writing Lisp code, the programmer can also write C code that provides new Lisp functions. This combination offers the benefits of an interpreted language, such as the ability to modify code on-the-fly, without sacrificing the speed that is crucial in computationally intensive routines.

One of the strong points of Skandha4 is that it provides primitives to support the standard 3D graphics pipeline. The programmer can construct polygonal objects, specify their material properties, position them in a scene, light them, and render the results. Skandha4 also provides the tools necessary for building a graphical user interface (GUI). Unlike most other systems, the on-screen GUI primitives are true 3D objects, so they can be incorporated into a 3D scene along with other objects.

Skandha4 was originally written for Silicon Graphics machines, which provide significant graphics hardware support. More recently, Skandha4 has been ported to work on machines running the Linux operating system. Since Linux runs on standard PCs, which are considerably cheaper than SGI machines, this development will enable a much larger user population to access Skandha4 and Scanner.

3.4 Evaluation

The testing strategy developed originally by Brinkley[11] was reused to validate the two types of radial contour models. For the M-model, this retesting merely confirms the correctness of the new implementation. For the P-model, which had only been simulated previously, it provides the first real gauge of performance in a working segmentation system.

The two shape models were tested on cross-sections of eight biological structures: the eye, kidney, liver, lung, rib, spinal cord, spleen, and vertebra. Shapes were chosen that (1) can be represented by the radial contour model, (2) are critical structures that are frequently segmented for radiation treatment planning, and/or (3) show interesting geometric properties. Since the models are two-dimensional, they can be used to describe only slices of three-dimensional structures. Consequently, multiple shape classes were used to capture the shape at different cross-sectional levels for four of the organs—two for the spleen, and three each for the kidney, liver, and vertebra. Thus a total of fifteen shape classes were used.

A set of 240 CT images was selected from an image archive—16 images for each of the 15 shape classes. The structure on each image was segmented by hand with a 24-radial contour and stored in a database. These hand-drawn contours served two purposes. First, they provided training data for the construction of shape models. Second, they acted as ground truth when evaluating the semi-automated segmentation results.

The 16 images for each shape class were partitioned into groups A and B, each containing eight examples. A shape model was built from each group. Each image was then segmented, using the reference axis from its own hand-drawn contour for initialization, but using the shape model from the opposite group. For example, kidney image #3 in Group A would be initialized with the hand-drawn reference axis from kidney #3, but segmented with the shape model derived from the eight kidney contours in Group B. Thus models were tested on unseen data, rather than on their own training data.

The segmented results were then evaluated by comparing them to the hand-drawn contour for each image. One claim of this dissertation is that a useful segmentation system must allow the user to make corrections interactively. Yet the system should still try to minimize the amount of work the user needs to do manually. Consequently, the "usefulness" of a shape model can measured by counting the number of radials the user must correct per contour; the lower this value, the higher the utility. (In the results reported here, this count does not include the two endpoints of the contour's reference axis, which are always specified by hand.) For testing purposes, it would require too much time to have an actual user inspect each of the 240 contours. Therefore user corrections were simulated, using the assumption that radial i would be corrected if $|r_i - \hat{r}_i|$ exceeds some distance threshold, where r_i is the correct length for radial i and $\hat{r}_i$ is the segmented estimate. The tests described here use a threshold of 4 pixels.

This process was run twice—once using M-models, and again using P-models. To provide a baseline comparison, each image was also segmented a third time, using a *control model* which contained no shape information whatsoever. This provided data on how well the system could do when it had no knowledge about the structure being segmented.

Two parameters for the search-and-propagate loop must be set prior to segmentation: an order for searching radials, and an edge selection rule for cases when multiple candidate edges are found for a given radial. For all of the tests reported here, radials were processed in order of increasing uncertainty interval size. If multiple edges were found along a radial, the one closest to the contour's center was chosen.

In addition, the experiments set the edge threshold at 10% of the grayscale range, assumed that detected edges were specified with a variance of 4.0 square pixels, and searched within 2 standard deviations of the mean for each radial in the P-model.

3.5 Results

Table 3.1 summarizes the performance of the control, M-, and P-models for the complete set of trials. The first column shows the mean and standard deviation for how many corrections were needed for each type of model; lower numbers are better. The M- and P-models required fewer corrections than the control model, suggesting that shape knowledge does

Table 3.1: Overall performance for the three model types. **Corrections** records the average number of radials corrected by the user (out of 22), and **Time** records the elapsed time for segmentation (in seconds).

Model Type	**Corrections** (μ/σ)	**Time** (μ/σ)
Control	9.7 / 6.6	9.9 / 1.1
M-model	6.7 / 6.4	20.8 / 5.8
P-model	5.9 / 6.3	166.2 / 47.4

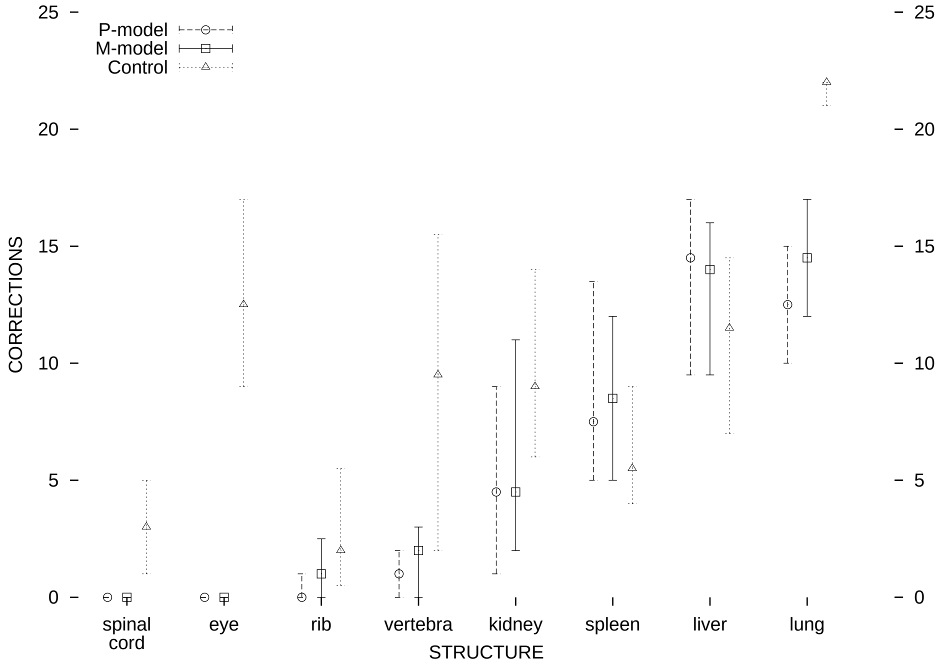


Figure 3.7: Performance of the P-, M- and control models on specific organs. In this modified quartile plot, a symbol marks the median number of corrections, and a vertical line extends outward to the first and third quartiles.

indeed lead to better segmentation. The difference in corrections for the M- and P-models is not statistically significant, but the difference between these and the control model is significant at $p \leq 0.01$.

A breakdown of how the models performed on each shape is shown in figure 3.7. The graph is a modified quartile plot. For each entry, a symbol marks the median number of corrections. A vertical line extends outward to the first and third quartiles (*i.e.* the 25th and 75th percentiles), giving a sense of how much the results vary and whether they are skewed in a particular direction. The results in figure 3.7 suggest that shape knowledge is much more effective on some structures than others. For the spinal cord, eye, rib, and vertebra, the models perform extremely well, making fewer than two errors on average. The models of the kidney and spleen were not quite as effective, but still did reasonably well. The models of the liver and the lung were the least helpful. In general, the liver is a difficult object to segment, because its gray tones are closer to those of surrounding tissue

Table 3.2: Comparison of search-and-propagate (S&P) and search-only (S-only) strategies.

	CORRECTIONS		TIME (S)	
Model Type	**S&P**	**S-only**	**S&P**	**S-only**
M-model	6.7	5.6	20.8	8.3
P-model	5.9	4.9	166.2	5.4

than other organs. Unfortunately the shape models of the liver exhibited high variability, which means they offered less guidance for the boundary finder, and therefore reduced the chances of selecting the correct edges. The lung models, on the other hand, showed very little variability. The uncertainty bounds were small enough that, for many of the radials, the true boundaries fell outside of the search region. Consequently, the edge detector had no chance of finding the proper boundary. This problem could be remedied by using more training examples when building the lung's shape model.

A second set of experiments was run to determine how helpful it is to use shape information to narrow uncertainty intervals during segmentation. This time, the search-and-propagate phase was simplified into a mere search phase. Shape constraints were used to derive the initial uncertainty bounds, but were *not* used to tighten the bounds as edges were found. Table 3.2 shows the rather surprising results. The CORRECTIONS columns show that the search-only (S-only) segmentations were slightly more accurate than those achieved previously (S&P). Furthermore, the TIME columns show that the search-only segmentations took significantly less time to compute. (Figure 3.8 shows the accuracy results on a shape-by-shape basis.) Whereas the P-model was prohibitively slow with the S&P method, it ran over 30 times faster (and achieved slightly better results) when constraint propagation was turned off.

Given these unexpected findings, the original results were examined more closely to find out why gradual narrowing of the uncertainty intervals did not improve segmentation. In many cases, uncertainty intervals were nearly eliminated after two or three edges were found, but the intervals were converging on non-boundary points. Consequently, the edge detector

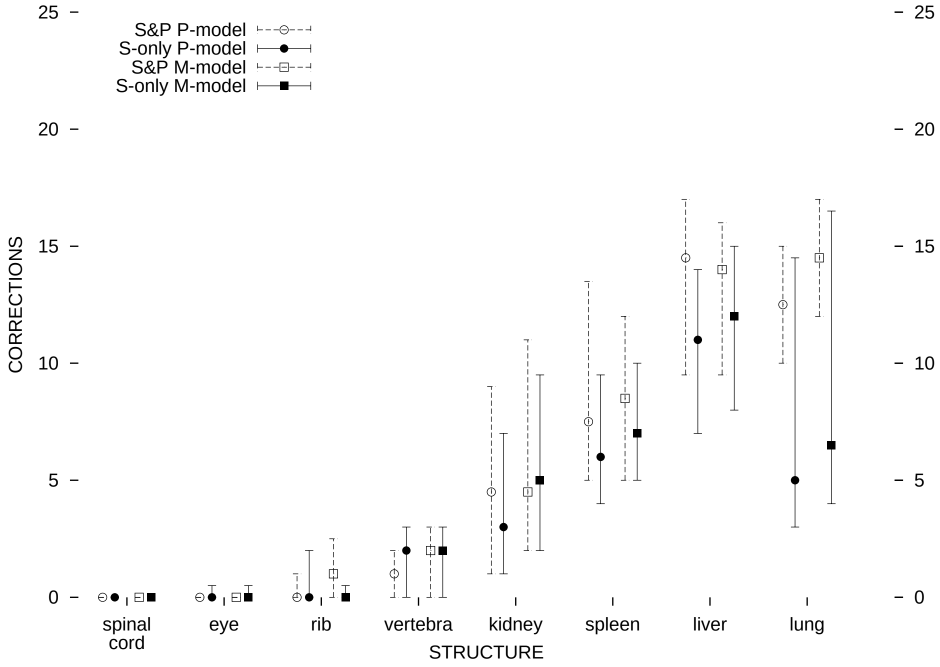


Figure 3.8: Comparison of search-and-propagate (S&P) and search-only (S-only) strategies.

had no chance of finding the correct edges for radials which were considered late in the segmentation process. Two possible causes for this behavior are (1) insufficient variation in the test models, and (2) poor choices by the edge detector early in the segmentation process.

To test the first hypothesis, the evaluation strategy was modified. Rather than testing the models on unseen data, they were used to segment images from their own training sets. By doing so, the true boundaries of each shape were guaranteed to fall within the initial uncertainty bounds for each model. Figures 3.9 and 3.10 show the results for the M-model and P-model, respectively. As expected, the models were more accurate when applied to their own training data (shown with filled symbols) than when applied to unseen data (shown with hollow symbols). Yet the S&P strategy still made several errors, and the S-only strategy still matched its accuracy in most cases. This implies that lack of variation in the test models was not the cause of poor convergence in the first set of experiments.

These results therefore support the hypothesis that poor choices by the system's greedy

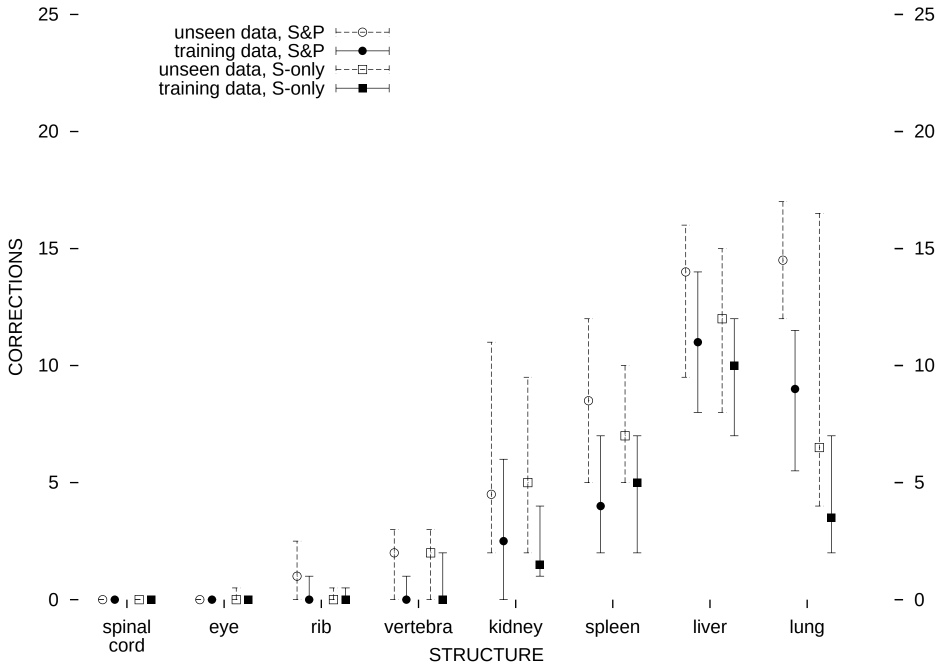


Figure 3.9: M-model's performance on training data vs. unseen data.

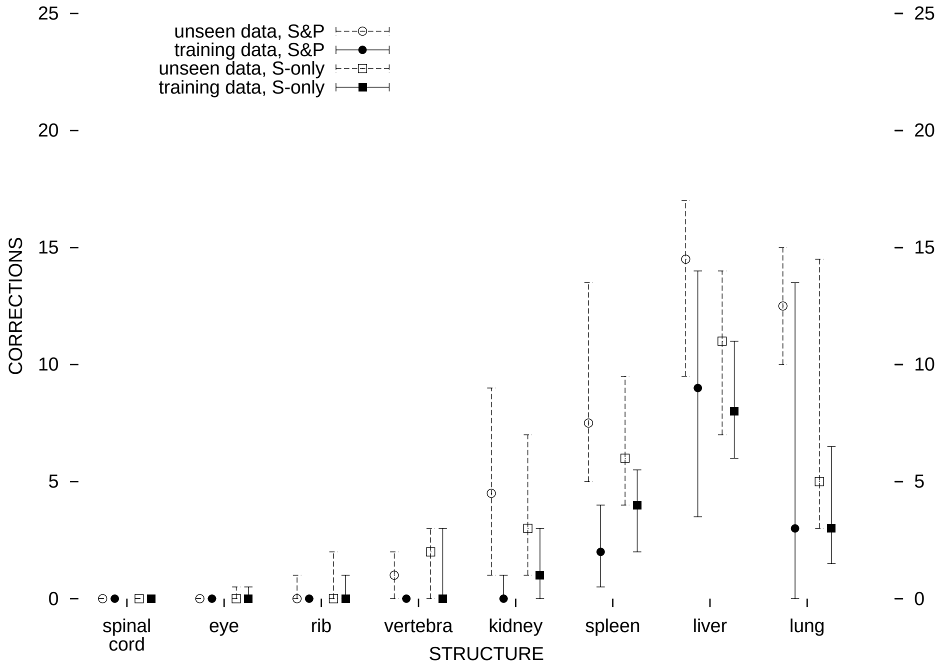


Figure 3.10: P-model's performance on training data vs. unseen data.

edge finder cause the shape model to diverge from the correct boundary. Further testing would be needed to prove such a theory outright, but it seems a likely explanation given the errors inherent with low-level edge detection. Better results might be obtainable by employing a search strategy which either could backtrack to recover from poor choices, or could simultaneously consider multiple edge candidates per radial. One such approach is described in section 7.2.2.

3.6 Conclusions

This chapter demonstrates how prior shape knowledge can be used in a semi-automated system for segmenting medical images. Two previously proposed methods for modeling shape—the M-model and the P-model—were described and implemented in Scanner. Experimental results showed that despite imperfections with the system's greedy model-fitting strategy, both models significantly reduced the user's workload when segmenting an image.

While useful for 2D segmentation, radial contour models have significant limitations when 3D segmentation is the goal. One drawback is that in order to segment a structure from a stack of 2D slices, the user would need to draw a new reference axis on every image. In addition, the shape of an object's cross-sections will vary considerably in different regions. Therefore to segment an entire organ, multiple shape models would be needed, with each one representing a separate range of slices. To make semi-automatic segmentation of image volumes practical, this approach needs to be improved to treat structures as true 3D objects. Such an expansion of the model is the topic of the next chapter.

Chapter 4

3D RECONSTRUCTION

This chapter moves from the 2D world of single image segmentation to the 3D world of reconstructing complete objects from volume data. Since the two types of radial contour models displayed similar accuracy, the simpler one—the M-model—is used as the basis for the *radial surface model (RSM)*. As a sample application, this model is shown to be useful for coarse-grain segmentation of the brain from head MRI scans.

4.1 The Radial Surface

To handle full 3D objects, the radial contour representation was generalized into a *radial surface*. As figure 4.1 illustrates, a radial surface is a stack of slices. The center points for the slices are collinear, forming an axis that runs perpendicular to all the slices. At given intervals along the axis, radials are extended outward in the slice plane to the surface boundary. Each surface also has a local coordinate system that describes its orientation within a 3D space. The coordinate system is derived from landmarks that provide translation, scale, and rotation information. As with the reference axis for the 2D model, these landmarks can be chosen arbitrarily, but they generally correspond to extremal points or axes of symmetry. (For example, section 4.3.1 will show how the landmarks of the Talairach coordinate system can be used to define a radial surface for the brain.) Structurally, the radial surface is equivalent to a stack of radial contours.

As with the radial contour, certain features of the radial surface are simplified to ease the problem of registration. The angle between radials in a slice is kept constant, and slices themselves are evenly spaced. It is also assumed that each radial will intersect the surface only once. Despite these limits on the class of describable shapes, the radial surface representation is flexible enough to be used for shape comparison and for guidance of low-level segmentation methods.

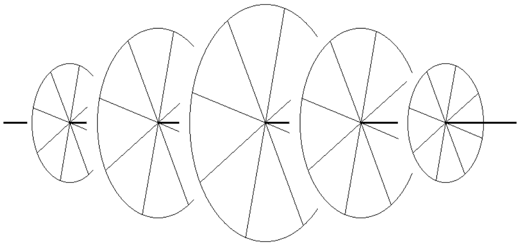


Figure 4.1: The radial surface representation. A series of parallel slices perpendicular to a reference axis, with evenly-spaced radials extending out from the center of each slice to the surface boundary.

User interface considerations strongly influenced this stack-of-slices layout. User-assisted reconstruction of volume data is essentially an exercise in 3D sculpting, a task which poses significant interface challenges and remains an open research problem[43]. Yet the reconstruction task is even more difficult than free-form sculpting—the goal, after all, is getting a close match between the model and the volume data. But how do you display a surface embedded in 3D data in a meaningful way? By using a stack of slices, this difficult 3D interface question was sidestepped, because each surface slice can be superimposed over its corresponding image slice and manipulated as a standard 2D contour.[1]

4.2 The Radial Surface Model

Naturally, a radial surface contains significantly more radials than a radial contour. For example, the brain can be sampled reasonably with 600 radials—20 slices of 30 radials each. Given this increase, performance was a concern when deciding which 2D shape model to use as a starting point. Due to its lower computational complexity, the M-model was chosen.

Thus the RSM tested here uses local constraints to model shape. For purposes of constraint computations, each radial has four neighbors—the two next to it within its slice, plus the nearest radial on each of the adjacent slices. Figure 4.2 shows the arrangement. In

---

[1] The primary problem here is that too much information must be funneled through a two-dimensional information display. More than likely, some clever design exists for an interactive sculpting tool which uses visual feedback only. However, a more fruitful strategy might be to employ the senses of hearing and touch. For example, the resistance of a haptic (force-feedback) device could be tied to the intensity gradient in the volume data, making it harder to chip through boundaries.

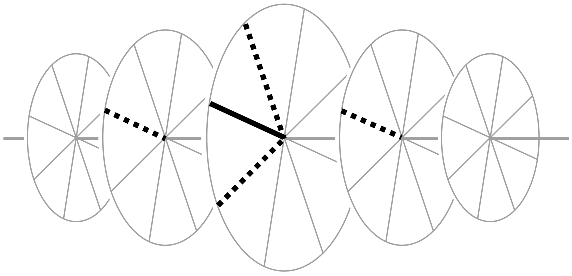


Figure 4.2: A radial and its four neighbors (shown with solid and dotted lines, respectively).

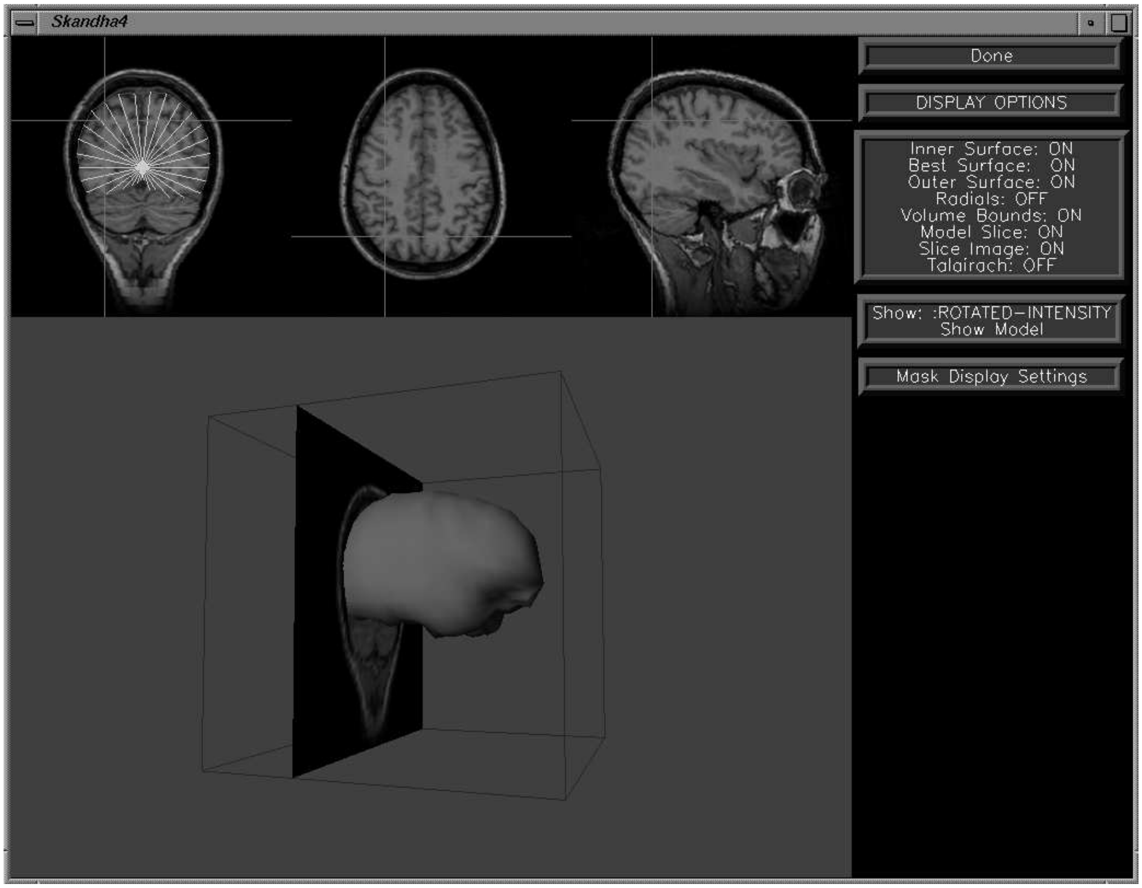
this way, shape information can propagate both within a slice and between slices.

4.3 Implementation: 3D Scanner

To test the radial surface model, a new version of Scanner was built for 3D segmentation. As before, the system was built as a Skandha4 module. Figure 4.3 shows a screen shot. The interface displays three orthogonal slices through the volume dataset: a coronal view on the left, an axial view in the middle, and a sagittal view on the right. On each view, draggable crosshairs control the position of the other two slices. Using these crosshairs, the user can browse quickly along any axis of the volume.

A primary component of the 3D Scanner interface is a surface editor for building radial surfaces. This editor is used both for manual surface creation (to generate training examples) and for correcting mistakes made during the automatic segmentation stage. During surface editing, the current model slice is superimposed over its corresponding coronal image. In a separate viewport, Scanner also shows a 3D view of the surface, along with the current image slice and a box representing the bounds of the volume dataset. Camera controls allow users to adjust the view of the surface.

To segment a volume, the user follows the same 5-step sequence that was described in section 3.3.1: (1) load a volume, (2) load a model, (3) draw landmarks, (4) run the search-and-propagate loop, and (5) make corrections. However, the details of steps 3 and 4 are sufficiently different to warrant further explanation.



Inner Surface: ON
Best Surface: ON
Outer Surface: ON
Radials: OFF
Volume Bounds: ON
Model Slice: ON
Slice Image: ON
Talairach: OFF

Show: :ROTATED-INTENSITY
Show Model

Figure 4.3: Scanner's interface for brain segmentation. Three orthogonal slices through the volume data appear at the top. Crosshairs on the images mark the locations of the other two slices. The surface view in the lower left displays the segmented surface within the volumetric data's bounding box. The surface slice currently being edited is superimposed on the coronal plane at the top left, and the corresponding image slice is displayed in the surface view for reference. The system's menu appears at the right.

4.3.1 Defining landmarks

When working in 3D, defining appropriate landmarks is a more involved task. For brain segmentation, Scanner was designed to use the landmarks of the Talairach coordinate system[62], which is a common standard in neurological studies. A special interface was designed to help the user enter these landmarks for a volume dataset:

1. The *mid-sagittal plane.* (See figure 4.4a.) This is the plane that divides the body into symmetrical left and right halves. Using the MRI crosshairs, the user selects the sagittal slice that is closest to the mid-sagittal plane.

2. The *commissure points.* (See figure 4.4b.) These points mark where the anterior and posterior commissures (AC and PC, respectively) cross the mid-sagittal plane. The user drags the MRI crosshairs until one of the crossings is visible, then clicks on the location to mark it. The points can be moved on the MRI slices to fine-tune their positions; constraints are used to keep them on the mid-sagittal slice. These two points determine the AC-PC line, which is used as the reference axis for the radial shape model of the brain.

3. A *bounding box.* (See figure 4.4c.) The user positions a bounding box around the cortex, which just includes the top of the parietal lobe, the bottom of the temporal lobe, the back of the occipital lobe, the front of the frontal lobe, and the side of the parietotemporal lobe. The box has one axis aligned to the AC-PC line, the second perpendicular to AC-PC in the mid-sagittal plane, and the third perpendicular to the mid-sagittal plane. The user specifies the first four of these bounds using a rectangle on the sagittal MRI cross-section. The lateral bounds are indicated by adjusting lines on the coronal and/or axial cross-sections. It is worth noting that these positions cannot be found with any single MRI slice; the user must scroll through the different cross-sectional views to find the true extremal points. The interface makes it easy to scan quickly through the slices and verify that the true extremal points have been selected.

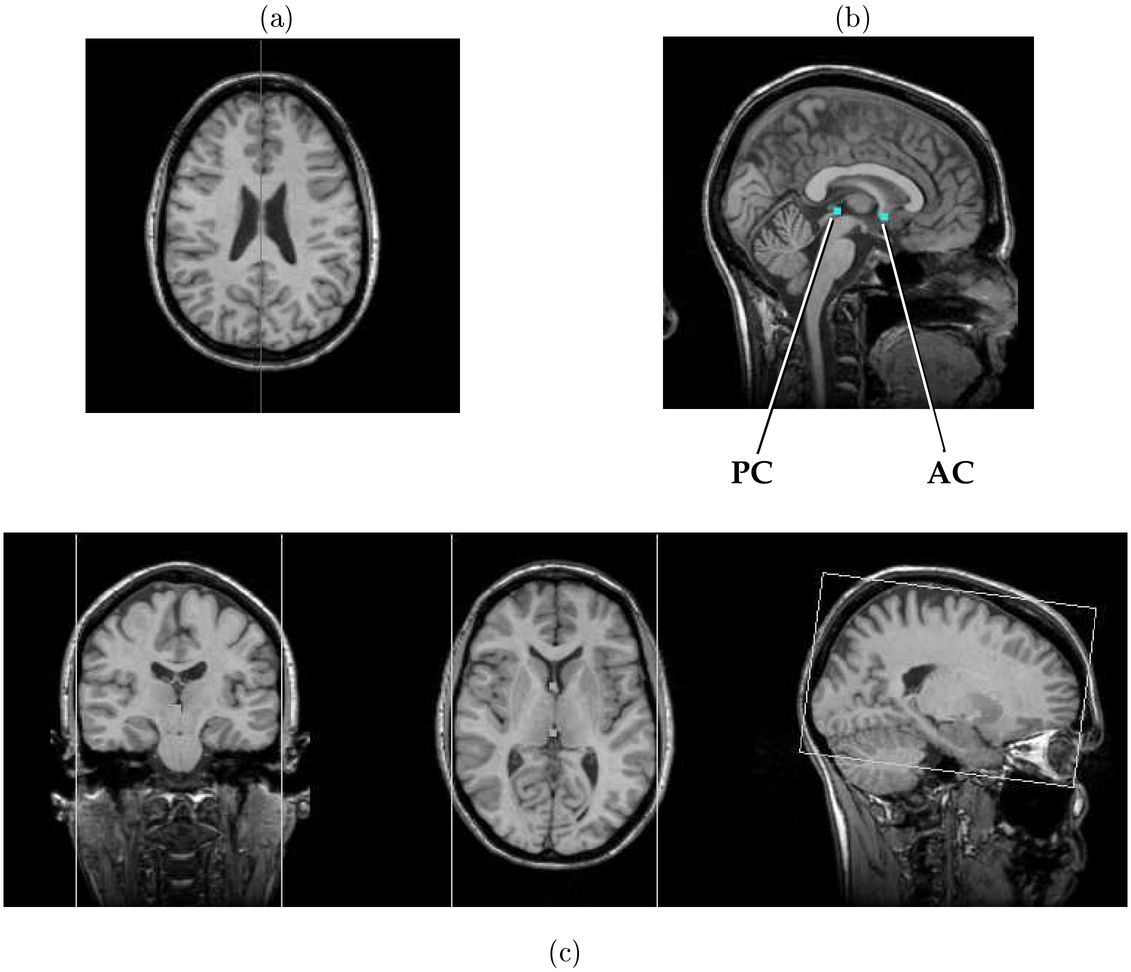
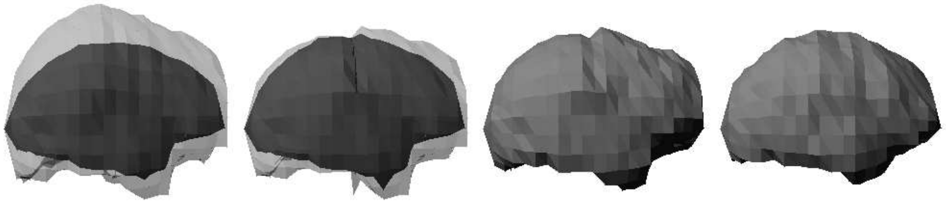


Figure 4.4: The landmarks for the Talairach coordinate system. (a) The mid-sagittal plane. (b) The anterior and posterior commissure points. (c) The bounding box.



(a) (b) (c) (d)

Figure 4.5: Propagation of shape constraints for a radial surface brain model. (a) The inner (dark) and outer (light) uncertainty bounds after a single radial has been entered and propagated. (b) The intervals after an edge detector has searched the central slice and the results have been propagated. (c) The final surface found by the search-and-propagate loop. (d) The surface after it has been corrected by the user.

Once these landmarks have been specified, they are used to initialize the radial shape model. The AC-PC line provides the model's reference axis. The back-to-front dimension of the bounding box is used to compute both the endpoints of the reference axis and the proper spacing between model slices. In the 2D case, it was noted that the reference axis provided two radial lengths for the model. This is not the case with the RSM. Therefore, the user typically enters one or more additional "hint" radials by hand to generate initial uncertainty bounds for the instantiated model.

4.3.2 A modified search-and-propagate loop

At this point, the automated search-and-propagate loop begins. Figure 4.5 demonstrates the process. In this example, the user has drawn a single extra radial on the central model slice. Figure 4.5a shows the uncertainty bounds after constraint propagation. Note that these bounding surfaces already capture the approximate shape of the brain.

The system's search-and-propagate strategy is modified in two ways for 3D processing. First, the system limits its search to the slice in which the user provided hint radials. (In this case, the central slice.) The rationale is that the uncertainty bounds will provide a better guide in the vicinity of the hint radials. Second, the system delays constraint propagation until the best edge has been selected for every radial in the slice. By committing to several radials at once, the system may be able to avoid the problem where a single poor edge choice skews subsequent model fitting.

Figure 4.5b shows the results after searching the central slice for edges and propagating the best ones. The bounds are much closer together than before because the extra radials and their relations to neighboring radials have been used to tighten the uncertainty intervals.

The edge detector is then applied to the slices adjacent to the middle one. This cycle of searching a slice and propagating new edges moves from the center slice outward, until all of the radials in the surface have been searched. For any radial with remaining uncertainty, a location is guessed by taking the midpoint of its uncertainty interval. (Figure 4.5c.) At this point, the user can manually correct any errors made by the system. (Figure 4.5d.)

4.4 Results

The radial surface model was tested on MR volume data from four different individuals. First, the cortical surface for each dataset was segmented by hand, using Scanner's surface editor to draw surfaces with 20 slices and 30 radials per slice. These surfaces were used for a series of leave-one-out experiments. Four brain models were constructed, with each one using a different set of three hand-drawn surfaces as training data. Each model was then used to segment the fourth volume dataset, and the result was compared to the hand-drawn version. To simulate user initialization of the segmentation process, the Talairach landmarks for each dataset were taken from the corresponding hand-drawn surface.

Scanner is an interactive system, but it should still do as much automatic segmentation as possible. Thus the best segmentations are those that achieve accuracy with the least amount of intervention from the user. Therefore, as in the 2D case, an extracted surface's quality can be gauged by counting the total number of radials that the user specified to define it. This measure has two components: the number of radials given as initialization, plus any corrections that must be made to bring the surface into agreement with the MR images. For testing purposes, it was assumed that a user would correct any radial in the segmented surface which differed from the hand-drawn surface by more than 4 pixels.

Experiments were run using the modified search-and-propagate strategy described in section 4.3. The model was initialized with a single radial from the hand-drawn surface, constraints were propagated to shrink the uncertainty bounds, then alternating phases of edge detection and constraint propagation were run until all radials had been searched.

Of the 600 radials in the segmented surface, the simulations showed that the user would have needed to correct an average of 196.25. Since defining this surface manually would require drawing all 600 radials, the shape constraints reduced the user's workload by two-thirds. This is the same degree of improvement that was seen with the 2D model, even though the 3D experiments started with considerably less information—a 3D reference axis and a single radial, as opposed to two radials per slice in the 2D case. This suggests that even better 3D results could be obtained if the user is willing to provide additional hint radials at the beginning of the segmentation process. Furthermore, using the 3D model requires less bookkeeping, since a single model can be used to segment the entire structure. Juggling the different cross-sectional models for the 2D method would get tedious very quickly.

4.5 Conclusions

This chapter demonstrates how a 2D shape model can be extended to handle a full 3D surface. This generalization makes it possible to represent a complete object with a single model, rather than using a collection of 2D models for differently shaped cross-sections. The results show that by using this radial shape model, users can significantly reduce their workload when segmenting image volumes.

Clearly the amount of detail that can be achieved with a radial shape model is a concern. The tests reported here acquire only a low-resolution model of the brain. More detail could be captured by increasing the density of radials, but for a convoluted object like the cortical surface, the RSM alone is insufficient to generate a high-resolution reconstruction. However, the results of the RSM *can* provide an important starting point for other segmentation methods. The next chapter addresses this issue, showing how the RSM can be incorporated into a working system for reconstructing and mapping the human brain.

Chapter 5

APPLICATION: THE BRAIN MAPPER

An ongoing research project provided an excellent opportunity to apply the radial surface model to a real-world problem. As part of the national Human Brain Project initiative, the Structural Informatics Group at the University of Washington is developing an information system for storing, organizing, and sharing functional brain map data[12]. A key component of the project was a system which used 3D region growing to reconstruct the surface of the brain from an MRI volume[48]. But due to complex control parameters and the lack of a unified interface, the system was difficult for novice users to learn.

Many researchers have studied the problem of segmenting the cortical surface[15, 21, 27, 35, 42, 56]. However, most of the reported techniques assume that the MRI images have been "scalped"—that the skull, skin, and other structures outside of the cortex have been removed. Unfortunately, nobody has discovered a reliable method for scalping an MRI volume automatically. Volumes must therefore be scalped by hand before these other segmentation methods can be applied. In a system intended for everyday use, this problem must be addressed. The radial shape model offers an excellent method for performing this very task.

This chapter describes the Brain Mapper, an interactive system for registering functional brain map data with a 3D reconstruction of the cerebral cortex. The system utilizes a novel two-stage method for brain segmentation, in which a radial surface model is used to scalp volume data, at which point an isosurfacing technique generates a more detailed cortical surface. By making user interaction an integral part of the reconstruction process, the Brain Mapper avoids the pitfalls of fully automated segmentation systems. The system has proven to be sufficiently fast and robust, enabling the reconstruction and mapping of over forty patients to date.

5.1 Cortical Stimulation Mapping

The functional maps used in this project are gathered via *cortical stimulation mapping (CSM)* during neurosurgery for intractable epilepsy. With certain forms of epilepsy, the frequency of seizures can be reduced by removing (or *resecting*) a portion of the temporal lobe. This region of the brain is also known to control language, although the specific arrangement of language processing sites varies from person to person. Therefore, cortical stimulation mapping is used during the operation to locate the patient's essential language sites and plan the temporal lobe resection.

The mapping proceeds as follows. After a portion of the skull has been removed to expose the cortical surface, the patient is awakened (but kept under local anesthesia). The patient is then asked to perform a simple, language-related task. An example task would be object naming, where the patient is shown pictures of common objects—a chair or a dog, for example—and asked to name them. For each repetition of the task, a different site on the surface of the brain is stimulated with a mild electrical current. Sites where the stimulation disrupts the task are deemed essential for language function, and therefore must be avoided during resection to prevent permanent language impairment.

Although these language maps are gathered primarily to guide surgery on individual patients, they also have research implications. For example, correlations have been shown between the patterns of language sites and behavioral indicators such as verbal IQ[51]. Such results raise intriguing questions about how variations in cortical anatomy may affect function. Avenues for studying a possible connection abound, especially with the growing availability of functional MRI (fMRI), positron emission tomography (PET), and other non-invasive methods for gathering functional data.

The UW Brain Project is building an information framework to support this type of research. Among other things, this system must be able to record stimulation maps for individual patients and support map comparisons between groups of patients. Initially, the maps are recorded with a photograph taken during the patient's surgery. A sample photograph is shown in figure 5.1. Sterile numbered tags have been placed on the cortical surface to mark stimulation sites. To facilitate further study, the 3D locations of these sites

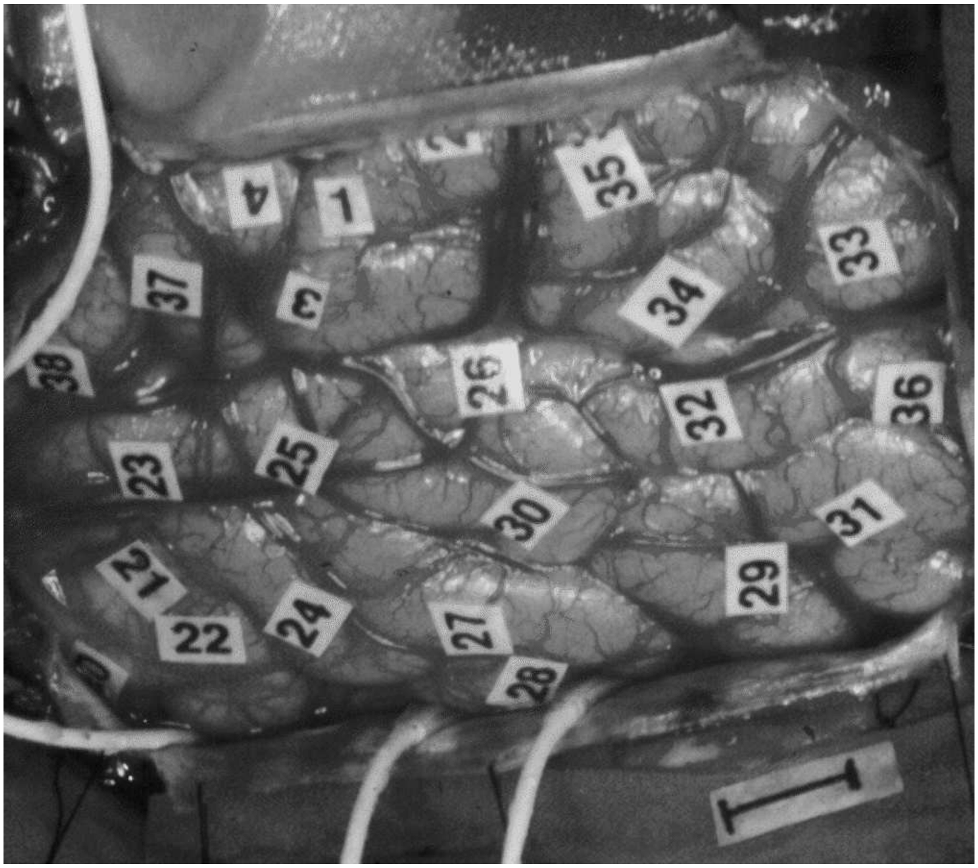


Figure 5.1: Intra-operative photograph taken during cortical stimulation mapping. Sterile number tags mark positions where stimulation tests were performed. The scale bar in the lower right-hand corner is 1 cm long.

need to be recovered.

The Brain Mapper is an interactive tool for performing this task. A visualization-based technique is used, in which the user places virtual number tags onto a 3D brain reconstruction to match the intra-operative photograph. As the photo demonstrates, the veins and arteries provide the most prominent landmarks in the photo. To make the visual mapping task easier, the Brain Mapper reconstructs not just the brain surface, but veins and arteries as well.

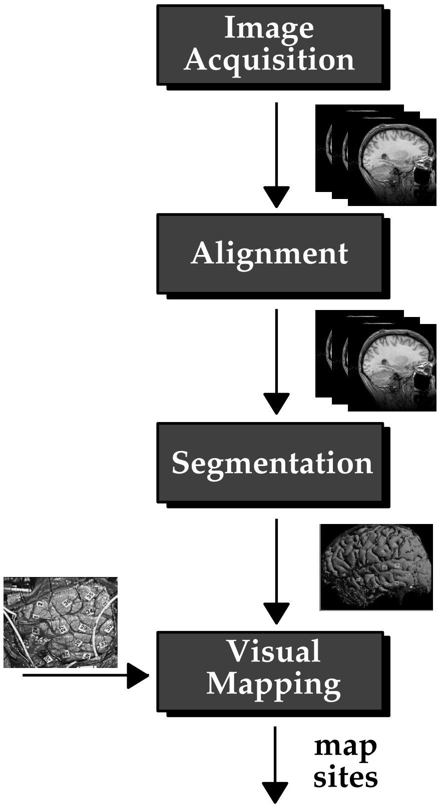
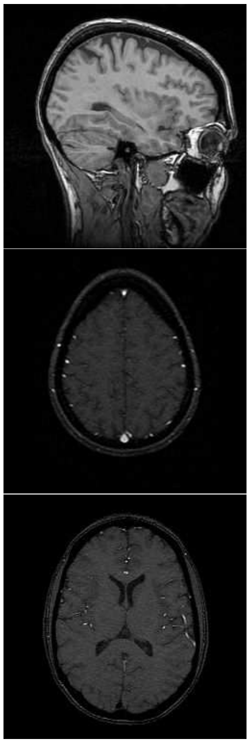


Figure 5.2: The mapping pipeline. After three sets of MR images are acquired and aligned, they are segmented to reconstruct the cortical surface and its superficial vessels. Stimulation site coordinates are recovered by interactively matching the surface to an intra-operative photo.

5.2 The Mapping Pipeline

The steps in the visualization-based mapping procedure are illustrated in figure 5.2. Three sets of MR images of the patient's head are acquired; one is optimized for cortical anatomy, the second for veins, and the third for arteries. After these images have been aligned to one another, the cortical surface and its superficial blood vessels are reconstructed. Next, using the intra-operative photo for reference, the user marks the positions of stimulation sites on the 3D reconstruction. The final result is a set of coordinates for the stimulation sites that are in registration with the MR volume. This section details the stages of the mapping procedure.



Surface anatomy: T1-weighted SPGR (29/5/1/45°) [TR/TE/NEX/flip angle], 22 cm field-of-view (FOV), 256 × 192, 1.2 mm sagittal slices.

Veins: 2D time-of-flight venogram (45/9/1/60°), 22 cm FOV, 256 × 192, 1.5 mm axial slices.

Arteries: 3D MOTSA MR angiogram, 4 overlapping slabs of 16 partitions each (36/6.9/1/25°), flow compensation, 22 cm FOV, 256 × 256, 0.9 mm axial slices.

Figure 5.3: The three MRI datasets used for reconstruction.

5.2.1 Image acquisition

During the week prior to a patient's surgery, three image series are acquired using a whole body 1.5 Tesla MR scanner: one for surface anatomy, one for veins, and one for arteries. Figure 5.3 shows samples of each type of scan, along with details about the imaging parameters used. After acquisition, these three datasets are transferred from the radiology department to a local database.

5.2.2 Alignment

The three sets of MR images have differing orientations and resolutions, so they must be aligned before further processing can take place. Using information that is stored in the image headers about the MR scanner's coordinate space, a single bounding box containing all three series is computed. Each series is then resampled to get a 256^3 volume with uniform-sized voxels, yielding three aligned datasets in which a given voxel will correspond to the same location in machine coordinate space.

This automatic approach to registration will work in cases where patient motion between MR scans is negligible. However, in some cases this is an unreasonable assumption. When patient motion cannot be ignored, the operator can perform a manual alignment of the datasets, using the Register tool developed at the Montreal Neurological Institute[41]. This tool lets the user place landmarks in the datasets by hand in order to specify correspondence points. The tool then performs an affine transformation in order to bring the datasets into alignment.

5.2.3 Segmenting the cortex

Figure 5.4 details the process for segmenting the cortex. First, a radial shape model is used to define an envelope around the brain, separating it from the skull and other surrounding structures. This envelope is then used to mask the volume data, so that a higher resolution surface model can be extracted.

Model fitting

In this stage, the system fits a radial surface model of the brain to a particular patient's MRI data. This is the most time-consuming step in the mapping process, since it requires a significant amount of user interaction. However, the interaction tends to be straightforward, allowing the user to directly manipulate objects of interest rather than tweaking unintuitive parameters.

As described in section 4.3, the user first defines the landmarks for the Talairach coordinate system—the mid-sagittal plane, the AC-PC line, and a bounding box around the brain. These landmarks are used to instantiate a radial surface model. The user may also provide "hints" to the system by drawing one or more radials by hand. In practice, specifying a few radials on one of the central slices of the model, especially around the temporal lobes, leads to much better results. The user then clicks a "Start Segmentation" button, invoking Scanner's automated search-and-propagate loop. When this finishes, the user may inspect the model slices and make corrections as needed.

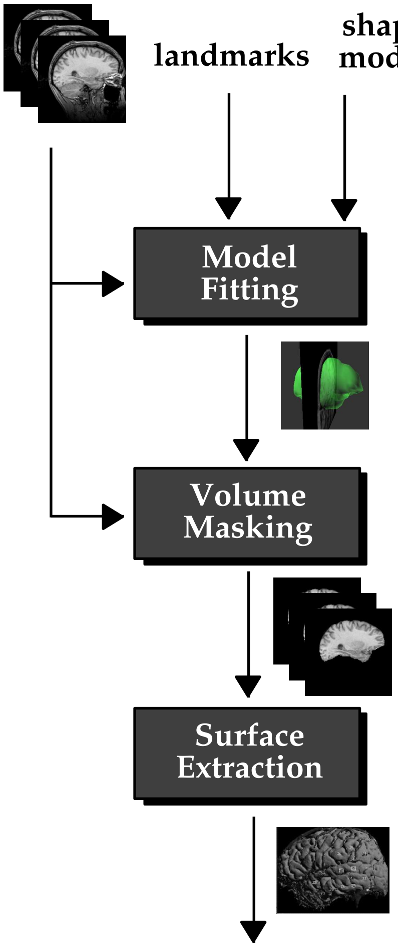


Figure 5.4: Details of the segmentation stage. Based on user-specified landmarks, the system fits a radial surface model to a patient's MRI data. The fitted model is used as a mask to remove structures outside the cortex, enabling a detailed isosurface to be extracted from the remaining data.

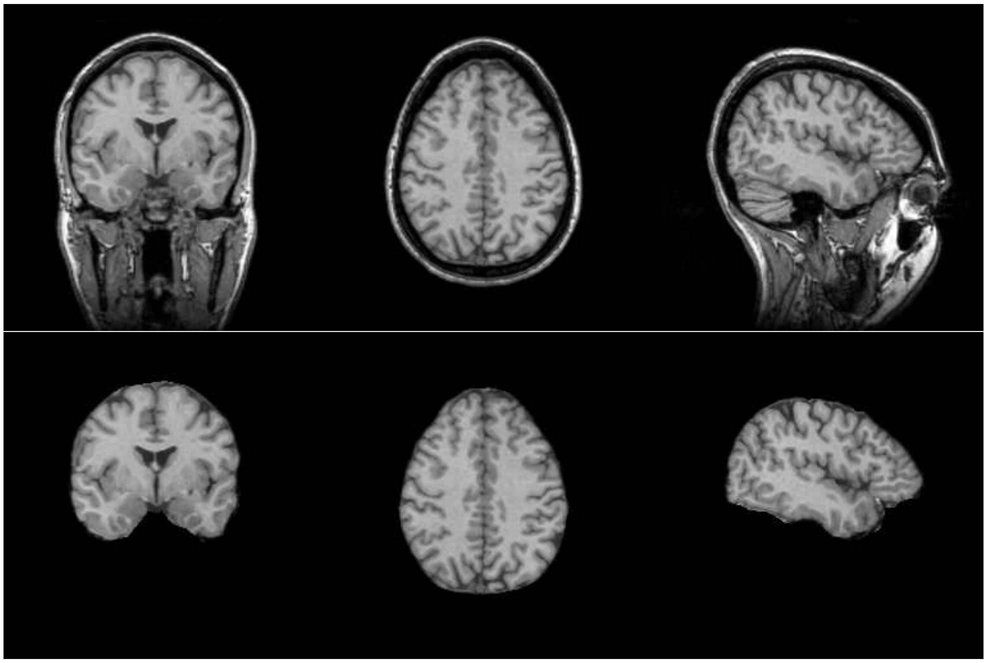


Figure 5.5: Three orthogonal MRI slices, before and after masking.

Creating the voxel mask

Once an initial surface has been found, it is used as a mask to exclude regions outside the brain's surface. The radial surface model is boundary-based, so it must be converted into a region-based voxel mask. This is accomplished in two steps. First, a shell is constructed by locating all voxels within a certain distance, ε, of at least one facet of the model. Using $\varepsilon = 0.5$ produces a strict voxel rasterization of the radial surface; increasing the tolerance gives the shell thickness and rounds out its corners. Second, a flood fill algorithm is used to add all voxels inside the shell to the mask. Since the model is closed by construction, this filling operation will not spill into the region outside the shell. Figure 5.5 shows the results of volume masking.

The goal of this strategy is to avoid doing an expensive point-in-polyhedron test for every voxel in the dataset, because that approach would require comparing each voxel to every facet in the initial surface. By including any voxel that is within some limited distance of the surface, all further voxel/facet checks for a particular voxel can be bypassed as soon as one sufficiently close facet is found.

The MRI datasets contain millions of voxels, so the amount of per-voxel computation must be kept to a minimum. Therefore the mask generation stage has been optimized in two ways. First, each voxel is checked against the bounding box of the radial surface; if it is not within ε of that box, then it can be discarded immediately. This simple check rules out approximately two-thirds of the voxels. Second, before computing the distance between a voxel and a facet, the distance between the voxel and the facet's plane is checked. The voxel-to-plane distance, d_p, is simple to compute, and we know that it cannot exceed the voxel-to-facet distance, d_f. If it turns out that $d_p > \varepsilon$, then $d_f > \varepsilon$, so the more complicated computation of d_f can be avoided. This second check eliminates roughly two-thirds of the voxel/facet pairs that survive the bounding box test. Combined, these two optimizations reduce the amount of computation required to generate the voxel mask by nearly an order of magnitude.

Extracting the surface

After the voxel mask has been computed, the final surface can be extracted. First a new volume dataset is created by setting the intensity of all voxels outside the masked region to zero. This masked dataset is then passed to an isosurface extraction algorithm to convert its voxel-based representation of the brain into a surface-based one. The technique is based on the marching cubes algorithm of Lorensen and Cline[40], using the method described by Bloomenthal to resolve topological ambiguities[10].

With the exception of the initial model fitting, the stages of segmentation run very quickly. Thus it is easy for the user to adjust the segmentation results interactively. By looking at the masked volume or the extracted isosurface, it is fairly easy to locate errors in the fitted model. Fixing such errors is straightforward—the user simply adjusts radials in the affected region, remasks the volume, and extracts a new isosurface.

5.2.4 Segmenting veins and arteries

The process for reconstructing the veins and arteries is nearly identical to that for the cortical surface. The fitted shape model obtained from segmenting the cortex provides an obvious starting point. One minor complication is that the vessels of interest lie just above the cortical surface, and therefore the larger ones may lie partially outside the fitted model.

To address this problem, a companion model is used for the veins and arteries. This model is initialized by copying the landmarks and radial lengths from the first fitted model, avoiding the overhead of a second round of constraint propagation and edge detection. The copied model is then “inflated” slightly by lengthening all of its radials by a fixed percentage. Since this inflation may not have the desired effect everywhere, the user can edit the result to make sure that surface vessels are included, but that radials do not extend so far as to include scalp vessels.

The rest of the segmentation then proceeds as it did with the cortical surface. The model is used to remove structures outside the brain, and isosurface extraction is used to reconstruct the veins and arteries from their respective datasets. The user can experiment with different thresholds until a satisfactory level is found, and the resulting surfaces are saved to the database. Figure 5.6 shows the final reconstruction with veins and arteries.

5.2.5 Visual mapping

In the final stage of the mapping process, the user marks the locations of the stimulation sites on the high-resolution reconstruction of the brain. The reconstruction is displayed alongside a photograph taken during the patient’s surgery. (See figure 5.7.) The user’s goal is to reproduce the locations of the numbered tags on the 3D reconstruction, so that their 3D coordinates can be recovered. The reconstructed blood vessels provide key landmarks.

In the mapping interface, the user selects numbered tags from a number palette and drags them onto the 3D reconstruction. When a tag is dropped, a 3D pick operation finds the closest point on the cortex under the cursor. Tags already on the reconstruction can be reselected and moved at any time. Sites which were deemed to be essential for language can be annotated as such through a menu option. In figure 5.7, these critical language sites

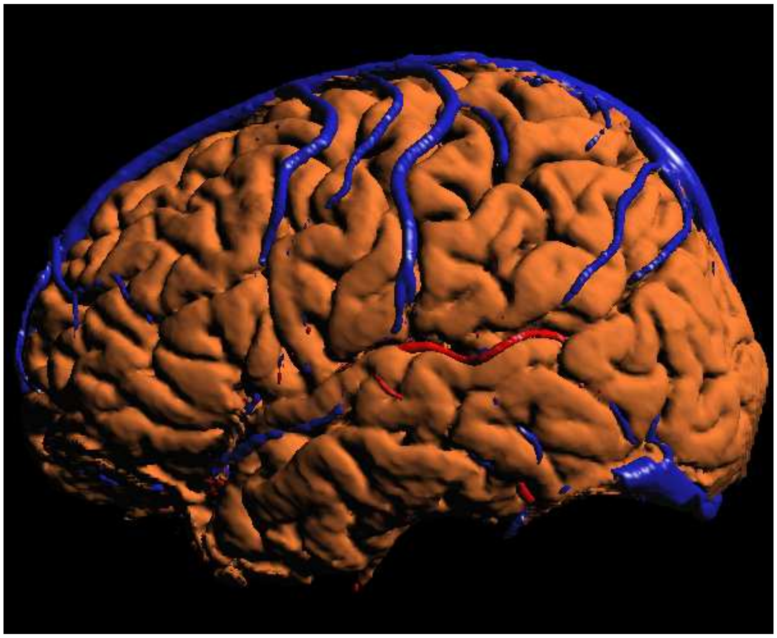
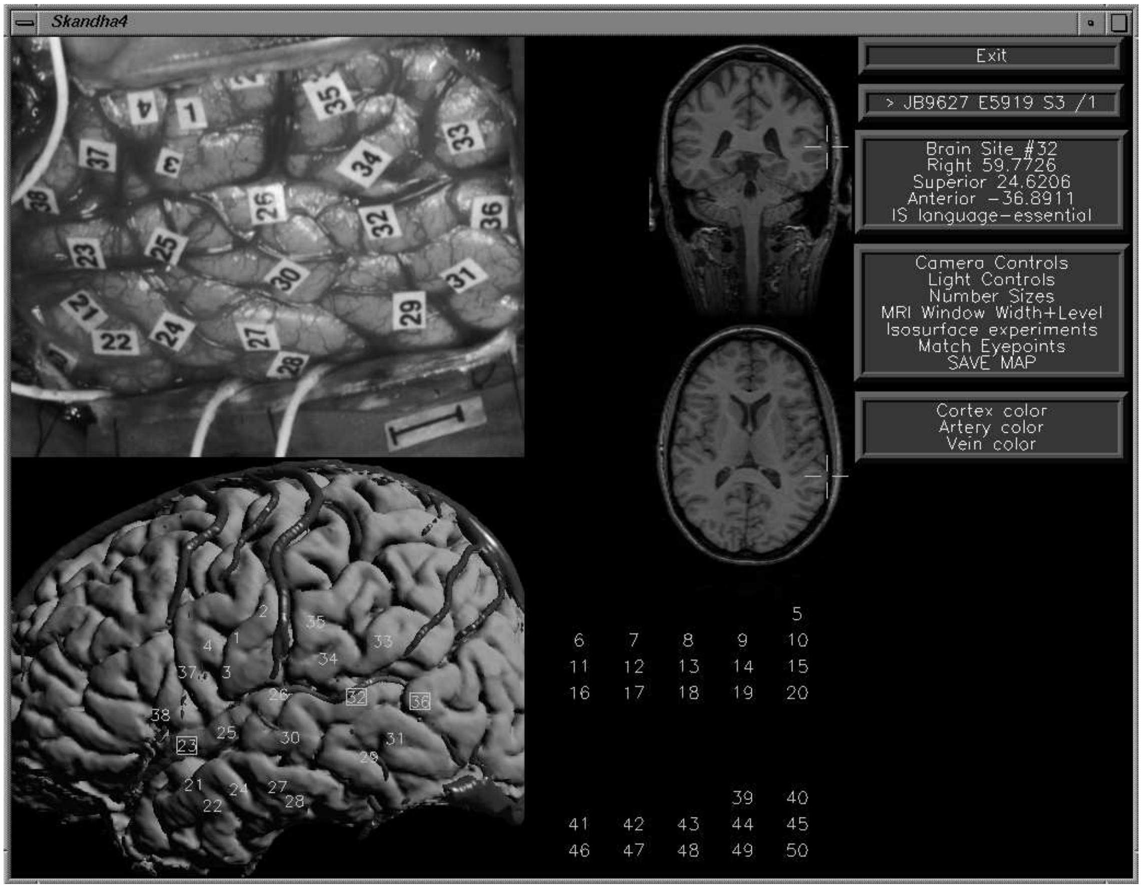


Figure 5.6: Reconstructed cortex with veins (blue) and arteries (red).

have a box drawn around them.

During visual mapping, the user has complete control over how the reconstruction is displayed. In particular, the camera parameters can be adjusted to view the 3D brain from different angles, or zoom in or out as needed. The site tags are actual 3D objects, so they will automatically maintain their proper positions with respect to the cortical surface. This ability to adjust the view is useful for situations in which the intra-operative photograph is taken from an unusual angle. Users of the mapping program have found that when the settings of the virtual camera nearly match those of the real camera, it becomes much easier to map stimulation sites.

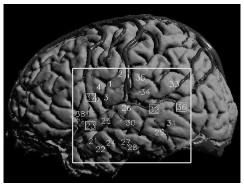
Figure 5.8 shows a completed mapping. The camera parameters have been adjusted to match those of the photograph as closely as possible.



> JB9627 E5919 S3 /1

Brain Site #32
Right 59.7726
Superior 24.6206
Anterior −36.8911
IS language−essential

Figure 5.7: The visual mapping interface. The reconstructed surface and vessels are shown below a photograph taken during surgery. The user drags tags from a number palette and drops them onto their correct positions on the 3D surface. After a tag has been placed, the system can also display its position on the original MRI slices (as shown in this example for site 32). Tags with boxes around them indicate language-critical sites. (Software by Jeff Prothero.)



Area of detail

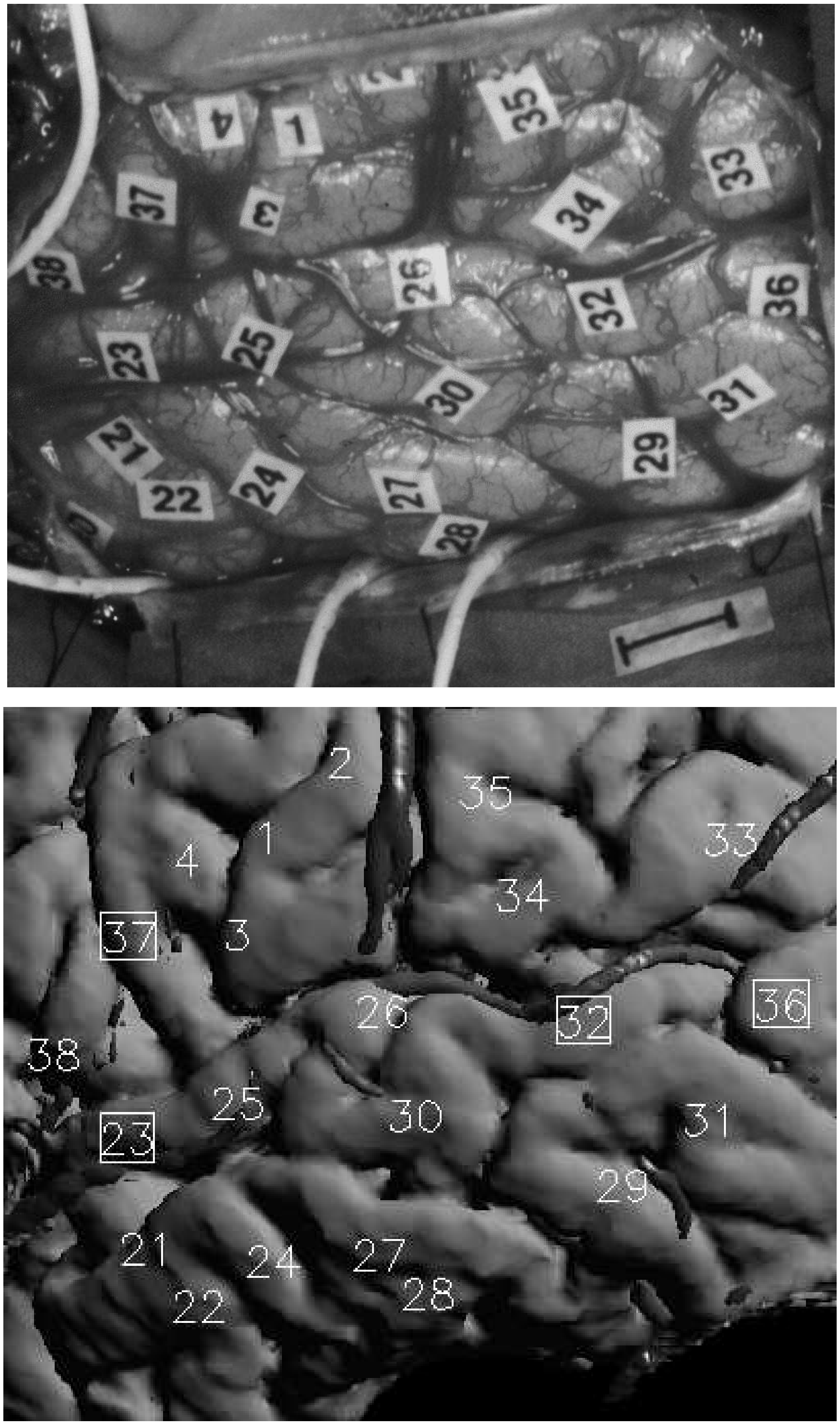


Figure 5.8: A completed brain map.

5.3 Software Architecture

Initially, the Structural Informatics Group used an assortment of software tools to perform the various parts of this mapping pipeline, but getting them all working successfully required considerable expertise. Different pieces were handled by different tools and scripts, making it difficult for the uninitiated to learn to use the system. The segmentation tools in particular required a solid understanding of rather unintuitive parameters to control a 3D region grower.

From the outset, the Brain Mapper had been intended to be usable by a technician, someone who might not have extensive experience using computer-based tools. Therefore it was decided that the Brain Mapper needed a more uniform interface, something that would tie together all the stages of the mapping task. As the earlier description demonstrates, the brain mapping process lends itself to a workflow description; the larger problem can be partitioned into smaller ones, each of which takes certain inputs, manipulates them in some way, and generates outputs for the next stage. Consequently a workflow model[2] was adopted to provide the high-level architecture for the integrated system.

5.3.1 Primitives

The workflow architecture includes three fundamental building blocks:

- a *workflow*, which groups related worksteps and datapaths into a task sequence
- a *workstep*, which performs a particular task, manipulating a set of inputs to generate a set of outputs
- a *datapath*, which transfers data between worksteps

Figure 5.9 shows an example assembly with three worksteps (the rounded rectangles), several datapaths (the arrows), and one workflow (the shaded rectangle). Skandha4 supports an object-oriented programming style, so each of these primitives is defined as an object class.

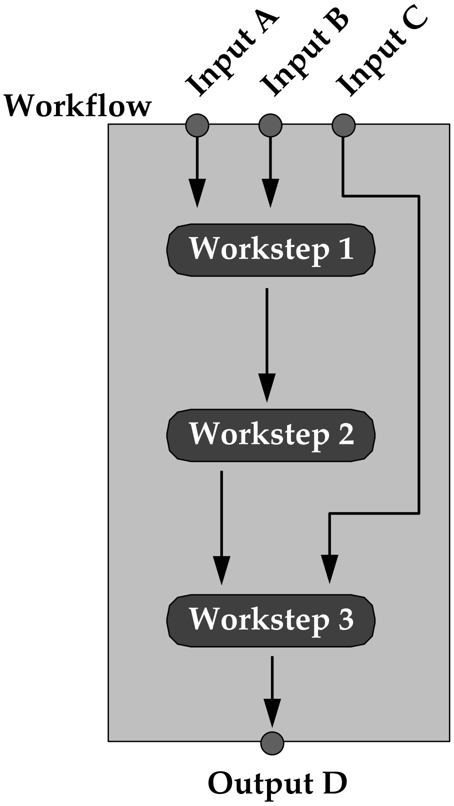


Figure 5.9: A simple workflow, consisting of three worksteps, six datapaths, and one workflow object.

Workflows

A workflow object represents a task sequence. It stores a list of workstep objects, each of which performs one task in the sequence. For example, the following Skandha4 code fragment defines the top-level workflow for the Brain Mapper:

```
(setq mapper-workflow
 (send class-workflow :new
       :name  "BRAIN MAPPER"
       :steps (list select-step
                    align-step
                    segment-cortex-step
                    segment-veins-step
                    segment-arteries-step
                    mapping-step)))
```

Two hook functions (`:start-hook` and `:end-hook`) can be defined to customize the workflow's behavior when it starts and ends. Each workflow object also maintains lists of input and output datapaths. In addition to defining how the workflow connects to the outside world, these paths are used to distribute inputs to and collect outputs from the worksteps within the workflow. This feature is critical for encapsulation purposes, allowing programmers to treat workflows as black boxes without worrying about their internal behavior.

The workflow concept is not strictly necessary; worksteps and datapaths would be sufficient for describing a task sequence. However, workflows provide a convenient method for treating a group of related tasks as an abstract entity, which is important when trying to construct complex software systems.

Worksteps

A workstep object corresponds to a single task in a workflow. Worksteps are verbs. They perform actions, transforming and combining input data in some specific way. This code fragment creates a workstep for cortex segmentation:

```
(setq segment-cortex-step
  (send class-workstep :new
        :name           "Segment Cortex"
        :init-hook      #'xmap-segment-init
        :start-hook     #'xmap-segment-cortex-start
        :end-hook       #'xmap-segment-cortex-end
        :valid-out-hook (lambda (step)
                          (xmap-validate-step
                           step :hires-surface :anatomy-dataset))))
```

In addition to defining a name, this code fragment associates four hook functions with the workstep.

`init-hook` performs any needed initialization the first time a workstep is invoked.

`start-hook` is called every time a step is started. This function generally provides the bulk of the step's behavior.

`end-hook` is called each time a step finishes and returns the flow of control to its parent workflow. This hook function is often used to save intermediate results to a database.

`valid-out-hook` checks to see if all of the step's outputs are up-to-date. In this example the function is defined on-the-fly, calling a helper function to see if the segmented cortical surface (`:hires-surface`) is available.

Each workstep also maintains lists of inputs and outputs, both in terms of the worksteps that provide (or receive) them and in terms of the datapaths that transfer the data. These lists are updated automatically each time a datapath between two worksteps is defined.

Naturally, some steps may be more involved than others; a high-level step may actually consist of a number of smaller tasks. Therefore a workstep can also have a pointer to a sub-workflow, which consists of a sequence of substeps. By allowing worksteps to have their own workflows, it becomes possible to build deeply nested workflows for very complex tasks.

Datapaths

Datapaths are conveyor belts. They connect worksteps, transferring output from one to be input for another. Datapaths are also used to connect workflow objects with their component worksteps. The following code creates a datapath that transfers the aligned vein volume data from the alignment step to the vein segmentation step:

```
(send class-datapath :new
      :source        align-step
      :sink          segment-veins-step
      :source-name   :aligned-vein-dataset
      :sink-name     :dataset
      :fetch-hook    #'xmap-fetch-aligned-vein-dataset)
```

The variables `:source` and `:sink` store pointers to the source and destination worksteps for a particular piece of data. Worksteps associate names with data, but the source and sink might use different names for the same data. Therefore two additional variables, `:source-name` and `:sink-name` are used to keep track of these names. (By default, `:sink-name` is assumed to be the same as `:source-name`.) Finally, the optional `:fetch-hook` stores a function for retrieving the datapath's data from external storage.

For complicated workflows, it may not be possible or desirable to complete all of the steps in a single session. The various steps may be performed over a matter of hours or even days. When the user restarts a partially completed workflow, the system must be able to find relevant pieces of data that have already been computed. For this reason, a datapath has two mechanisms for retrieving a particular piece of data. First, it checks the data's source workstep to see if the data are ready and waiting. If not, the datapath invokes the fetch hook function to retrieve the data from the database, if available.

What about data?

Since the whole point of workflows is pushing data around, one might wonder why this architecture does not include a data primitive. This decision was primarily influenced by the programming environment. The workflow code is written in Skandha4's Lisp-like scripting language, and in Lisp, *anything* can be data. Functions can be passed as easily as numbers, and new data structures can be constructed on-the-fly. In a sense, then, the system already supports a unified data primitive, and so it was deemed unnecessary to wrap data items within another layer of abstraction.

5.3.2 Building a workflow

The usual process for building a workflow is:

- create the necessary workstep objects
- create a workflow object to encapsulate the worksteps
- create the datapaths to connect the worksteps

Currently, workflows are built by writing Lisp code. A useful extension of the system would be a graphical editor for viewing and manipulating workflows, perhaps similar to the network editor used by the AVS system[1].

The modular nature of the workflow approach should make it easier for future developers to expand the capabilities of the Brain Mapper. For example, the MRI images used in this project sometimes suffer from inhomogeneities, either within a single image or from slice to slice. (Low-frequency intensity variations between adjacent arterial slices can be particularly noticeable.) Suppose it was deemed necessary to correct such inhomogeneities before segmentation. The programmer would need to create a new workstep object—call it the “homogenizer” step—which takes its input from the alignment step and sends its output to the segmentation step. The step would automatically be included in the right place in its parent workflow's menu, and its state would be tracked as well.

5.3.3 Dependency tracking

An important feature of this workflow architecture is automatic tracking of data dependencies between steps. By checking the input and output paths for a workstep, the system can

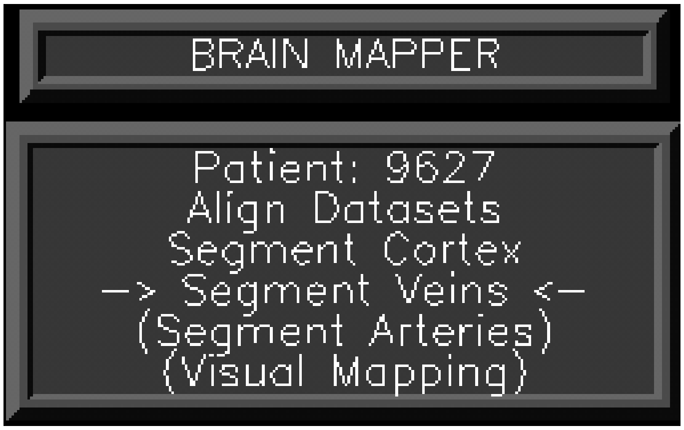


Figure 5.10: The top-level menu for the Brain Mapper, which is generated automatically by the workflow architecture. Completed stages display their label only; pending stages are enclosed with arrows; and blocked stages are enclosed in parentheses.

determine that step's state. If all of its outputs are already available, then the step is in a *finished* state; if some of its outputs are missing, but its inputs are all available, then the step is *pending*; and if some of the step's inputs are missing, then the step is *blocked*. This state information is used to generate menus automatically for workflow objects. Figure 5.10 shows a sample menu for the top level of the Brain Mapper. The top-level workflow object asks each workstep for a label, and each step uses its name along with its state information to generate a properly annotated entry to be included in the workflow's menu. In this example, finished steps show their name only, pending steps are enclosed by arrows (to draw the user's attention, since they are the next to run), and blocked steps are enclosed in parentheses.

5.4 Results

With any system intended for production use, there are three natural measures of performance: correctness, speed, and ease of use.

For the purposes of the brain mapping task, "correctness" applies primarily to the site locations extracted from the final mapping. Specifically, the strategy of visually matching the 3D surface to the photo must produce reliable coordinates. The intra-operative photographs provide a rather unique gold standard—researchers attempting to segment the brain rarely have an actual picture against which to compare. As figure 5.8 demonstrates, correspondences between the photo and reconstruction are clearly visible. In preliminary testing, the previous version of this system showed good repeatability results for a collection of expert and non-expert users[47]. The new shape-based method produces surfaces that look as good as or better than the earlier reconstructions, so it is reasonable to conclude that it will be at least as accurate.

To gauge the speed of the system, measurements were made of the time required to run a patient through the entire mapping procedure. Table 5.1 shows the timing results for the various stages of the mapping pipeline, based on the author's processing of three patients. (The time for image acquisition has been omitted, since it is independent of the rest of the mapping procedure.) Trials were performed on a Silicon Graphics Indy workstation with a 133 MHz R4600 processor and 256 megabytes of memory. On average, it took under two hours to complete a patient's mapping.

Table 5.1: Timings for the mapping pipeline stages (*h:mm*).

	Patient 1	**Patient 2**	**Patient 3**	**Average**
Alignment	0:10	0:09	0:10	0:10
Segment Cortex	1:07	0:53	1:00	1:00
Segment Veins	0:20	0:33	0:14	0:22
Segment Arteries	0:15	0:08	0:08	0:10
Visual Mapping	0:10	0:07	0:06	0:08
Total	2:02	1:50	1:38	1:50

The third performance measure, ease of use, is harder to quantify. One indicator is the type of interaction made possible by the shape-based approach. When segmentation errors occur in the current system, the user can quickly locate them, adjust the fitted model in that region, and extract a new surface. From a user's standpoint, this is a significant improvement over the earlier region growing method, which offered no local control over the segmentation results.

The system's primary user reports that it takes approximately 90 minutes to step through the entire pipeline from raw MRI data to a mapped 3D reconstruction. The majority of this time is devoted to the model fitting stage, which requires a significant amount of user interaction. While the automated fitting process is not perfect, this user estimates that it saves him 10 minutes, compared to the time that would be required to trace the entire RSM by hand. Furthermore, he believes that by having to redraw only selected radials, the quality of his manual corrections is better due to reduced eye and hand fatigue.

But the best evidence that a "useful system" has been produced is that the system actually gets used. To date, the Brain Mapper has been used by four different individuals to reconstruct and map over 40 patients. Furthermore, it is playing a significant role in neuroscientific research. For example, the system provided key support for a rare case study involving a deaf signer undergoing cortical stimulation mapping[19]. In addition, the system is being expanded to support comparisons between CSM results and fMRI[53], and other researchers have expressed interest in using it for their own projects.

5.5 Conclusions

Given the inherent challenges for any segmentation system, the Brain Mapper strikes a good balance between automation and practicality. It provides an integrated set of tools for tackling all of the tasks that must be completed for the problem of language mapping: aligning multiple datasets, reconstructing the relevant structures, and annotating them with stimulation site locations. Furthermore, it employs a reasonably fast and robust segmentation algorithm which addresses the critical problem of "scalping" MRI brain data. In short, the Brain Mapper is a working system, and it is playing a key role in ongoing neuroscientific research.

Most of the work in the Brain Mapper focuses on getting a good reconstruction of the cortex. Yet building good reconstructions is only half the battle. For interactive applications in particular, geometric models are of little use until they have been converted into pictures. Standard methods for rendering such models have been developed, but for the biomedical domain, they often produce unsatisfactory results. The next chapter addresses this issue and investigates ways in which non-standard rendering methods can be used to create better illustrations of anatomy.

Chapter 6

ILLUSTRATION

As the figures from the previous chapter demonstrate, images play prominent roles throughout the Brain Mapper. The center of attention is the 3D brain reconstruction itself—specifically, getting a picture of the reconstruction that looks enough like the photograph that the user can find correspondences between them. Given this task, it is understandable that the system uses standard, photorealistic methods to render the reconstruction.

However, consider the *other* reasons why someone using the system might want to see a rendering of the brain:

- to evaluate the quality of the surface during reconstruction
- to study the stimulation map for a particular patient
- to access detailed information about specific stimulation sites
- to examine different types of language data simultaneously (*e.g.* CSM sites and fMRI)
- to compare maps for multiple patients
- to plan for surgery
- to learn about neuroanatomy and language

Notice that all of these examples are informational in nature. And in each case, the image must convey different information. It therefore comes as no surprise that a single type of image cannot satisfy each goal equally well. Rather, a good informational image must be *designed* with a particular purpose in mind. Who will use it? What information should it convey? What information should it hide? These are the questions that an illustrator must consider during the design process.

Surprisingly, these issues are largely ignored within the vision community. Once a geometric model has been created, vision researchers typically consider their job done. If a picture is needed, the model will be run through the standard graphics pipeline: material properties are added, lights are positioned around it, its geometry is projected to the image plane of a virtual camera, and finally the shaded polygons are rasterized to create a 2D picture. The result will be yet another plastic-looking rendering of yet another 3D model. For biomedical domains in particular, this reliance on standard rendering techniques is often not enough.

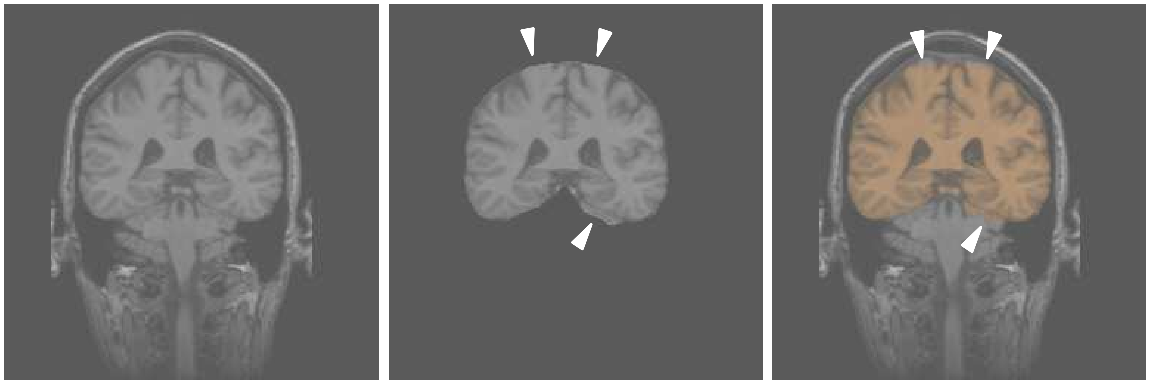
This chapter demonstrates the importance of *illustration*—not just simple conversion of 3D models into 2D pictures, but active design of images which clarify and enlighten. The display needs of three real-world situations are presented to motivate a set of requirements for a medical illustration system. (A solution for the first of these will be presented right away; the other two will be handled later in the chapter.) These requirements are then used to design Pendragon, a prototype of such a system.

6.1 Three Case Studies

6.1.1 Evaluating brain reconstructions

Recall the segmentation strategy from the previous chapter. Based on user landmarks, the system attempts to fit a brain shape model to a particular volume dataset. The user corrects any obvious flaws in the model and masks the volume with it. If the masked results look good, then an isosurface is extracted to get a high-resolution surface model. In practice, this process tends to be one of iterative refinement. Looking at the isosurface often reveals errors in the fitted model, so the user will backtrack to fix the model, remask the volume, and re-extract the isosurface.

In the initial implementation of this strategy, the results of volume masking were presented to the user as MRI slices in which regions outside the mask had been blacked out. The user could then scroll through the cross-sectional views to evaluate the quality of the



(a) (b) (c)

Figure 6.1: Giving visual feedback during model fitting. (a) An MRI slice. (b) The original mask display, in which areas outside the mask were blacked out. (c) The improved display, in which a colored tint is used to highlight the masked region.

mask. With repeated use, though, it became apparent that this display method was not ideal. Although this method does make it easy to find regions where the mask extends out too far, it can be very difficult to find places where it does not extend far enough. (See figure 6.1a and b.) When this happens, parts of the cortex will be trimmed incorrectly. Such errors are generally easy to find on the reconstructed isosurface, because they create a jagged surface artifact. However, it would be better if these errors could be found during the volume masking step itself.

To solve this problem, a better display method was designed. Instead of blacking out regions outside the mask, a colored tint was added to voxels inside the mask. (See figure 6.1c.) The tint is achieved by working in the HLS (hue-lightness-saturation) color space. First, each voxel's RGB color is mapped to its HLS equivalent. By construction, all shades of gray have a saturation of zero and a lightness equal to their original intensity; hue is technically undefined. Pixels inside the masked region can be tinted by giving a hue and non-zero saturation. The color can then be converted back from HLS to RGB and displayed.

The effect is subtle, yet remarkably effective for drawing attention to regions where either too much or too little has been included by the mask. This small improvement can save users time by allowing them to see errors during the volume masking stage, instead of having to backtrack after generating a faulty isosurface.

6.1.2 Studying stimulation maps

Suppose we want to design a new brain illustration, one which is better suited for studying the stimulation map for a particular patient. With this goal, the first question on a viewer's mind will be, "Where are the language-critical sites?"

Two guidelines can be inferred from this question. First, map sites take priority over the brain itself. The brain's geometry is necessary to show where sites are located, of course, but it should be more background than foreground. Second, language-critical sites should be more prominent than non-critical sites.

6.1.3 Displaying coronary arteries

A third test case arose from a request from the National Library of Medicine for illustrations of the coronary arteries. Researchers at NLM are developing a program that uses natural-language processing to identify references to arterial branches in coronary catheterization reports[58]. As part of the program, they have created web-accessible versions of the annotated reports, in which references to arteries in the report text are highlighted. These references are linked to a symbolic diagram which shows the position of that branch relative to the complete arterial tree. However, the researchers also want to provide a *picture* of the highlighted branch, giving viewers a much more concrete understanding of its relation to the rest of the heart. The team at NLM knew that the Structural Informatics Group had 3D models of the heart and coronary arteries, so they asked for our help.

Early on, a model of just the coronary arteries was created for them, using the Virtual Reality Modeling Language (VRML) format. It was immediately clear that such a model was insufficient. As figure 6.2 demonstrates, an image showing the coronary arteries alone is very confusing. The arteries wrap around the surface of the heart, but without showing the heart as well, the picture fails to convey that information.

To improve the illustration, additional models must be added to the scene to provide the necessary context. These contextual structures must be less prominent than the arteries.

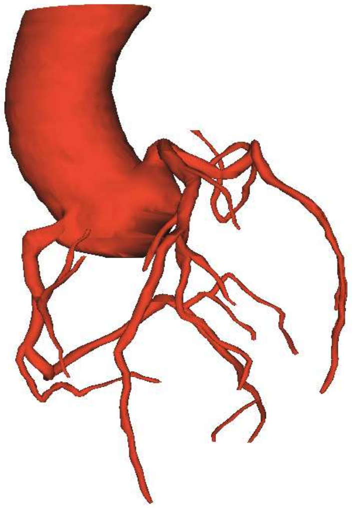


Figure 6.2: Original illustration of the coronary arteries. Without additional context, it is difficult to tell how the individual arteries relate to one another and to the rest of the heart. (Model by David Conley.)

In addition, some mechanism for highlighting individual arteries is needed.

6.2 Requirements for an Anatomy Illustration System

Based on these examples, plus examination of real medical illustrations[50, 67], the following requirements for an anatomical illustration system can be proposed:

Non-standard rendering styles. In the early days of 3D anatomical modeling[60], *any* picture was a significant accomplishment. However, graphics techniques have evolved considerably since then, and the tools should be expanded to reflect these advances. All three case studies could benefit from non-standard rendering techniques. By providing access to a set of simple NPR methods, an illustration system could greatly expand the 3D artist's tool palette.

Information layers. As Edward Tufte observes, "Among the most powerful devices for reducing noise and enriching the content of displays is the technique of layering and separation, visually stratifying various aspects of the data."[65, page 53] Good medical illustrations utilize this principle, displaying several layers of information simultaneously. Layers can be purely graphical (as with the colored tinting example), or they can involve annotations of various types (as with the brain map example). Systems should therefore support the construction of such layers and methods for assembling them.

High-level grouping. In any field, abstraction is a powerful tool. If illustrators can group related structures and work on them as a unit, they will be able to work at higher levels of abstraction. As shown by the coronary artery example, these relationships may have clear anatomical foundations (*e.g.* the left coronary artery and all of its branches), but they may also be purely conceptual (the "context" structures). The artist must be allowed to define the groups that are useful for a particular illustration task.

Anatomical coordinate systems. The standard Cartesian coordinate system, with its arbitrary XYZ axes, inevitably leads to confusion for artists and software developers alike. By necessity, anatomy has a well-established vocabulary for describing locations and directions related to the body. Coordinate information should be presented in terms of this vocabulary.

Interactive lighting control. When the standard graphics pipeline is used to render 3D models, the quality of the output greatly depends on the positioning of the scene's lights. Small changes can make significant differences. With complex anatomical models, this dependence becomes apparent very quickly. When working with traditional 2D media, artists have direct control over the apparent lighting of an object. They must have similar control in a 3D setting.

Knowledge access. In a well-studied domain such as anatomy, there is a wealth of prior knowledge which could be used to assist the illustrator. For example, symbolic knowledge of anatomy has been used to rapidly assemble a 3D scene with a high-level request such as “load the heart and all of its parts.”[14] The main obstacle is getting the knowledge into a usable form.

Layered cutaways. One idiomatic style of medical illustration is the layered cutaway, in which successive layers of a complex object are peeled away to show the underlying structure. An illustration system would therefore benefit from an interactive tool for performing this specific kind of 3D cropping.

Characteristic textures. Another common technique in medical illustration is the use of texture to convey tissue type. For example, muscles are drawn with striations to show the orientation of muscle fibers. While certainly secondary to other requirements in this list, a small library of common textures could do much to increase the visual appeal of illustrations.

Flexibility. The call for flexible tools is a ubiquitous one, but still one worthy of mention. Artists will inevitably use software tools in ways the designers never dreamed of. Consequently, developers should strive to build tools that can be used in multiple ways, but simultaneously stay out of the artist’s way as much as possible.

Dynamic illustrations. Computer-generated illustrations will never rival those drawn by a skilled medical artist. However, computers beat humans hands-down at repetitive tasks. Thus computer-based illustration systems offer enormous potential for building dynamic illustrations—images that can be tailored on-the-fly to fit the needs of the viewer. Tools should therefore give the artist ways to specify the important parts of an illustration—the structures shown, their relative importance, the general appearance of each—in such a way that the image can be re-rendered without losing the underlying “style” of the original.

6.3 A Prototype System: Pendragon

This list of requirements has been used to guide the design of Pendragon, a prototype anatomy illustration system. The main idea behind Pendragon is to let illustrators use and

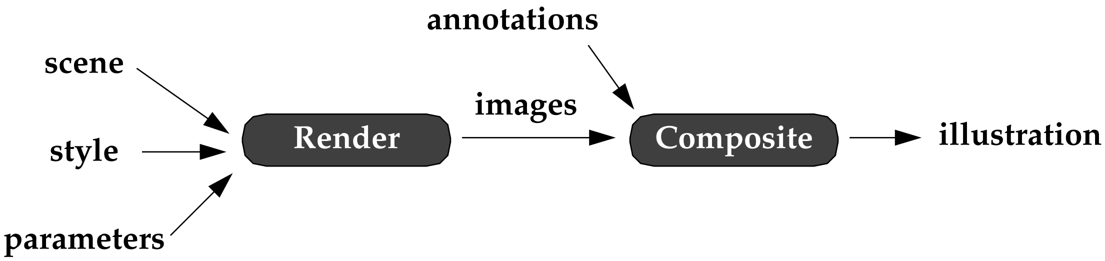


Figure 6.3: Basic architecture for Pendragon. Given a scene, an illustration style and style parameters, one or more images of the scene are generated by the renderer. These images, along with optional annotations, can then be composited to create the final illustration.

define *styles*—high-level descriptions of how basic rendering techniques should be combined to illustrate a 3D scene. By decoupling the style components from the models, it becomes possible to apply styles to dynamically generated scenes. A style that was originally created for the coronary artery example could therefore be transferred to a completely different set of models, so long as those models were partitioned into the same logical groups.

Figure 6.3 shows the basic architecture for Pendragon. The illustration process begins with three inputs: a scene description, a style, and style parameters. The scene description provides a list of geometric models, along with material properties, lights, and a camera. The list of models is also annotated with tags that divide the scene into groups of related structures. For example, the coronary artery illustrations involve three conceptual groups: the objects of interest (the coronary arteries), a highlighted structure (a specific coronary artery), and context (the myocardium, ascending aorta, and pulmonary trunk). The second input, a style, defines which rendering methods should be used for each group of objects. Style parameters allow the illustrator to fine-tune the rendering styles for a particular illustration.

The scene, style, and style parameters are passed to a renderer to generate one or more images. Generally, there will be one image per structure group; however, some complicated styles may generate multiple image layers. These layers, along with optional 2D annotations, are then passed to a compositor to make the final illustration.

6.3.1 Simple styles

Styles are implemented as functions which take a scene description as input and return an image as output. Each style function can also include parameters to control the details of how it performs its rendering. It is important to note that style functions are not restricted to use 3D rendering techniques; 2D methods can also be used as part of a style. The only requirement is that the final result be a 2D image.

This section presents a few sample styles, all of which can be implemented using the standard graphics pipeline. Figure 6.4 demonstrates how each style looks when applied to a scene of two vertebrae (T9 and T10) and the intervertebral disc between them.

Standard rendering

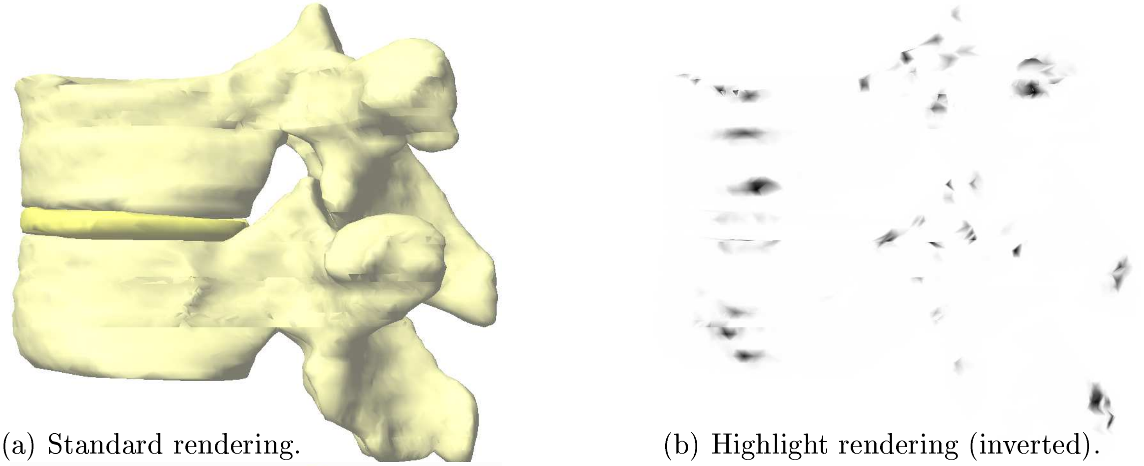
The *standard style* passes the scene description directly through the rendering pipeline, without making any changes to material parameters. Shown in figure 6.4a, this style is the system default.

Highlight rendering

This style draws only an object's specular highlights. This effect is accomplished by setting an object's diffuse reflectance to black, while enabling white specular reflections. Figure 6.4b shows where these highlights would appear for the vertebral example. (The image has been inverted for printing purposes.) This style is of little use by itself, but can add critical shape information to an illustration when used in conjunction with other styles.

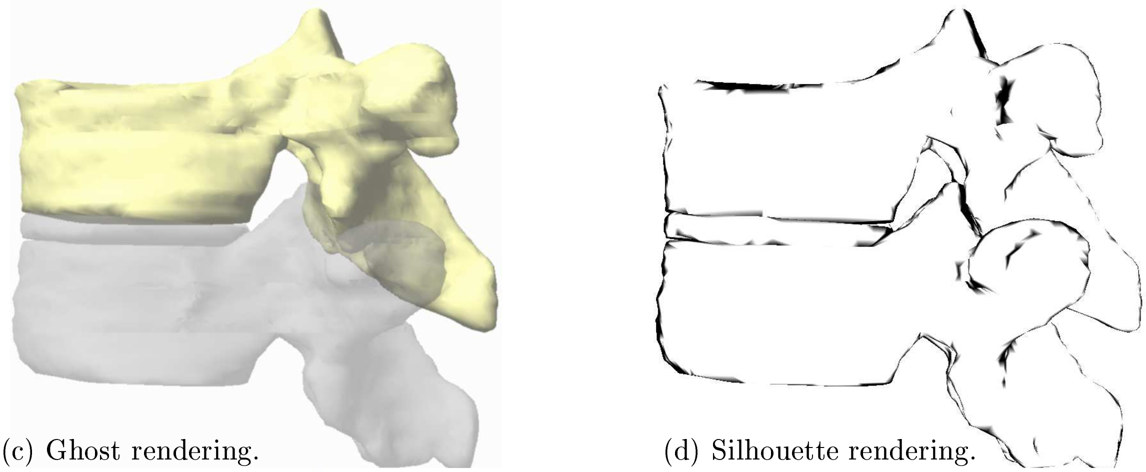
Ghost rendering

An example of a simple style is *ghost rendering*, which makes an object colorless and translucent. This effect is accomplished by setting the object's diffuse color to gray and its opacity to 50%. This ghosting style is useful for objects which are not the central focus of a scene, but provide essential context for other structures, such as the heart in the illustration of the coronary arteries. In figure 6.4c, the lower vertebra and the intervertebral disc are drawn with the ghost style, while the upper vertebra is drawn in a standard style.



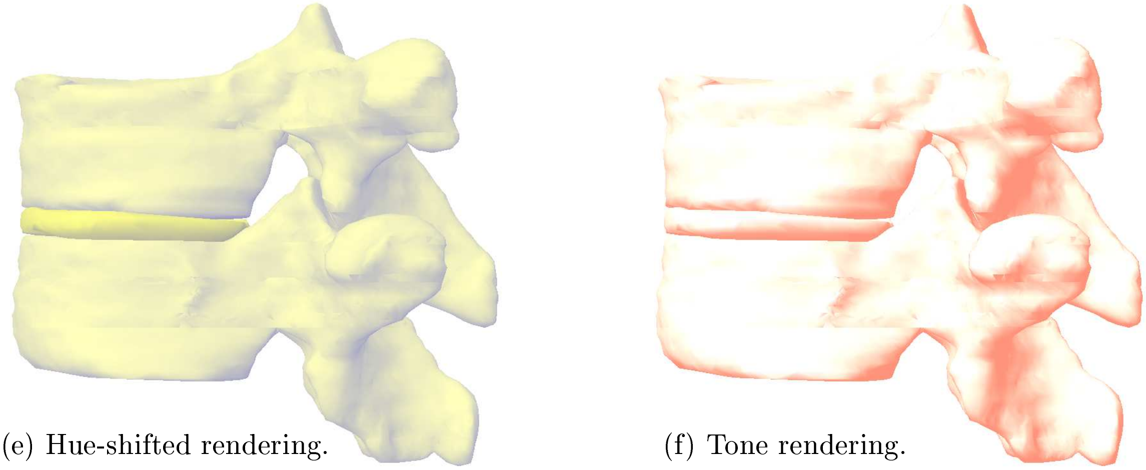
(a) Standard rendering.

(b) Highlight rendering (inverted).



(c) Ghost rendering.

(d) Silhouette rendering.



(e) Hue-shifted rendering.

(f) Tone rendering.

Figure 6.4: Examples of simple styles. (Models by David Conley.)

Silhouette rendering

A more involved style is *silhouette rendering*, which draws just the outline of an object, as seen from a particular viewpoint. (See figure 6.4d.) The most common approach to drawing object outlines is to use Appel's hidden-line algorithm[6] to render only the visible portions of an object's *silhouette edges*. A silhouette edge connects two polygons, one of which faces toward the viewer and the other of which faces away from the viewer. These edges will form the outline of the object when it is projected from three-space onto the camera's 2D viewing plane. Unfortunately, identifying and clipping all of these silhouette edges requires exhaustive search—a time-consuming task for large models.

The approach used by Pendragon approximates this idea, but is designed to take advantage of the standard graphics pipeline. Instead of identifying and drawing only silhouette edges, a view-dependent texture map is applied to the surface. The texture itself is trivial, containing only an object color and an edge color. The color applied to a particular location depends on its surface normal. Regions where the surface normal points toward the viewer will be drawn in the object color; regions where it points away are drawn in the edge color. The key idea is to allow the edge color to bleed over into the barely-visible regions where surface normals are nearly perpendicular to the viewer, as shown in figure 6.5. When the object is rendered, these regions will be seen at a glancing angle, so drawing them in the edge color will generate a thin boundary line for the object. The thickness of this boundary can be changed by adjusting how far the edge color spills over into visible territory.

This method for rendering silhouette objects has advantages and disadvantages. For polygonal models, it can leave gaps in the boundary if the tessellation of the objects is not sufficiently fine. It also generates boundaries of inconsistent width, an effect which can be attractive in some cases but detrimental in others. Most of these shortcomings could be addressed by incorporating ideas from recent work by Markosian *et al.*[44] and Raskar and Cohen[54]. The primary advantage of the approach is that objects can be rendered automatically using the standard graphics pipeline. The view-dependent texture map can be computed very quickly, making this style viable for real-time interaction.

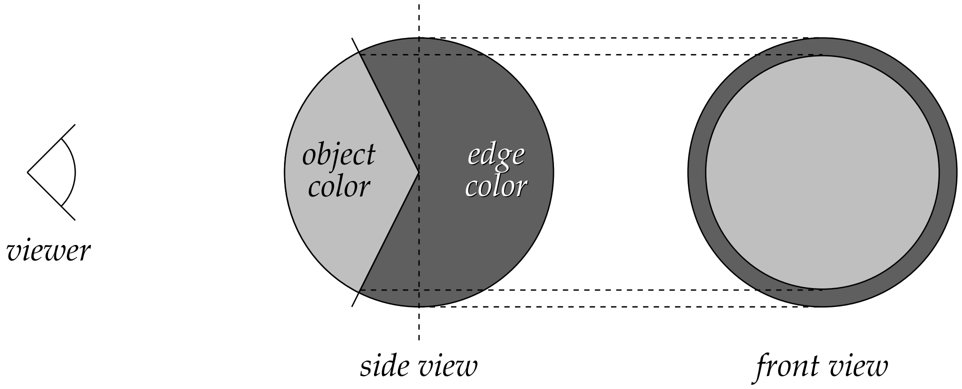


Figure 6.5: The texture map for a silhouette rendering of a sphere. By applying the edge color to regions that are just barely visible to the viewer, the object's silhouette can be drawn.

Hue-shifted rendering

The *hue-shifted rendering* style incorporates a subtle change in an object's hue to convey shape information. Described by Gooch *et al.*[30], this technique imitates one used frequently by technical illustrators. The tonal range in technical illustrations must be limited so that black outlines and white highlights can be seen clearly. This limitation restricts the artist's ability to convey the shape and curvature of an object. To compensate, illustrators often incorporate a "cool-to-warm shift" in their color palette, adding more blue in shaded areas and more yellow in lit areas. Figure 6.4e shows an example; note the improved definition of the models, especially in shadowed regions.

Pendragon implements this cool-to-warm hue shift using the colored light method proposed by Gooch *et al.* For each object in the scene, two opposing colored lights are used. Their colors are computed as follows from the object's diffuse color, $(d_r \; d_g \; d_b)$:

$$L1 = \frac{1}{2} \begin{pmatrix} d_r(\beta - \alpha) + w \\ d_g(\beta - \alpha) + w \\ d_b(\beta - \alpha) - c \end{pmatrix}$$

$$L2 = -L1$$

where:

c controls “cooling” (the intensity of the blue color)

w controls “warming” (the intensity of the yellow color)

α controls the contribution of the object’s color relative to cooling

β controls the contribution of the object’s color relative to warming

Note that these colored lights will have negative-valued components. Although such “negative lights” have no physical analog, they serve to reduce the blue component from lit areas and yellow from shaded areas. Most OpenGL implementations allow negative lights.

Since the light colors depend on an object’s color, each differently colored object requires its own two lights. This poses a problem for complex scenes, because rendering systems typically restrict the number of lights that can be used simultaneously. To compensate, Pendragon renders in multiple passes. On a given pass, a single object will be rendered with its two colored lights. The depth buffer is left unchanged between passes, so that occlusions will be rendered properly.

Tone rendering

The *tone rendering* style is basically a reversal of the standard 3D rendering style, because the lit areas of an object appear white, while color is applied in areas of shadow. Tone rendering provides a rough approximation of how an artist would use ink to shade an object in a hand-drawn sketch.

Tone rendering is implemented by temporarily adjusting the material parameters of an object. By adding a strong emitted light component, an object can be oversaturated. This will make the regions with the strongest illumination turn white, with a gradual fade into the object’s normal color in the regions of strong shadow.

An example of tone rendering is shown in figure 6.4f. To make the effect clear, the light yellow color of the vertebrae was darkened to a reddish-orange color: (0.9 0.25 0.1) in the RGB color space. The scene’s light has an intensity of (1 1 1) for direct lighting and (0.2 0.2 0.2) for ambient lighting.

To get an object's diffuse color to appear in the shadow regions, the tone rendering style sets the object's emission term equal to the diffuse term, sets the ambient term to 50% gray, and lightens the diffuse term. This lightening is accomplished by transforming the RGB color into HLS color space, scaling the lightness value by a user-specified scale factor, and then transforming back into RGB color space. For this example, a scale factor of two was used, resulting in a new diffuse color of (1 1 1).

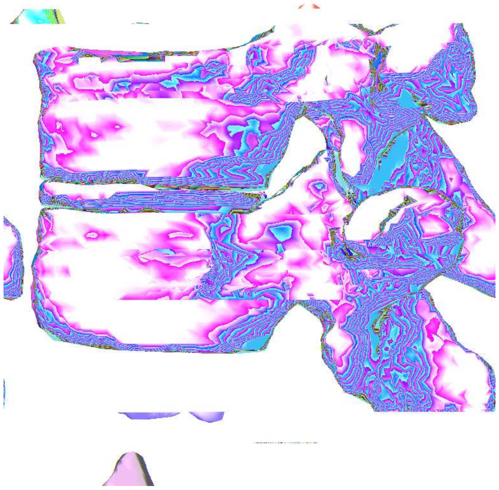
In places without direct light, only emitted light will appear, and thus the object will have its "natural" color. In regions of strong illumination, the RGB components get pushed beyond the legal range of [0,1]; each color component will therefore be clipped to the maximum value of one, turning the object's color white.

6.3.2 Compound styles

While many of these simple styles can generate attractive pictures on their own, far more interesting pictures can be achieved by using the styles in combination. For example, the combination of silhouette and tone rendering is particularly effective. Figure 6.6a shows this combination for the vertebral example. Much like a good sketch, the silhouette provides the object's definition, serving to separate the object from the background. The shading from tone rendering then adds depth and dimension. Figure 6.6b shows the same model, but using a combination of hue-shift, highlight, and silhouette styles. The limited tonal range of the hue-shifted interior makes it possible for the outlines and highlights to be seen.

Pendragon uses image compositing to implement compound styles. The basic style functions are invoked to render the image components. A compositing function is then called, taking two images and a compositing operator as input, and returning the combined image as a result. Repeated composites can be used to combine multiple layers.

The interface for compound styles is the same as for simple styles—provide a scene description, camera and style parameters as input, and get an image as output. Consequently, the difference between simple and compound styles is transparent to the illustrator, and the two style classes can be used interchangeably.



compositing operation to use, and an appropriate alpha setting, a short Script Fu script is invoked to composite the two images. The resulting image can then be returned to the caller, making the compound style function behave just like a simple style function.

6.4 Results

This section shows how the illustration ideas just described can be applied to the case studies presented in sections 6.1.2 and 6.1.3.

6.4.1 Improving the stimulation maps

The goal of this example is helping the viewer to answer the question, "Where are the language-critical sites?"

The current display is poorly suited to answering this question. (See figure 6.7a.) First, the image contains lots of saturated color, drawing particular attention to the veins and arteries. For purposes of the visual matching task, this emphasis makes sense, because the blood vessels provide the key landmarks. However, when the goal is to convey information about language sites, these bold rivers of color are no longer appropriate. Second, the critical sites do not stand out from the non-critical sites. The use of a box around the site number is not sufficient to attract the eye; instead, the viewer must scan the entire image to find the critical sites.

How could the display be improved? First, the anatomy must be de-emphasized considerably. The blood vessels are no longer critical, and could even be omitted entirely. The cortex itself is still needed, but primarily to provide spatial organization for the language sites. The sites themselves can be differentiated with color, using more contrast to emphasize critical sites.

Figure 6.7b shows one possible redesign. The vessels have been downplayed, and a tone rendering style has been assigned to the brain. Using the scene's camera, the 3D stimulation site locations were projected into 2D, then passed to a specialized script in the Gimp, which added number tags as a separate layer over the brain image. Notice the dramatic reduction in contrast without loss of information. It is still clear (perhaps more than before) where

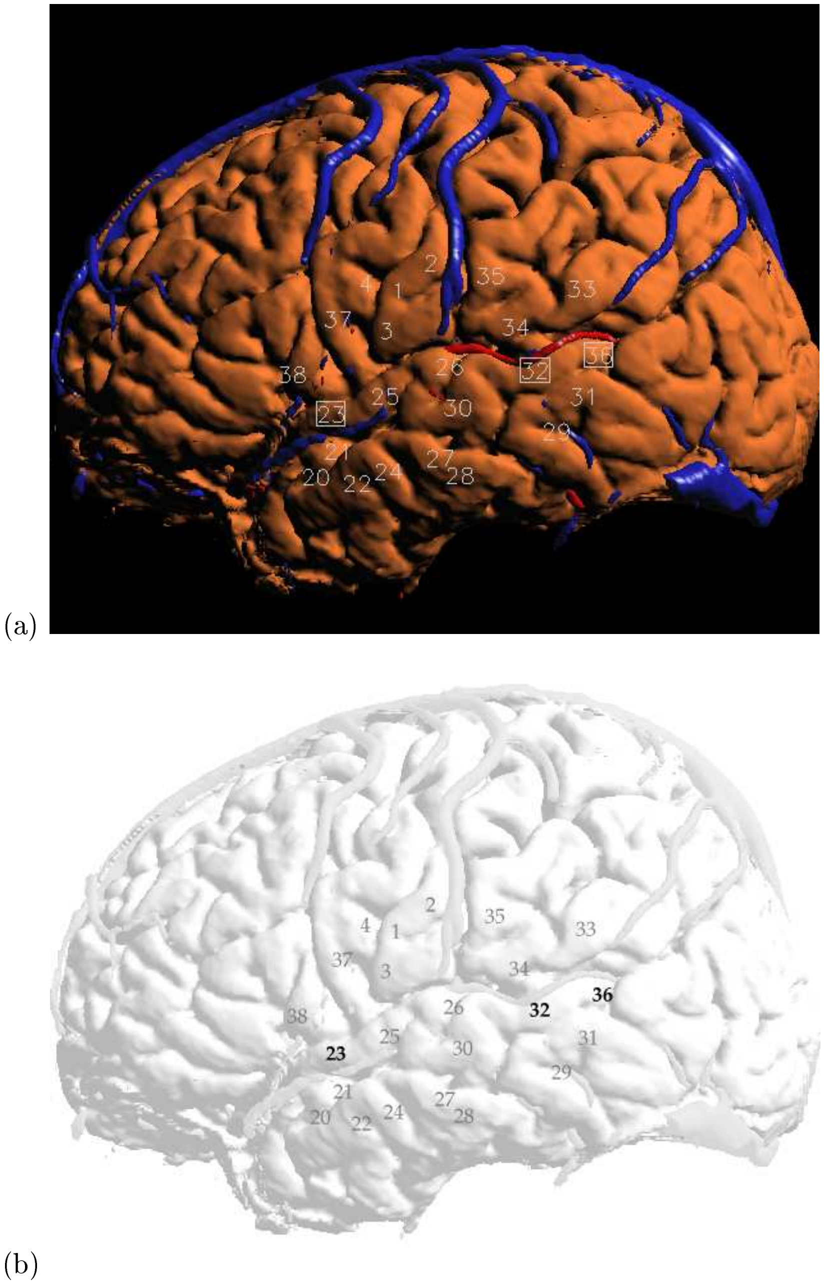


Figure 6.7: Redesigning the brain map display. (a) The original design. (b) A revised version which de-emphasizes the brain and uses contrast differences to draw attention to the language-critical sites.

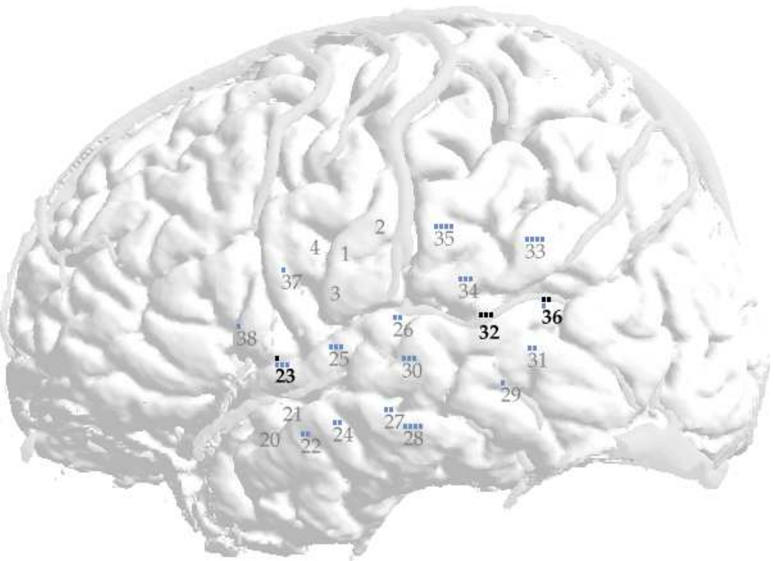


Figure 6.8: An alternate brain map which displays more information per site. Blue boxes denote the number of correct answers, and black boxes denote the number of naming errors.

sites are located and whether or not they are language-critical.

In addition to clarifying the illustration, these changes make space so that the additional information can be added to the image. For example, instead of providing a yes-or-no indication about whether a site is language-critical, it might be useful to show the actual data from which this determination is made—namely, the number of trials and the number of naming errors that were made at each site. Figure 6.8 shows one such attempt. Each site number is accompanied by two rows of small boxes. The lower row (in light blue) shows the number of correct responses, while the upper row (in black) shows the number of errors. This redesigned style achieves the following:

1. It provides finer-grain information about naming errors, showing a ratio instead of a binary value. In other words, it helps the viewer answer a second question: "How reliable is this site's classification?"

2. It draws the viewer's eye to language-critical sites—the points of interest—by using

darker colors where errors occurred. In the old style, the eye needs to scan the entire image systematically to find the critical sites.

3. It visually distinguishes sites where many tests were performed, which are potentially more interesting and reliable than those where few were performed.

4. It contains more white space, which would make it easier for someone to annotate a printed copy of the map.

In Edward Tufte's theory of data graphics, each of these changes increases the *data-ink ratio*—the percentage of ink within the image that could not be erased without loss of information[64]. The redesigned image conveys significantly more information than the original, thanks to an economical use of data-ink.

6.4.2 Improving the coronary arteries

The goal of this example is clarifying the positions of the coronary arteries.

To provide more context, the original scene (figure 6.9a) was augmented with models of the myocardium, the pulmonary trunk, and the superior vena cava. This illustration must differentiate between three groups of structures: the contextual structures, the complete set of coronary arteries, and the individual coronary artery being highlighted. Therefore a three-group style was created. Context structures were assigned the ghost style, which simultaneously de-emphasizes them and makes it possible to see the vessels wrapping around the back of the heart. The light in the scene was also moved to coincide with the camera's position, so most of the shading would occur along the heart's silhouette. For highlighting, the standard style works well, because it retains the saturated red of the arteries. Figure 6.9b shows an example in which all of the arteries have been highlighted. (Note that the ascending aorta, which dominated the original scene, has been downplayed by assigning it to the context group.)

When highlighting a particular artery, the others need to be de-emphasized. Therefore this third grouping was assigned a tone rendering style, which causes most of each artery's area to be drawn in white, with just enough coloration at its boundaries to make its outline

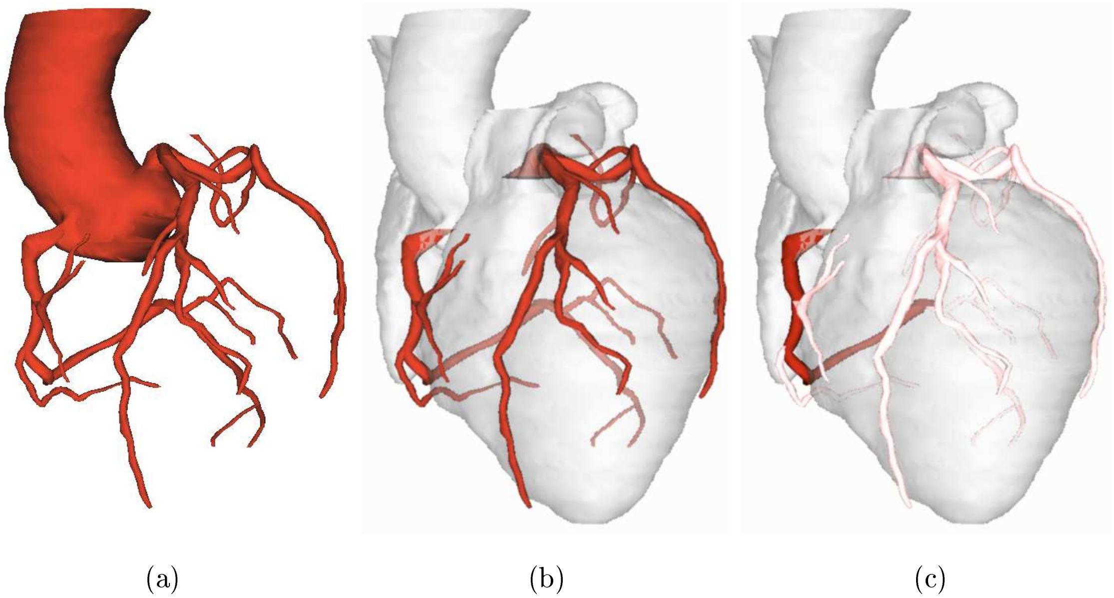

(a) (b) (c)

Figure 6.9: Redesigned illustrations of the coronary arteries. (a) The original image. (b) A revised version, which provides critical context information. (c) A version which highlights the trunk of the right coronary artery.

clear. Figure 6.9c shows the result. Note how the three styles clearly differentiate the three structure groups, even for those arteries on the far side of the heart.

6.5 Conclusions

The examples in this chapter give a taste of the challenges that face a medical illustrator. While computer-based tools have expanded the options available to illustrators, the unique needs of this field have not been adequately addressed. The requirements described here provide a first step toward fixing that oversight.

Pendragon offers an example of how these requirements can be used to design an anatomy illustration system. It demonstrates how non-photorealistic techniques—even simple ones which use the standard graphics pipeline in unusual ways—can improve the quality of computer-generated images of anatomy. By applying styles not to individual models, but to conceptually related groups of objects, it paves the way for dynamic illustrations. Styles designed for one scene can be transferred to another, and individual components of a scene can be modified by the user, but without losing the original intent of the illustrator. Support for layer compositing makes it possible to build more complex styles, and also to design images with multiple information layers. And even though they only scratch the surface of possibilities, the resulting images are both promising and exciting.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

This dissertation addresses the challenges that surface when trying reconstruct and illustrate structures from volume data. Its primary contributions are:

An in-depth analysis of two methods that use shape knowledge to guide 2D image segmentation. Several batteries of tests were run to compare the M-model, which uses bounds on ratios of radial lengths to represent shape, with the P-model, which uses probability distributions and covariance information. The M-model was slightly less accurate than the P-model, but ran significantly faster because its method for constraint propagation has a lower order of complexity. Accuracy on particular organs was mixed, with better results achieved on regular structures like the eye than on variable structures like the liver. Relative to completely manual methods, both shape models reduced the average amount of work required to segment an organ by about two-thirds. Surprisingly, both models performed slightly better when information provided by detected edges was *not* used to shrink the search space. This behavior was observed even when the models were used on their own training data, suggesting that early commitment to incorrect edges causes the segmented boundary to diverge from its true location. The tests for both models used a greedy algorithm for selecting edge candidates, so a less greedy strategy might alleviate this problem.

A novel algorithm for 3D brain segmentation. To reconstruct the brain's surface from an MRI volume, a two-stage method can be adopted: first fit a shape model to the MRI data to isolate the cortex, then extract an isosurface from the region inside the fitted model to achieve a more detailed surface reconstruction. Unlike most other methods for segmenting the brain, this method explicitly addresses the problem of "scalping" the volume data to isolate the cortex. In addition, the fitted model provides the user with an intuitive method for making local corrections to the segmentation results.

A working, interactive software system for reconstructing, mapping, and visualizing the human brain. A new segmentation algorithm is of little use in isolation; it needs to be applied. The Brain Mapper demonstrates how the shape-based approach to segmentation can help solve a real-world problem. The Brain Mapper is an interactive tool for reconstructing the cortical surface and its superficial blood vessels from three sets of MRI data; these reconstructions are then used to recover the 3D locations of language sites gathered during cortical stimulation mapping. By employing a lightweight workflow architecture to connect the various steps of the mapping process, the system provides a unified interface and thereby simplifies the user's job. The user can complete the entire transformation from MRI data to a mapped reconstruction in about 90 minutes, and comparisons with intra-operative photographs verify the accuracy of the reconstructions. To date, the Brain Mapper has been used to map over 40 patients, and it is playing a key role in ongoing neuroscientific studies of language organization in the human brain.

Requirements for and a prototype of a medical illustration system. Although standard rendering tools can be applied to medical illustration problems, they often fail to meet the unique needs of the medical illustrator. By identifying requirements for a medical illustration system, this work makes a critical first step in an area that has received almost no attention from the computer science community. The Pendragon system demonstrates how these requirements could be put to use, providing non-photorealistic rendering techniques and methods for combining them to form high-level "styles" which can be transferred from one annotated scene to another.

7.2 Future Work

By necessity, any research project can tackle only a limited number of problems. There is always more work to be done. This section describes how the solutions proposed here could be expanded and improved.

7.2.1 Improving the Shape Models

Changing radial arrangements

The shape restrictions of the radial surface model impact the quality of reconstructions. For example, the brain RSM does a poorer job of isolating the inferior portions of the temporal lobes because of the glancing angle between those regions and the radials that sample them. Similar problems occur at the front and back poles of the brain, due to the fixed spacing of model slices. Such limitations are tolerable for the language mapping domain, in which the primary region of interest is the lateral surface of the left temporal lobe, but they might pose a problem for other applications.

These shortcomings could be addressed by changing the distribution of radials in the model. The most straightforward approach would be a uniform increase both in the number of radials per slice and in the number of slices. While this would provide better sampling, it would negatively impact system performance—as the number of radials increases, so does the time required to propagate shape constraints. It would also force the user to verify and potentially correct many more radials. (Nevertheless, the primary user of the system has requested just such an increase.)

A more sophisticated solution worthy of further investigation would be adding more radials only where necessary. Shape information computed from training examples could be used to identify places where sampling is too sparse and radials should be added. Such places could be identified by using the ratios of neighboring radials. If two adjacent radials are significantly different in length (*e.g.* one is more than three times as long as the other), then an extra radial could be added between them. This might help to maintain sufficient sampling over the entire cortical surface. However, some mechanism would be needed for computing the proper shape information for these extra radials.

More extreme changes could be made by abandoning the RSM's stack-of-slices arrangement altogether. However, any such modification would need to be accompanied by a method for editing such radials, raising the challenging 3D sculpting issues mentioned earlier.

Learning intensity profiles

Scanner's current segmentation strategy always looks for strong gradients. Yet clear edges are not always present. Furthermore, many organs have boundary regions where strong edges will *never* occur.

The radial surface model already captures location-specific shape information. It could be expanded to capture location-specific intensity information too. Suppose that training examples recorded not just radial lengths, but also the image intensities over a small region (*e.g.* 5×5) at the end of each radial. This extra data could be used to tune the segmentation process. Rather than always looking for strong gradients, the boundary search along each radial could be tailored to look for an intensity distribution similar to the training data. Such a strategy would likely provide much more accurate results than the system's current edge detector.

7.2.2 Improving Segmentation

Much could be done to improve Scanner's shape-guided segmentation strategy. The system's biggest failing is its greedy search mechanism. On each iteration of the search-and-propagate loop, the system uses edge information to select a new length for one of the radials. While the criteria for this selection may vary, the choice is final. If a wrong choice is made—which is inevitable, given the error-prone nature of edge detection—it will negatively influence subsequent choices and force the segmented boundary to diverge from its true location.

This problem could be alleviated by adopting a strategy that simultaneously considers multiple candidate edges per radial. For example, *relaxation labeling*[34] could be used to find those combinations of edge locations that are consistent with the shape constraints. For the P-model, a similar result could be accomplished with an extension to the Kalman filter algorithm, which uses constraints that are weighted mixtures of Gaussian distributions[4]. Each potential edge could be modeled as a Gaussian with a weight determined by the edge strength, and the algorithm could be used to choose the most likely set of edges.

7.2.3 Improving the Brain Mapper

The mapping procedure itself could be improved in a couple of ways. Given four points of correspondence between the photo and the 3D model (*e.g.* the locations of four stimulation sites), it is possible to estimate the parameters of the camera that took the photograph. These parameters could be applied to the virtual camera to obtain the best possible match between the two views, thus simplifying the user's mapping job. This idea could be taken a step further to partially automate the site recovery process. Given an in-focus photograph, optical character recognition techniques could be used to locate at least some of the site numbers. If the virtual camera were registered with the real camera, then these site locations could be transferred directly to the 3D model.

In the Brain Mapper, the radial shape model performs coarse-grain segmentation as input for isosurface extraction. Similar two-stage approaches could be built using other segmentation methods. For example, deformable models tend to succeed only when they are initialized near the correct solution. An RSM could be used to provide this critical starting point.

Ultimately, a more detailed segmentation of the brain is needed. Finding the cortical surface is an important first step, but locating individual sulci and gyri would also be useful for researchers. Identifying these smaller structures automatically is a difficult problem. The deformable brain model mentioned in section 2.3.4 showed some success with this task, and atlas-based segmentation methods might also work. A more practical approach might be a tool for painting regions onto the surface of the brain by hand[32]. Such a tool could also be used to mark other regions of interest, such as the portion of the brain that was exposed during surgery.

7.2.4 Improving Illustration

The example illustrations from Pendragon are promising, but the system needs a real user interface to become a truly useful tool. As mentioned earlier, Pendragon works with scenes generated by the Scene Builder. This application provides a powerful, web-based interface for retrieving anatomical models, and it already uses Skandha4 as a graphics server for rendering. An augmented version of the Scene Builder would provide a natural front-end to Pendragon's capabilities.

The requirements from section 6.2 are only partially addressed by Pendragon. The unsolved items from this list provide ample material for additional research. For instance, high-level knowledge about an object's shape could be used to choose its default lighting. Work has been done to associate abstract geometric labels (*e.g.* "cone" or "cylinder") with specific anatomical structures[49]. These classifications could be used to pick appropriate lighting. For example, cylinders generally look more "cylindrical" if they are lit perpendicular to their central axis, whereas cones look better if the light placed is closer to the apex than the base. Given some simple lighting rules and a structure's shape class, a reasonable light position could be guessed. This approach to providing default lighting could be very useful for a web-based anatomy browser, in which bandwidth and latency limitations preclude the use of an interactive lighting tool.

Ironically, the trend towards generic rendering can be blamed, at least in part, on rapid advances in hardware-based graphics acceleration. Applications that once required an expensive graphics workstation can now run on a home PC with a decent graphics card. Unfortunately, such progress has also handcuffed software developers, because the graphics hardware supports only a single rendering pipeline. As a result, creativity and variety are sacrificed in the name of speed. It would be nice to see graphics library or hardware support for a broader range of rendering styles. As the examples in chapter 6 show, object silhouettes are particularly important, and therefore would be a welcome addition to any rendering platform. Support for non-photorealistic methods has begun to appear in commercial-level applications[16], but more widespread availability is needed.

7.3 Putting It Together

In this work, considerable attention has been paid to putting pieces together to create a working system, echoing a recurring theme in medical informatics research. Even more could be done. For example, the previous chapter showed how Pendragon could be used to generate better brain map illustrations, but so far these features are not tied into the Brain Mapper's pipeline. Both are built within Skandha4, so combining them would not be difficult. Better yet, Pendragon could be turned into an illustration server, making its functionality available not just to the Brain Mapper, but to other applications as well. Prior work has already shown how Skandha4 can be used as a graphics server, providing rendering support for web-based applications[14]. This type of support for dynamic generation of medical illustrations could significantly impact the field of medical illustration, both for educational purposes and for clinical applications.

Small pieces, working together. Keeping the human in the loop. Using prior knowledge when it helps. These are the ideas that lead to a working system, one which is capable of helping human and computer alike to see structure.

BIBLIOGRAPHY

[1] Advanced Visualization Systems. *AVS*. http://www.avs.com/.

[2] A. Ailamaki, Y.E. Ioannidis, and M. Livny. Scientific workflow management by database management. In *Proceedings, Tenth International Conference on Scientific and Statistical Database Management*, pages 190–199, 1998.

[3] R.B. Altman and J.F. Brinkley. Probabilistic constraint satisfaction with structural models: Application to organ modeling by radial contours. In *Proceedings of the Annual Symposium on Computer Applications in Medical Care*, pages 492–496, 1993.

[4] R.B. Altman, C.C. Chen, W.B. Poland, and J.P. Singh. Probabilistic constraint satisfaction with non-gaussian constraint noise. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 15–22, August 1994.

[5] F. Amador, D. Berman, A. Borning, T.D. DeRose, A. Finkelstein, D. Neville, D. Notkin, D.H. Salesin, M. Salisbury, J. Sherman, Y. Sun, D. Weld, and G. Winkenbach. Electronic "how things work" articles: Two early prototypes. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):611–618, August 1993.

[6] A. Appel. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of ACM National Conference*, pages 387–393, 1967.

[7] K. Bathe. *Finite Element Procedures in Engineering Analysis.* Prentice-Hall, 1982.

[8] J.C. Bezdek, L.O. Hall, and L.P. Clarke. Review of MR image segmentation techniques using pattern recognition. *Medical Physics*, 20(4):1033–1048, 1993.

[9] J.C. Bezdek and S.K. Pal. *Fuzzy Models for Pattern Recognition.* IEEE, Piscataway, New Jersey, 1992.

[10] J. Bloomenthal. Polygonization of implicit surfaces. *IEEE Computer Graphics and Applications*, 5(4):341–355, 1988.

[11] J.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):121–142, April 1993.

[12] J.F. Brinkley, L.M. Myers, J.S. Prothero, G.H. Heil, J.S. Tsuruda, K.R. Maravilla, G.A. Ojemann, and C. Rosse. A structural information framework for brain mapping. In S.H. Koslow and M.F. Huerta, editors, *Neuroinformatics: An Overview of the Human Brain Project*, chapter 9, pages 309–334. Lawrence Erlbaum, Mahwah, New Jersey, 1997.

[13] J.F. Brinkley and J.S. Prothero. Slisp: A flexible software toolkit for hybrid, embedded and distributed applications. *Software—Practice and Experience*, 27(1):33–48, 1997.

[14] J.F. Brinkley, B.A. Wong, K.P. Hinshaw, and C. Rosse. Design of an anatomy information system. *IEEE Computer Graphics & Applications*, 19(3):38–48, May/June 1999.

[15] H.S. Choi, D.R. Haynor, and Y. Kim. Partial volume tissue classification of multichannel magnetic resonance images — a mixel model. *IEEE Transactions on Medical Imaging*, 10(3):395–407, September 1991.

[16] P.H. Christensen. Contour rendering. *Computer Graphics*, 33(1):58–61, February 1999.

[17] T.F. Cootes and C.J. Taylor. Combining point distribution models with shape models based on finite element analysis. *Image and Vision Computing*, 13(5):403–409, June 1995.

[18] T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham. Active shape models — their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, January 1995.

[19] D.P. Corina, S.L. McBurney, C. Dodrill, K. Hinshaw, J. Brinkley, and G. Ojemann. Functional roles of Broca's area and SMG: Evidence from cortical stimulation mapping in a deaf signer. *NeuroImage*, 10:570–581, 1999.

[20] C.J. Curtis, S.E. Anderson, J.E. Seims, K.W. Fleischer, and D.H. Salesin. Computer-generated watercolor. In *Proc. SIGGRAPH*, pages 421–430, 1997.

[21] C. Davatzikos and R.N. Bryan. Using a deformable surface model to obtain a shape representation of the cortex. *IEEE Trans. Medical Imaging*, 15(6):785–795, December 1996.

[22] D. Dooley and M.F. Cohen. Automatic illustration of 3D geometric models: Lines. *Symp. Interactive 3D Graphics*, 24(2):77–82, March 1990.

[23] D. Dooley and M.F. Cohen. Automatic illustration of 3D geometric models: Surfaces. In *Proc. of IEEE Visualization*, pages 307–314, October 1990.

[24] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, August 1988.

[25] S.K. Feiner and D.D. Seligmann. Cutaways and ghosting: Satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer*, 8:292–302, 1992.

[26] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, second edition, 1990.

[27] J.C. Gee. On matching brain volumes. *Pattern Recognition*, 32(1):99–111, January 1999.

[28] S.F. Frisken Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings, IEEE Symposium on Volume Visualization*, pages 23–30, 1998.

[29] *The GNU Image Manipulation Program.* http://www.gimp.org/.

[30] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proc. SIGGRAPH*, pages 447–52, 1998.

[31] P. Haeberli. Paint by numbers: abstract image representations. *Computer Graphics*, 24(4):207–214, 1990.

[32] P. Hanrahan and P. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. *Computer Graphics*, 24(4):215–223, August 1990.

[33] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proc. SIGGRAPH*, pages 453–460, 1998.

[34] R.A. Hummel and A. Rosenfeld. Relaxation processes for scene labeling. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-8(10):765–8, October 1978.

[35] R.K. Justice, E.M. Stokely, J.S. Strobel, R.E. Ideker, and W.M. Smith. Medical image segmentation using 3-D seeded region growing. In *Proc. SPIE Image Processing*, volume 3034, pages 900–10, 1997.

[36] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME—Journal of Basic Engineering*, pages 35–45, March 1960.

[37] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, January 1987.

[38] M. Kobashi and L.G. Shapiro. Knowledge-based organ identification from CT images. *Pattern Recognition*, 28(4):475–491, April 1995.

[39] P. Litwinowicz. Processing images and video for an impressionist effect. In *Proc. SIGGRAPH*, pages 407–414, 1997.

[40] W.E. Lorensen and H.E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *Computer Graphics*, volume 21, pages 163–170, 1987.

[41] D. MacDonald. *Register.* Montreal Neurological Institute, 1993.

[42] D. MacDonald, D. Avis, and A.C. Evans. Multiple surface identification and matching in magnetic resonance images. In *Proc. SPIE Visualization in Biomedical Computing*, volume 2359, pages 160–169, 1994.

[43] L. Markosian, J.M. Cohen, T. Crulli, and J.F. Hughes. Skin: A constructive approach to modeling free-form shapes. In *Proc. SIGGRAPH*, pages 393–400, 1999.

[44] L. Markosian, M.A. Kowalski, S.J. Trychin, L.D. Bourdev, D. Goldstein, and J.F. Hughes. Real-time nonphotorealistic rendering. In *Proc. SIGGRAPH*, pages 415–420, 1997.

[45] P. Maybeck. *Stochastic Models, Estimation, and Control, Volume 1.* Academic Press, Inc., 1979.

[46] B.J. Meier. Painterly rendering for animation. In *Proc. SIGGRAPH*, pages 477–484, 1996.

[47] B. Modayur, J. Prothero, G. Ojemann, K. Maravilla, and J. Brinkley. Visualization-based mapping of language function in the brain. *Neuroimage*, 6(4):245–258, 1997.

[48] B.R. Modayur, J. Prothero, C. Rosse, R. Jakobovits, and J.F. Brinkley. Visualization and mapping of neurosurgical functional data onto a 3-D MR-based model of the brain surface. In *Proceedings, American Medical Informatics Association Fall Symposium*, pages 304–308, 1996.

[49] P.J. Neal, L. G. Shapiro, and C. Rosse. The digital anatomist spatial abstraction: a scheme for the spatial description of anatomical features. In *Proceedings, American Medical Informatics Association Fall Symposium*, pages 423–427, 1998.

[50] F.H. Netter. *Atlas of Human Anatomy.* CIBA-GEIGY Corporation, Summit, New Jersey, 1989.

[51] G.A. Ojemann, J. Ojemann, E. Lettich, and M. Berger. Cortical language localization in left, dominant hemisphere. *J. Neurosurgery*, 71:316–326, 1989.

[52] A. Pentland and S. Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(7):715–729, July 1991.

[53] A.V. Poliakov, K.P. Hinshaw, C. Rosse, and J.F. Brinkley. Integration and visualization of multimodality brain data for language mapping. In *Proceedings, American Medical Informatics Association Fall Symposium*, pages 349–353, 1999.

[54] R. Raskar and M. Cohen. Image precision silhouette edges. In *Symposium on Interactive 3D Graphics*, 1999. In press.

[55] M.P. Salisbury, M.T. Wong, J.F. Hughes, and D.H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proc. SIGGRAPH*, pages 401–406, 1997.

[56] D.J. Schlesinger, J.W. Snell, L.E. Mansfield, J.R. Brookeman, J.M. Ortega, and N.F. Kassell. Segmentation of volumetric medical imagery using multiple geodesic-based active surfaces. In *Proc. SPIE Image Processing*, volume 2710, pages 243–253, 1996.

[57] D.D. Seligmann and S.K. Feiner. Automated generation of intent-based 3D illustration. In *Proc. SIGGRAPH*, pages 123–132, July 1991.

[58] C.A. Sneiderman, T.C. Rindflesch, and C.A. Bean. Identification of anatomical terminology in medical text. In *Proceedings, American Medical Informatics Association Fall Symposium*, pages 428–32, 1998.

[59] M.C. Sousa and J.W. Buchanan. Computer-generated graphite pencil rendering of 3D polygonal models. In *Proc. Eurographics*, pages 195–207, 1999.

[60] J.W. Sundsten and J.W. Prothero. Three-dimensional reconstruction from serial sections: II. a microcomputer-based facilty for rapid data collection. *The Anatomical Record*, 207:665–671, 1983.

[61] R. Szeliski and D. Terzopoulos. Physically-based and probabilistic models for computer vision. In *Geometric Methods in Computer Vision*, volume 1570, pages 140–152. SPIE, 1991.

[62] J. Talairach and P. Tournoux. *Co-Planar Stereotactic Atlas of the Human Brain: 3-Dimensional Proportional System: An Approach to Cerebral Imaging.* George Thieme Verlag, Stuttgart, 1988.

[63] P.M. Thompson, D. MacDonald, M.S. Mega, C.J. Holmes, A.C. Evans, and A.W. Toga. Detection and mapping of abnormal brain structure with a probabilistic atlas of cortical surfaces. *Journal of Computer Assisted Tomography*, 21(4):567–581, 1997.

[64] E.R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut, 1983.

[65] E.R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1990.

[66] G. Welch and G. Bishop. An introduction to the Kalman filter. Technical Report 95-041, University of North Carolina at Chapel Hill, 1995.

[67] P.L. Williams, R. Warwick, M. Dyson, and L.H. Bannister, editors. *Gray's Anatomy.* Churchill Livingstone, Edinburgh, thirty-seventh edition, 1989.

[68] G. Winkenbach and D.H. Salesin. Computer-generated pen-and-ink illustration. In *Proc. SIGGRAPH*, pages 91–100, 1994.

VITA

Kevin Hinshaw has lived his entire life along the I-5 corridor of Washington. Born in Seattle, he grew up in Vancouver and Olympia, graduating from Olympia High School in 1988. (Go Bears.) He then moved thirty miles north to attend the University of Puget Sound in Tacoma, where he earned the BS degree in Computer Science/Math in 1992. He then moved *another* thirty miles north to attend the University of Washington, where he earned the MS degree in Computer Science in 1994. Along the way, Kevin has learned to play the guitar, recorder, piano, clarinet, french horn, saxophone, uilleann pipes, conga drum, tin whistle, low whistle, and doumbek. He hopes to find a real job soon after graduating, so that he can afford even more instruments and—perhaps more importantly—more space for storing them.