1996

Three-Dimensional Segmentation and Visualization of Magnetic Resonance Imaging Data

William L. Bell Jr.
Univeristy of North Florida

THREE-DIMENSIONAL SEGMENTATION AND VISUALIZATION OF MAGNETIC RESONANCE IMAGING DATA

by

William L. Bell, Jr.

A thesis submitted to the Department of Computer and Information Sciences in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

November, 1996

The thesis "THREE-DIMENSIONAL SEGMENTATION AND VISUALIZATION OF MAGNETIC RESONANCE IMAGING DATA" submitted by William L. Bell, Jr. in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

Signature Deleted

Nov 27, 1996

Yap S. Chua
Thesis Adviser and Committee Chairperson

Signature Deleted

11/27/96

Ralph M. Butler

Signature Deleted

11-27-96

Behrooz Seyed-Abbassi

Accepted for the Department of Computer and Information Sciences:

Signature Deleted

11/27/96

Charles N. Winton
Chairperson of the Department

Accepted for the College of Computing Sciences and Engineering:

Signature Deleted

11/27/96

Charles N. Winton
Acting Dean of the College

Accepted for the University:

Signature Deleted

12/3/96

William J. Wilson
Dean of Graduate Studies

ACKNOWLEDGMENT

Although many people have contributed to the achievement of my educational goals, certain individuals are especially deserving of my gratitude:

- My wife Kathi, for her support and toleration of my unhealthy preoccupation with my studies;
- my parents, for providing me with a strong work ethic;
- Dr. Yap Siong Chua, my thesis adviser, for his friendly help and encouragement;
- Dr. Charles N. Winton, my graduate studies adviser, for his wise counsel throughout my career as a graduate student;
- Dr. Ralph M. Butler and Dr. Behrooz Seyed-Abbassi, for generously donating their time to serve as members of my thesis committee;
- and from Jacksonville University, my friends and colleagues Dr. Marilyn L. Repsher, Prof. Dennis W. Dormady, and the late Dr. R. Wayne Hamm, who provided me with much of the original inspiration to pursue a higher degree of education and accomplishment.

CONTENTS

TABLES

FIGURES

ABSTRACT

In this thesis, I shall study and compare various methods for manipulating two- and three-dimensional image data produced with a nuclear magnetic resonance scanner. In particular, I will examine ways of focusing upon specific structures internal to the object under study (segmentation); and will explore means of rendering realistic images of these structures on a computer screen using depth-cueing, shading, and ray-casting techniques.

The 3DHEAD volumetric dataset used for this project was created with the Siemens Magnetom and was provided courtesy of Siemens Medical Systems, Inc., Iselin, NJ. This dataset consists of 109 slices of a human head, with each slice stored consecutively as a 256 x 256 array. Each pixel is represented by two consecutive bytes, which make one binary integer. (A similar dataset of a human knee is also available.) The 3DHEAD dataset requires about 14 Mb of disk space uncompressed. The programs which manipulate this data are MS-DOS-based and were written and compiled using Microsoft QuickC version 2.51. The 2-D programs were executed on a CompuAdd 486DX/2-50 with 8 Mb of RAM, running MS-DOS version 6.22; the 3-D programs were executed on a 133 MHz Pentium clone with 48 Mb of RAM, running the DOS shell of Microsoft Windows 95.

Our immediate objectives are to produce pleasing and informative 2-D and 3-D pictures of the internal structure of some component of the human head: for example, the brain.

We need to remove from the original dataset all of the data which do <u>not</u> represent the brain. Then, for the 3-D images, we need to render the remaining data in such a way that it possesses depth and realism.

The overall job can be divided into three smaller tasks:

(1) Acquire the range of densities of brain tissue, expecting that the brain will not be all of one uniform density, but that it will be fairly homogeneous.

(2) Filter out all "non-brain" data from the original dataset, using density or density gradients as the criteria for segmentation.

(3) Use the remaining "brain" data to create a realistic computer image.

In [Udupa82], the authors suggest the following steps for 3-dimensional organ display:

1. segmentation of the three-dimensional array into regions corresponding to organs
2. identification of the organ of interest
3. detection of its boundary
4. hidden surface removal
5. shading

Using the approach described in this thesis, we shall already have identified the organ of interest using two-dimensional images taken from the original three-dimensional dataset. We will proceed to segment data corresponding to the organ of interest using various boundary-detection algorithms. Finally, we will create realistic images from the remaining data through the use of rendering algorithms which both remove hidden surfaces and simulate variations in the light level.

CHAPTER 1

INTRODUCTION

We begin our exploration into the subject of segmentation and rendering with a discussion of some important background information about the technology, terms, and concepts to be encountered.

1.1 Nuclear Magnetic Resonance and Magnetic Resonance Imaging

Nuclear magnetic resonance (NMR) is a phenomenon which occurs when atoms are exposed to extremely strong magnetic fields. Atomic nuclei (particularly those of hydrogen atoms) tend to line up along the axis of the magnetic field to which they are exposed. Then, a radio-frequency pulse is directed at the atoms, which causes the nuclei to twist out of alignment. When the pulse is removed, the atoms "relax" and, in the process, give off RF signals which are measured and analyzed by a computer.

NMR technology is particularly important in the field of medicine. A living human being (which, by virtue of being composed mostly of water and aliphatic [fatty] compounds, is especially rich in hydrogen) may be placed inside a large electromagnet, and his internal tissues and organs may be scanned and their hydrogen-ion densities recorded for study. The densities may be correlated to different colors on a computer screen, enabling physicians to discern the different organs and to discover abnormalities such as tumors, breaks, or lesions. [Higgins84] states:

> The gray scale of the NMR images displays fat as the brightest intensity (white), followed by brain and spinal cord, solid viscera, vessel wall, and muscle in descending order. Air, bone, and calcification produce almost no MRI signal (black)... Fat in the bone marrow produces high signal intensity... Fluid-filled cavities tend to be low intensity.

Magnetic resonance imaging (MRI) scanners make it possible to scan a body in many small increments along some axis, effectively producing many very thin "slices" of data. These slices may be stacked to create a 3-dimensional image of the body. In addition, advantage may be taken of the fact that, since different components of the body (organs, bone, blood vessels, etc.) possess different hydrogen densities, some organs' data may be filtered out, or segmented, so as to reveal only those organs which the imager desires to study in detail. MRI requires neither surgery nor contrasting dye nor ionizing radiation to create its images; the patient need only lie very still while the image is generated [Bell94B]. (With the advent of new high-speed MRI scanning devices, even this requirement is somewhat relaxed.) Exposure to intense magnetic fields has not been shown to be harmful to living tissue.

1.2 Visualization

Scientific visualization has been described as "[the development of] algorithms and methods that transform massive scientific datasets into pictures and other graphic representations that facilitate comprehension and interpretation" [Samtaney94]. Datasets may contain values derived from studies of fluid flow, weather patterns, stock market index fluctuations, or, in the present case, relative hydrogen density as determined by MRI scans of the human body.

On a practical level, one must consider certain properties relevant to visualization: in particular, the generation, manipulation, storage, and display of data [Ranjan94]. Visualization gives us the ability to observe trends and relationships present, but perhaps hidden or obscured, in complex datasets. With respect to this project, the relationship we seek is that of connectedness, or homogeneity, of certain organic tissue in the human body. The challenging aspect of the visualization task is that the structures we seek to visualize are hidden from direct view, and have been heretofore observable only through the use of (a) invasive techniques such as surgery, or (b) non-invasive techniques involving X-rays or substances which emit ionizing radiation. Both techniques involve some risk to the patient; and in any event, the quality of the resulting information often leaves much to be desired.

1.3 Computed Tomography

Computed Tomography (CT) is described as the mathematical reconstruction of internal structural information within an object from a series of projections [Russ95]. In the present context, our projections are 2-dimensional "slices" of relative density data provided by an MRI scan along some axis at regular intervals. The slices are stacked to reconstruct a 3-dimensional volumetric dataset (VDS), to which we apply various methods of computation for separating or segmenting tissue types of interest.

1.4 Segmentation

Segmentation is the process of extracting meaningful regions from images or volumes [Carlbom92]. According to [Schalkoff89], segmentation groups pixels to form higher-level regional image structures in a manner which is either non-contextual or contextual.

During non-contextual segmentation, relationships between features (at either the pixel- or region-level) are ignored. Instead, the process relies upon the recognition of a statistical pattern in the value of the pixels under consideration. For example, the density-range-based approach to biomedical image segmentation employed in [Bell94B] depended upon each pixel's density value falling within a specified range, without regard for the density value of its neighbors. As noted in section 1.4.2 (Thresholding), this approach has its flaws.

The contextual segmentation process, on the other hand, considers the relationships between neighboring pixels to support the decision to include a particular pixel in the region being segmented. In other words, we assess the local pixel region content, rather than simply the value of each individual pixel. Edge-detection and density-gradient analysis are two methods which assess the contents of a local pixel region.

Segmentation involves two considerations. First, in order to acquire data for a particular homogenous and connected structure, we must apply some technique for "growing" a region of data from a specified seedpoint. Second, we must select from our dataset only those values meeting specified criteria (thresholding).

1.4.1 Region-growing

Region-growing begins with the selection of a seedpoint, a single data point located within the region to be segmented from the surrounding area. Each surrounding point is examined and is added to the region if its value is sufficiently similar and if the point is connected, that is to say, adjacent to the point which came before. The region grows in all directions until no more points are encountered which meet the criteria for homogeneity. Region-growing can be performed in either two or three dimensions.

Because of the recursive nature of the problem, the implementation of region-growing algorithms is not conceptually difficult. However, the large number of recursive calls likely to be encountered during segmentation can strain a computer's memory resources. Therefore, we explore region-growing algorithms which implement recursion to a lesser extent, or simulate recursion through an iterative process [Tenenbaum90] [Rohl84] [Foley90].

1.4.2 Thresholding

During the thresholding phase of segmentation, we evaluate each datum in the set under consideration, retaining it only if it is above or below some limit (threshold), or within some specified range of limits. In a previous experiment [Bell94B], a simple density-threshold algorithm was applied to a VDS of MRI density data for a human skull, with the intention of segmenting brain tissue. The results were less than satisfactory, due to the fact that (a) the density range of brain tissue is fairly broad, and (b) the density ranges of other types if tissue found in the skull overlap that of brain tissue. Also, no attempt was made to grow a region from a seedpoint. Therefore, brain tissue was not clearly and distinctly segmented from adjacent tissue.

Although a density-range thresholding scheme might be an adequate criterion under some circumstances, it could turn out that tissue of one type (X) with a certain density range may lie adjacent to tissue of a different type (Y) with an overlapping density range. In this case, complete segmentation of (X) from (Y) would be unsuccessful. Or, due to a narrow specification of density range, a region could have many of its data points discarded and thus be incompletely represented.

1.4.3 Gradient Approximation and Edge Detection

In either case, we might wish to examine the rate of change in density over the region. This rate of change is referred to as the gradient (∇). A low gradient indicates small or smooth changes in density, characteristic of tissue which is homogeneous. A high gradient indicates a sudden change in density, very likely a boundary between tissue types. In image processing, algorithms which measure rates of change over a region are often used to detect edges and boundaries between dissimilar areas.

An approximation of the gradient of a particular area may be determined using weighted-sum masking [Bell94A], where the kernels to be used are especially designed to display a strong response to changes in intensity. In three dimensions, such operators approximate the overall gradient in a volume by computing gradients in three orthogonal directions and summing them. In this project, we compare the characteristics of several different kinds of operator. A mathematical presentation of the notion of gradient may be found in Appendix B.

1.5 Surface Rendering

When we display a slice of hydrogen-density data on a computer screen, each point of data from the slice is represented by a single point on the screen. The color (or gray-shade) of the point on the screen corresponds to the value of the datum. Throughout this paper, reference will be made to "pixels" and "voxels". A pixel (picture element) refers to a single point on a computer screen; each pixel, and each point in a slice of data, may be referenced by a set of coordinates (x, y). A voxel (volume element) refers to a single point in a volume of data at a set of coordinates (x, y, z). Since an image on a computer screen is two-dimensional, we will need to resort to various techniques to give our images the illusion of depth or realism.

Surface rendering is the process of adding the appearance of a (more or less) realistic surface to a graphical object on a computer screen. Complex mathematical operations on a scene's dataset add realism to the scene through the artful use of color, shading, lighting, reflectivity, and refractivity.

1.5.1 Depth-cueing

One method of adding a sense of three-dimensionality to an object is known as depth-cueing. This involves correlating the brightness level of each voxel in the object to the voxel's distance from the viewer. Therefore, closer voxels appear brighter, and more distant voxels appear more dim.

Although the depth-cueing method is not computationally difficult, it ignores issues important to realism, such as the location of point light sources, ambient lighting, surface texture, and the degree of specular reflection ("hot spots") at any given point in the object. A surface-rendering method which addresses these issues is called ray-tracing.

1.5.2 Ray-tracing

Ray-tracing begins by tracing backwards the path of an imaginary single ray of light (called an eye-ray) from the viewer's eye, through a viewscreen, to a region in the scene, and eventually back to some point light source of known coordinates. During this process, we may compute not only the path of the eye-ray (which may, in fact, be reflected by or refracted through one or more surfaces on its journey), but also the brightness of the ray, which diminishes both in proportion to the distance it travels and as a result of reflecting off coarse surfaces and refracting through translucent material.

Ray-tracing determines the visibility of an area on an object's surface by examining the relationship of an eye-ray's vector direction to the normal vector of the surface; if the vector's included angle is less than 90 degrees, then the surface is visible. To define an object's surface, an array of points is created containing the three-dimensional coordinates of each point. We regard the object's surface as a collection of small triangular patches, whose vertices are specified in the array. The vertices describe a plane whose normal vector is computed for comparison with the eye-ray vector.

1.5.3 Lighting and Shading Techniques

Depth-cueing provides to a scene a sense of three-dimensionality by diminishing the brightness of the more distant parts of an object. Ray-tracing makes it possible to simulate the location of point light sources in the scene, and to remove the hidden surfaces of an object. However, the presence of "ambient" light (light which may have come from a point light source, but which has since been scattered by atmospheric disturbances) affects the perceived brightness of an image. Also, the texture of the surface of an illuminated object will determine not only how much light is reflected, but also the degree to which reflected light is highly concentrated in one spot (specular reflection). Finally, the amount of reflected light seen by the viewer must be correlated to the angle at which the viewer observes the different aspects of the object.

All of these factors are dealt with through the use of various illumination models. These models view the amount of light seen by the observer as the sum of ambient light and (possibly multiple sources of) specular reflected light, adjusted with coefficients and trigonometric relationships to account for the texture of the surface, the viewing angle, and atmospheric attenuation [Foley90].

1.6 Storage Considerations

In [Bell94B], the Z-buffer was used as a method of representing the visible aspect of certain volumetric data. The Z-buffer is a two-dimensional array which maintains distance information for the closest voxels which have been seen so far in a slice-by-slice pass through the VDS of dimensions n x n. (Note that the dimensions of the Z-buffer are the same as those of a single slice of the VDS.) The Z-buffer offers the advantage of requiring relatively little memory, representing n^2 points rather than n^3 points. However, the Z-buffer method is also limiting in that, in order to rotate the object and view it from a different angle, each point in the original VDS must be re-examined.

An alternative method for representing a segmented 3-D object would be to copy the entire object to a second n^3 array; this approach is called spatial occupancy enumeration [Foley90]. The obvious disadvantage to this approach is the requirement for twice as much memory or disk space in which to hold the data. In addition, much of the second array would be used only to store zeros, indicating the absence of points in the segmented object (in other words, a sparse matrix).

A data structure which has proven popular for maintaining sparse matrix 3-D data is the octree [Foley90]. The idea behind the octree storage method is to recursively subdivide a 3-D scene into octants. Each octant is evaluated as being full, partially full, or empty, depending on how much of the octant intersects the volume of the object. A partially full octant is further subdivided, until its suboctants are evaluated as being all full or all empty (or until some cutoff is reached). Whenever eight sibling octants are all full or all empty, they are merged back into their single parent, which is then marked full or empty accordingly. Represented as a tree, the root node and intermediate nodes are always partially full; leaf nodes are always either full or empty. (A bottom-up approach is also possible, resulting in improved efficiency.)

In [Kippenhan94], the authors suggest the use of a hierarchical matrix of pointers to store and access volumetric data:

The "base pointer" will be a pointer to an array of sub-pointers (each of which will point to a "slice" of volume data), each of which will in turn be pointers to another array of sub-pointers (each of which will point to a "row" of data within the slice of interest).

CHAPTER 2

ANCILLARY SOFTWARE TOOLS AND LIBRARIES

For this project, various utility programs were required for the extraction and manipulation of slices of data from the VDS. Files of general- and special-purpose C functions were created which could be called from different programs as needed; here follows a brief description of the role of each utility in the overall mission.

2.1 ANYSLICE

This program was written for use on an Intel 80x86-based computer running the DOS operating system. ANYSLICE is designed to work with files of 3-D volumetric data stored in "slice" format, such as the MRI or CT files available from the University of North Carolina/Chapel Hill (see Appendix A). These files contain some number (typically around a hundred) of slices of tomographic data stored sequentially in z-y-x order. Every two bytes in each slice represents the magnetic resonance datum for the point at (x, y) in slice (z).

ANYSLICE is designed to acquire a specified slice of data from the specified input file and write that slice's data to the specified output file. The resulting binary file should be 131,072 bytes in length for a volumetric data file whose slices measure 256 x 256 pixels.

File slices shall be numbered by the user beginning with 1. Within the program, we revert to numbering the slices beginning with 0 after having called the atoi() function to get the slicenumber argument. atoi() returns 0 if its argument was '0' or if it was unsuccessful in converting the ASCII string it was passed. Furthermore, the user may specify the orientation of the desired slice (sagittal, coronal, or transverse) with the final argument "s", "c", or "t".

In order to understand these terms, consider three axes at right angles to each other: the x-axis passes from the front of the head to the back, the y-axis parallel from the top of the head downward through the spinal column, and the z-axis through the ears from left to right,. A sagittal slice of the dataset is parallel to the x-y plane; a coronal slice is parallel to the y-z plane; and a transverse slice is parallel to the x-z plane.

Usage: ANYSLICE <slicenumber> <infile> <outfile> <orientation>

Example: ANYSLICE 54 3dhead slice54.dat s

2.2 SLICE2IP

This program inverts the byte-order of, and quantizes to the range 0-255, the pixel-integers present in a slice-file created by ANYSLICE. The modified file will be suitable for viewing with the MIDTERM image-processing program [Bell94A]. The user should record the lower and upper histogram-scaling limits found to provide the most pleasing contrast between the different components shown in the slice data. These limits will be given to the GETCOORD program in order to display a better image (although the actual raw data underlying the image will be unaffected).

Usage: SLICE2IP <inputfile> <outputfile>

2.3 ROTATE

This program takes as its input a RAW-format density data file (a 256 x 256 array of 2-byte integers); rotates the data 90 degrees clockwise; and produces an output file of the same format as the input.

ROTATE is designed to be used in conjunction with data files produced by the ANYSLICE program, which creates slice-wise data files from 3-dimensional volumetric datasets. Since the slice files created may not always be in a pleasing orientation (e.g. a coronal slice lying on its side), some means of rotating the picture will be found useful.

To rotate a picture by 180 or 270 degrees, simply apply ROTATE twice or three times, respectively:

command	result
ROTATE original.dat 90.dat	---> 90 degrees clockwise
ROTATE 90.dat 180.dat	---> 180 degrees clockwise
ROTATE 180.dat 270.dat	---> 270 degrees clockwise
ROTATE 270.dat 360.dat	---> 360.dat == original.dat

Usage: ROTATE <inputfile> <outputfile>

2.4 ANYVIEW

ANYVIEW allows the user to extract from a VDS file a Z-buffer view of the contents of that VDS file from one of the three axial orientations (x, y, or z). ANYVIEW's output is a 256x256x8 RAW file of grayshade data.

Usage: ANYVIEW <infile> <outfile> <orientation>

Example: ANYVIEW 3dhead sagview.raw s

2.5 XMSIF

XMSIF (version 1.5, written by James W. Birdsall, copyright 1993) is a C interface to extended memory functions, and is widely available via FTP to the Internet. Many of its routines were incorporated into the MRI3D program in order to create and manipulate a very large stack in extended memory.

2.6 VSA256

VSA256 (version 3.01, written by Spyro Gumas, copyright 1994) is a C interface to functions which make it possible to generate graphics output on video adapters running with VESA BIOS extensions. Use of VSA256 makes it possible to activate the high-resolution, 256-color video modes necessary to display multiple grayshades.

CHAPTER 3

IMPLEMENTATION IN TWO DIMENSIONS

The GETCOORD program was created for the purpose of performing segmentation in two dimensions, both as an end in itself and as a preliminary step towards successful 3-D segmentation. GETCOORD takes as its input a slice-file of 2-D density data, such as that created by the ANYSLICE utility from the original VDS. This slice is displayed on the screen. (Prior to loading, the intensity histogram of the slice-file may be modified to improve its appearance and usability.)

Using the mouse, the user points to any pixel in the image to be used as a seedpoint for region-growing. Via the keyboard, the user also specifies:

- which thresholding method is to be used
- the parameters for lower and upper density limits
- the density-gradient threshold
- the manner of floodfilling

When segmentation is complete, the user may save the coordinates of the seedpoint in a small text file (to be used in the 3-D segmentation process implemented by the program MRI3D). The user may also save the segmented image as a RAW-format file (suitable for viewing with the MIDTERM image-processing program [Bell94A]) or as an encapsulated PostScript file (which includes all of the settings information for the current image).

GETCOORD, an MS-DOS-based program, requires 16-color VGA and 640 kb of conventional memory to run. The usage of the GETCOORD program is:

```
GETCOORD <input_slice_filename> <slicenumber>
```

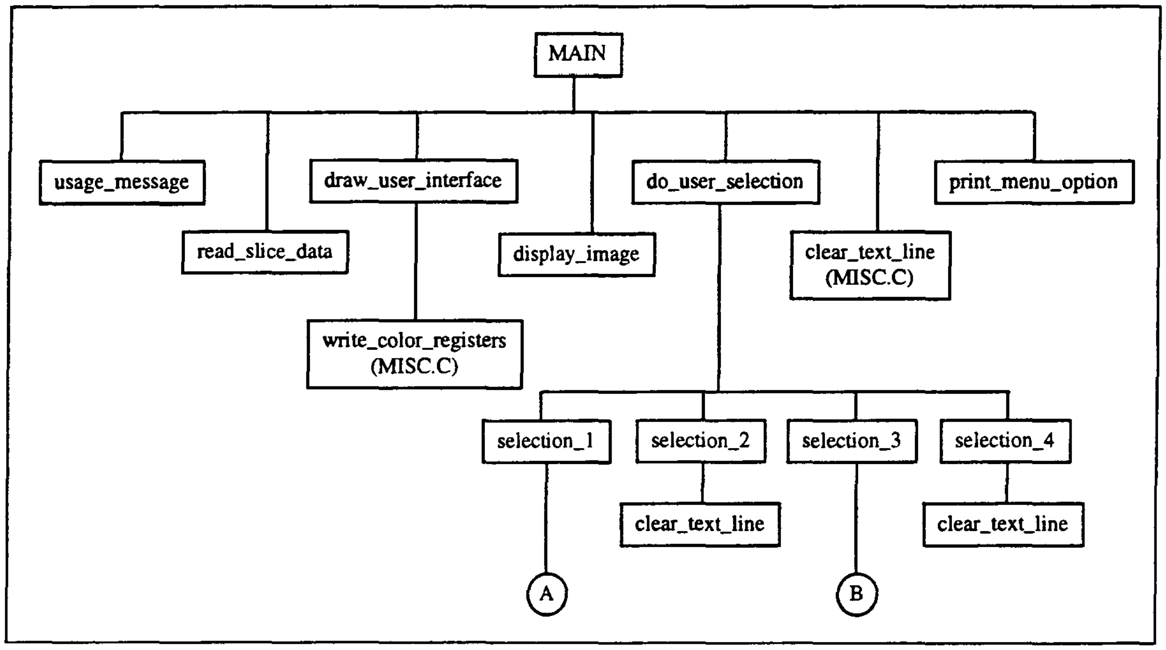


Figure 3.0-1a: GETCOORD Structure Chart (1 of 3)

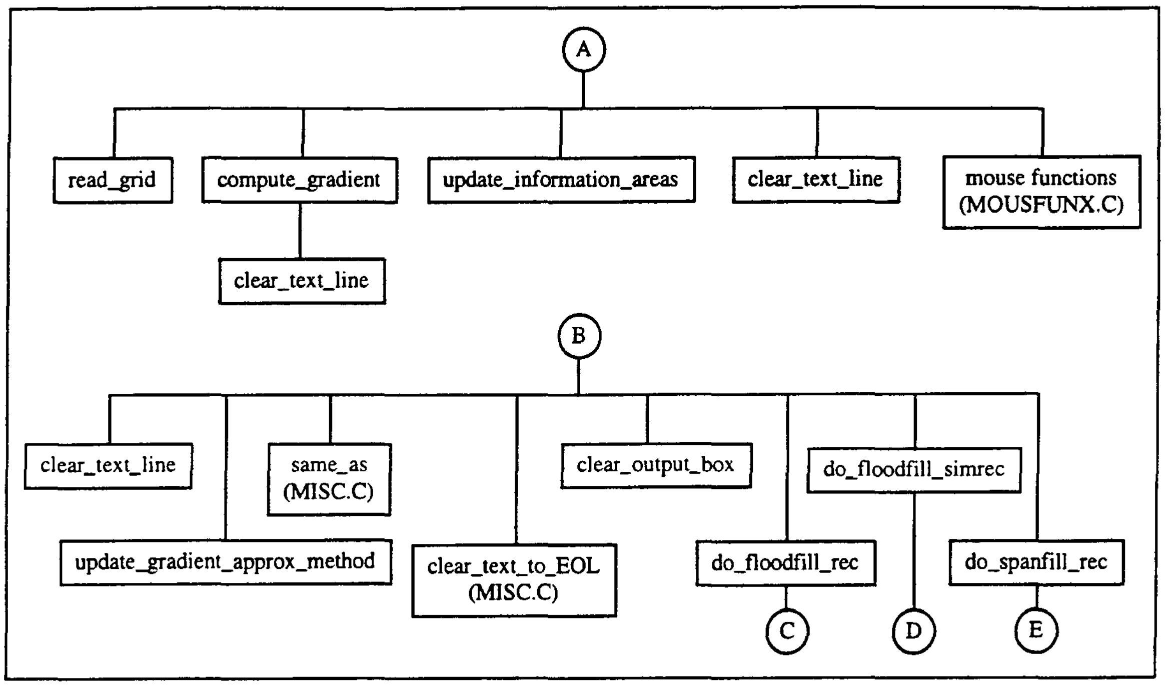


Figure 3.0-1b: GETCOORD Structure Chart (2 of 3)

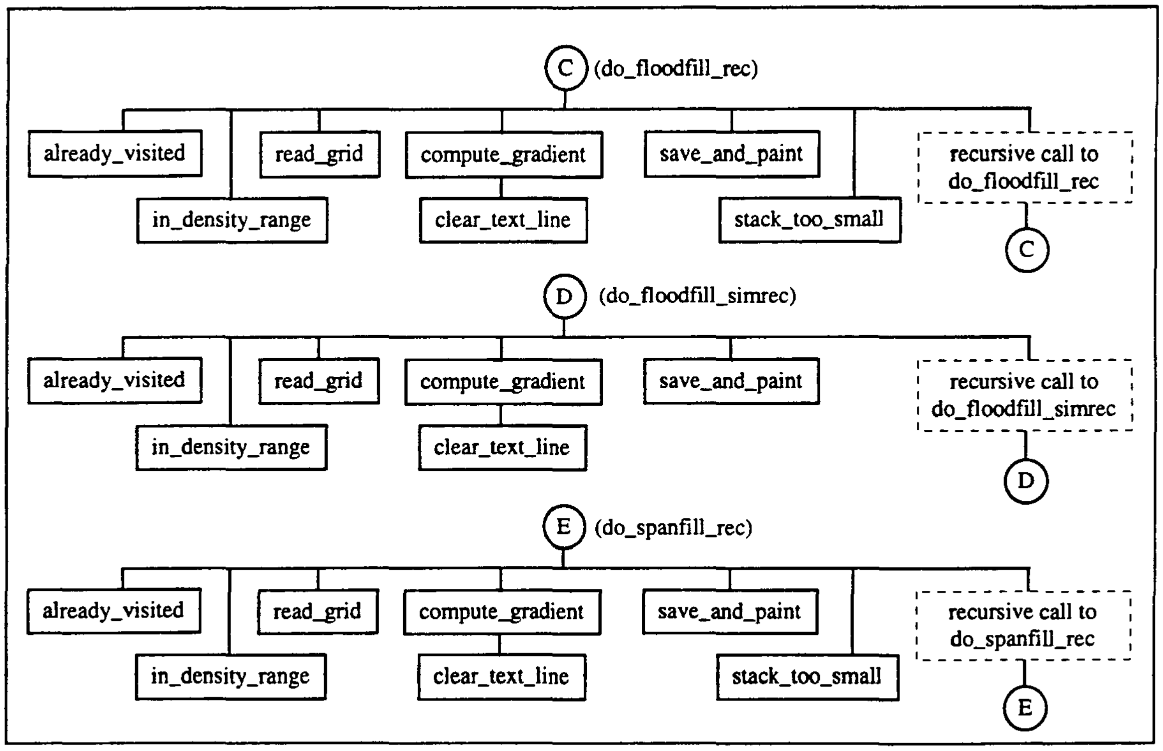


Figure 3.0-1c: GETCOORD Structure Chart (3 of 3)

3.1 Thresholding Methods

In this program, we permit two types of thresholding: evaluation by density-range and evaluation by density-gradient. Both types may be used together as well, i.e. a point must meet both criteria in order to be accorded membership in the region.

3.1.1 Density-range Thresholding

When performing segmentation, the user may specify a density range in terms of a lower density limit (LDL) and an upper density limit (UDL), between which a point's density value must fall in order to be considered a member of the region. This task is computationally easy and is performed before gradient approximation (section 3.1.2). The default LDL at the start of the program is 1; the default UDL is 4095. (If these values are modified, their new settings will be remembered between segmentation runs.)

The Magnetom does not report absolute hydrogen density directly in terms of any particular unit of measure, such as grams per cubic centimeter; rather, its numbers represent the relative intensity of RF signals given off by scanned tissue during the MRI scanning process. For convenience' sake, we may casually regard hydrogen density and RF signal intensity as equivalent.

3.1.2 Gradient-approximation Kernels

The user may select one of three different ways of computing the approximate gradient (∇) of the 3 x 3 matrix surrounding each pixel being considered for membership in the region of interest. In each case, wherever the tissue is homogeneous, we would expect the rate of change over the 3 x 3 area to be low; at the boundary between differing tissue types, we would expect a high gradient. The gradient-approximation methods will differ in their response to changes in the gradient over the 3 x 3 area; we will observe how the difference in response affects image quality.

To compute the gradient approximation is the most computationally expensive part of the segmentation process. To begin with, we must acquire the values of 9 pixels (in two dimensions), and then perform either integer or floating-point arithmetic on the values. Therefore, we only perform gradient-approximation after having determined that a pixel has not already been visited, and that its density is within the specified range. In the discussion which follows, we consider a 3 x 3 grid M, whose elements are numbered thus:

$$\begin{bmatrix} M_1 & M_2 & M_3 \\ M_4 & M_5 & M_6 \\ M_7 & M_8 & M_9 \end{bmatrix}$$

More discussion of these and other spatial filtering methods may be found in [Gonzalez92].

3.1.2.1 The Prewitt Kernels

The Prewitt kernels (or operators) are a pair of 3 x 3 matrices which are each multiplied by the 3 x 3 matrix of density values with pixel P at its center. We compute the partial derivative (or rate of change) in the vertical and horizontal directions, and then take the sum of the absolute values of the partial derivatives to compute an approximation of the gradient. The difference between the first and third rows approximates the derivative in the X direction; the difference between the first and third columns approximates the derivative in the Y direction [Gonzalez92]. The Prewitt kernels are:

$$K_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad K_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

An equivalent equation which summarizes the operation described above is:

$$\nabla M_5 = |(M_7 + M_8 + M_9) - (M_1 + M_2 + M_3)| + |(M_3 + M_6 + M_9) - (M_1 + M_4 + M_7)|$$

3.1.2.2 The Sobel Kernels

The Sobel kernels are similar in appearance and application to the Prewitt kernels, except that the middle term is doubled, as shown. According to [Gonzalez92], derivative filters enhance noise; the doubling of the center term in the Sobel operators works to provide a smoothing effect in the resulting image. The Sobel kernels are:

$$K_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad K_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The equivalent equation is:

$$\begin{aligned} \nabla M_5 = {} & |(M_7 + 2M_8 + M_9) - (M_1 + 2M_2 + M_3)| + \\ & |(M_3 + 2M_6 + M_9) - (M_1 + 2M_4 + M_7)| \end{aligned}$$

3.1.2.3 The Square-Sum-Root Method

The Square-Sum-Root (SSR) method is a simple method of approximating the gradient which does not implement kernels in the same way as the previous two methods. The SSR equation is:

$$\nabla M_5 = \sqrt{(M_2 - M_8)^2 + (M_4 - M_6)^2}$$

The SSR method involves floating-point arithmetic in the form of the square root functions, which requires more computer time than integer arithmetic.

3.2 Region-growing Methods

Upon reflection, it will be seen that the region-growing problem is merely a variation of the floodfilling problem. In floodfilling, we choose a seedpoint pixel within a region, and color the seedpoint, the seedpoint's neighboring pixels, and their neighbors, and so on *ad* *infinitum*, either until some boundary is reached (a boundary-defined region), or as long as a pixel's value is the same as that of the seedpoint (an interior-defined region). The decision to color any pixel is based on the pixel's present value; if it is not the boundary value, or if it is the same as that of the seedpoint, we color the pixel and proceed to consider its neighbors for coloring [Foley90].

The first three region-growing algorithms described here all operate in about the same way, and begin with a user-specified seedpoint, pixel P. During region-growing, we first check to ensure that P has not already been visited. If it has been visited, there is no point in spending time processing it again, and we continue with the recursive growth (section 3.2.1). Next, we ensure that the density of P is within the specified range. If not, we continue with the recursive growth. (We may effectively remove density-range from consideration by making the limits maximally broad, e.g. LDL = 1, UDL = 4095.)

We save the most computationally expensive stage for last, and compute the gradient ∇ for the 3 x 3 grid surrounding P. If ∇ is greater than or equal to the specified threshold T, we continue with the recursive growth. Otherwise, we save the density of P in the output buffer (P's coordinates being implicit in its position in the memory array), and then continue with the recursive growth.

3.2.1 Recursive 4-connected Region-growing

As pointed out, the region-growing process involves examining the neighbors of a pixel in a recursive fashion. In two dimensions, regions are said to be 4-connected if every two pixels can be joined by a sequence of pixels using only up, down, left, or right moves [Foley90]. Thus, after examining a pixel P for membership in the region to be segmented, we recursively examine P's neighbors to the north, south, east, and west.

This recursive approach possesses the virtue of simplicity of understanding and coding. However, stack space is needed during each recursive call to store the values of local variables, arguments passed to the recursive function, and the calling function's return address. (Each time a program makes a call to another procedure or function, it must store these values, arguments, and addresses on the stack in an area of memory called a stack frame.) For large regions, the many levels of recursion involved impose substantial requirements upon the computer's available memory for stack space. Recursive calls also may require more time to execute than an equivalent iterative approach due to the need for stack manipulation.

In order to assess the algorithm's requirements for stack frames, we maintain a counter which records the largest number of recursion levels for a particular run of the program. Before making a recursive call, the program ensures that there is sufficient stack space available; if not, the function returns to the calling function. Figure 3.2.1-1 displays an algorithmic flowchart for the recursive 4-connected region-growing process.

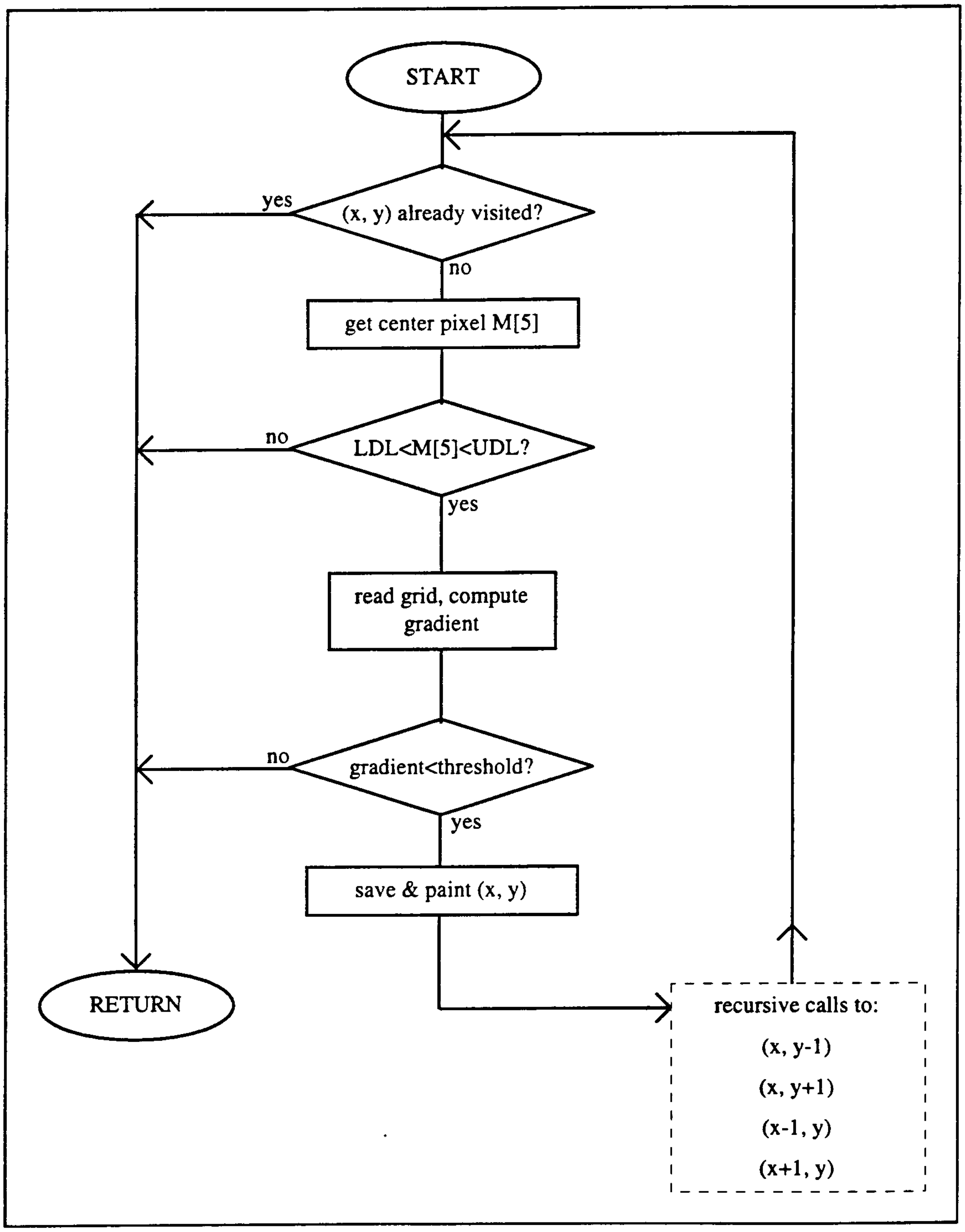


Figure 3.2.1-1: Recursive 4-connected Region-growing Algorithm

3.2.2 Recursive 8-connected Region-growing

In two dimensions, regions are said to be 8-connected if every two pixels can be joined by a sequence of pixels using only up, down, left, right, up-and-right, up-and-left, down-and-right, or down-and-left moves [Foley90]. Thus, after examining a pixel P for membership in the region to be segmented, we recursively examine P's neighbors to the north, south, east, and west, northwest, northeast, southwest, and southeast.

In all other respects, the 8-connected region-growing algorithm is identical to the 4-connected algorithm described in section 3.2.1. It is presented primarily to assess differences in image quality due to the additional degree of connectedness.

3.2.3 Iteration and Simulated Recursion; Stack Self-management

As noted, recursive methods require more time and memory space than equivalent iterative methods. In order to reduce the program's need for time and memory, we simulate recursion through an iterative implementation of the 4-connected recursive region-growing algorithm described above. [Tenenbaum90] presents some useful guidelines for maintaining one's own stack, and for simulating recursive calls for which there is no longer a requirement to preserve local variables. By maintaining the stack ourselves, we are able to use more efficiently the memory needed for arguments to the pseudo-recursive call, as well as the return-label used in lieu of a return address. The self-managed stack makes possible a greater number of pseudo-recursive calls for the same amount of physical memory, resulting in more complete growth of large regions for which many levels of recursion would be necessary. Figure 3.2.3-1 shows an algorithmic flowchart of the simulated-recursion 4-connected region-growing process.

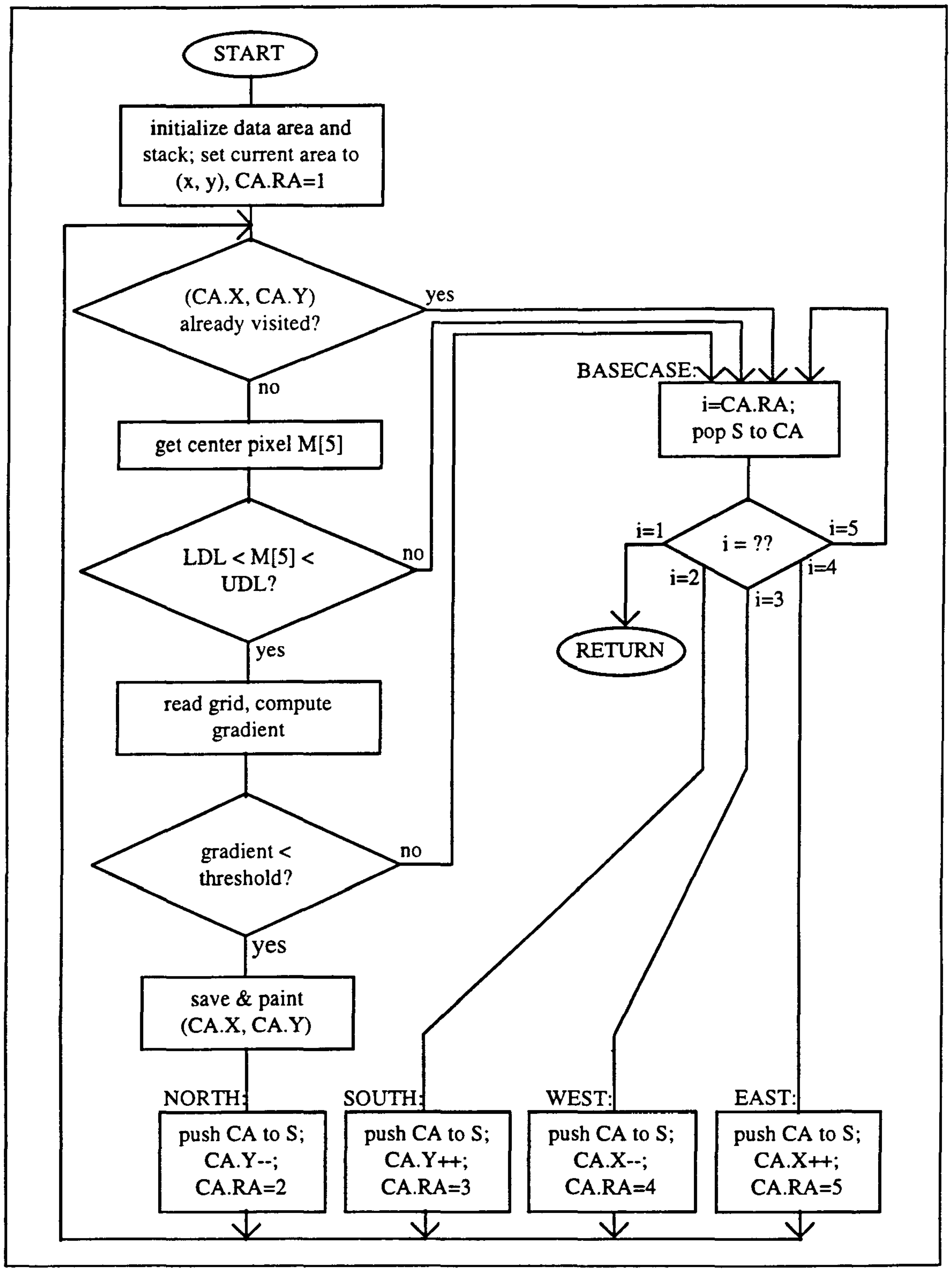

START
initialize data area and stack; set current area to (x, y), CA.RA=1
(CA.X, CA.Y) already visited?
yes
no
get center pixel M[5]
LDL < M[5] < UDL?
no
yes
read grid, compute gradient
gradient < threshold?
no
yes
save & paint (CA.X, CA.Y)
NORTH:
push CA to S; CA.Y--; CA.RA=2
SOUTH:
push CA to S; CA.Y++; CA.RA=3
WEST:
push CA to S; CA.X--; CA.RA=4
EAST:
push CA to S; CA.X++; CA.RA=5
BASECASE:
i=CA.RA; pop S to CA
i = ??
i=1
i=2
i=3
i=4
i=5
RETURN

Figure 3.2.3-1: Simulated-recursion 4-connected Region-growing Algorithm

3.2.4 A Spanfill Algorithm

The spanfilling algorithm is recursive, and starts (as usual) with a seedpoint, pixel P. This time, pixels to the left and right of P are examined for and granted membership in the region to be segmented. This horizontal row of member pixels is called the starting span.

We next examine each pixel in the row immediately above the starting span. When a new member pixel is found, we make a recursive call to the spanfill function, which saves and paints this new pixel, fills <u>its</u> span, examines the span above it, and so on. After the upward-recursive calls return, we make recursive calls to the spanfill function with respect to rows of pixels beneath the starting span.

Although recursive in nature, this algorithm is much less reliant on stack space than the n-way region-growing algorithms, because most of the pixel-coloring is done iteratively within each span; the function only recurses when it is necessary to look at the row above or below the current row.

Note that it is possible to have more than one span of member pixels in each row of pixels, each span being separated horizontally by non-member pixels. Note also that there are only 256 rows in the images acquired from the VDS. Although it is possible to conceive of degenerate cases where there are relatively many member spans on each row of pixels, it seems unlikely that one would encounter these cases very often in practice.

Most of the time, the number of levels of recursion needed by the spanfill algorithm is quite small. Figures 3.2.4-1a and 3.2.4-1b show an algorithmic flowchart of the recursive spanfill region-growing process.

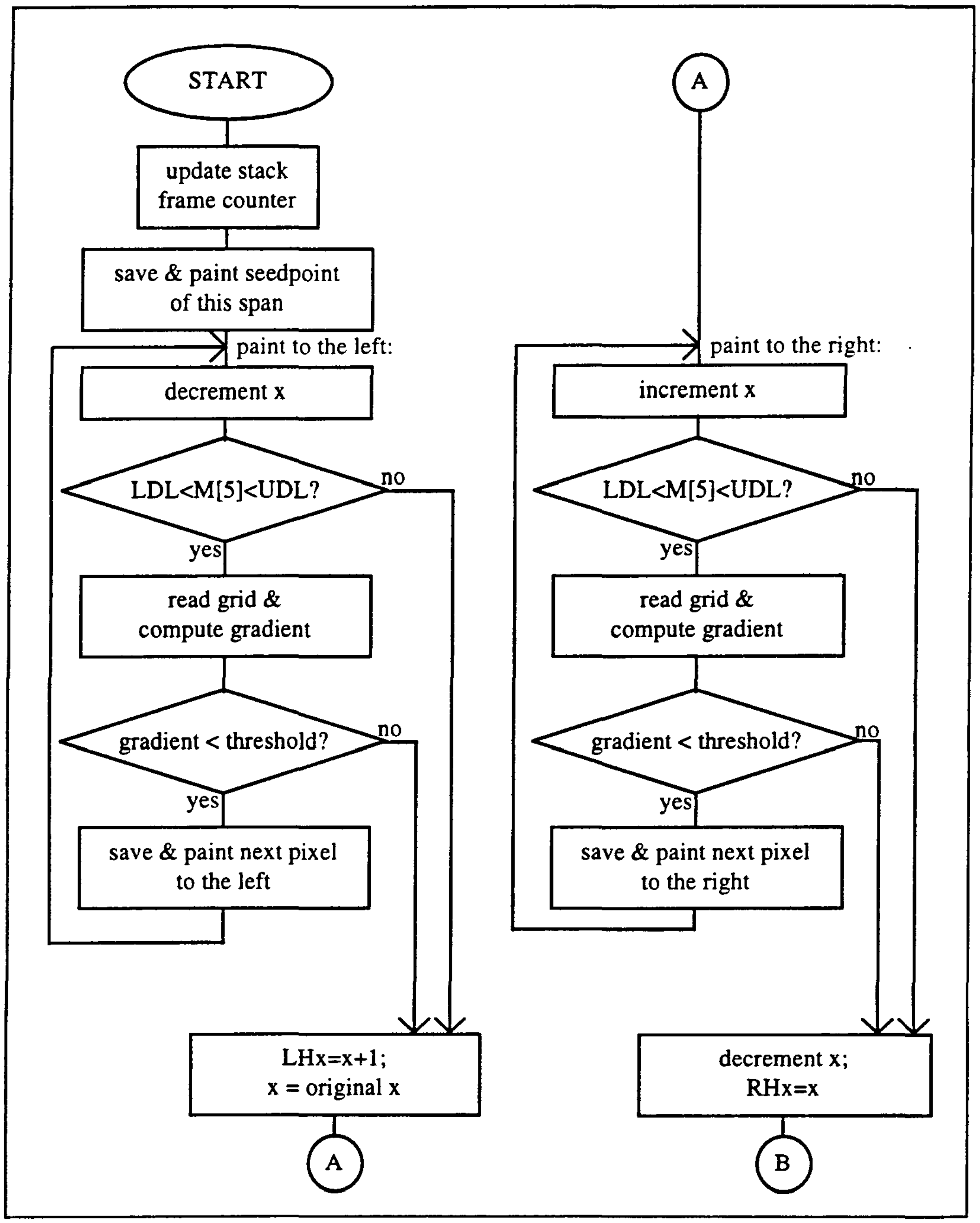


Figure 3.2.4-1a: Recursive Spanfill Region-growing Algorithm (1 of 2)

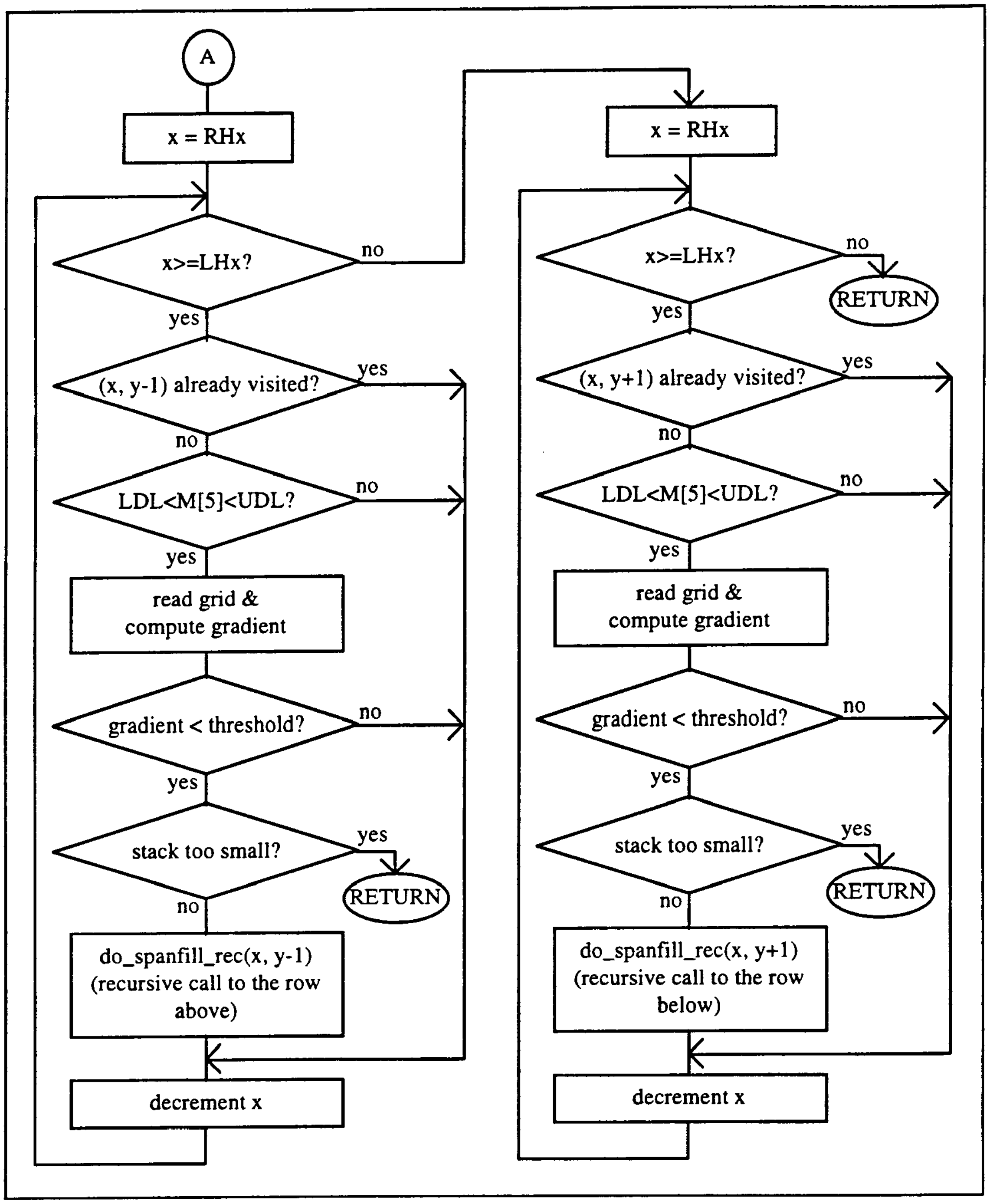

A
x = RHx
x>=LHx?
no
yes
(x, y-1) already visited?
yes
no
LDL<M[5]<UDL?
no
yes
read grid & compute gradient
gradient < threshold?
no
yes
stack too small?
yes
RETURN
no
do_spanfill_rec(x, y-1) (recursive call to the row above)
decrement x
x = RHx
x>=LHx?
no
RETURN
yes
(x, y+1) already visited?
yes
no
LDL<M[5]<UDL?
no
yes
read grid & compute gradient
gradient < threshold?
no
yes
stack too small?
yes
RETURN
no
do_spanfill_rec(x, y+1) (recursive call to the row below)
decrement x

Figure 3.2.4-1b: Recursive Spanfill Region-growing Algorithm (2 of 2)

3.3 Data Representation and Storage Methods

3.3.1 Data Structures

We discuss two data structures used to store two-dimensional MRI segmented data: spatial occupancy enumeration (SOE) and the quadtree. We compare them with regard for ease of coding, ease of data manipulation, speed and memory conservation.

3.3.1.1 Spatial Occupancy Enumeration Using 2-D Arrays

Spatial occupancy enumeration (SOE) is a form of solid-figure representation in which the solid is decomposed into identical cells (or cells possessing similar characteristics, such as the same density- or density-gradient range) arranged in a fixed, regular grid (e.g. 256 x 256) [Foley90]. In two dimensions, these cells are called pixels (picture elements); in three dimensions, they are called voxels (volume elements). We represent an object by deciding which cells are occupied, and which cells are not. Simple to implement and manipulate, SOE may be used to organize data both in memory and in disk files, and is often used in biomedical applications.

In the GETCOORD program, we have established two 256 x 256 square arrays for the input and output slice-image buffers. In order to improve efficiency, we use the output buffer's initial zero values as an indicator of whether a particular pixel was already visited during region growth.

For a resolution of *n* voxels in two dimensions, we may need up to n^2 cells to represent an object. This may provide only a rough approximation of the object; however, we are already limited to this approximation in MRI work because of the nature of the original VDS and its manner of generation.

We perform three basic operations in working with SOE: defining the memory array, writing to the array, and reading from the array. We may define the size of the array either at compile time or at run time. For an image of *n* x *n* pixels, and each pixel requiring *m* bytes to store its value, the size of the array must be *n* x *n* x *m* bytes. A 256 x 256 array of 2-byte-integer data would therefore require an array of 131,072 bytes.

To access any particular pixel in the array for either reading or writing, we need only calculate the byte-offset from the beginning of the array. If the coordinates of the pixel are (x, y), and the number of rows in the array is R, then the byte-offset is

$$x + (y * R)$$

and the value of the pixel P at (x, y) is

$$P = \text{array}[x + (y * R)].$$

Similar remarks hold for accessing a pixel in a random disk file; we may use file-pointer-positioning functions to access specific pixels in an already existing file. However, while the size of the data type is usually already figured in by the compiler when accessing a memory array, this must be explicitly done by the program when using byte-offsets on a file. Then, the byte-offset is

$$(x + (y * R)) * \text{sizeof(data type)}$$

where sizeof() is an operator which returns the size (in bytes) of the data type.

The SOE approach using a memory array is very easy to implement, since the data structure is topologically congruent to the way that the data occur in physical reality. Manipulation of data is simple and fast, since to access any particular pixel requires the computation (involving integer arithmetic only) of a single array address. The only disadvantage of SOE is that, for a scene in which the segmented region is very small in relation to the rest of the scene, most of the array will be used to store zeros, representing the absence of segmented data. Such an array is called a sparse matrix, and might be considered wasteful of memory space. In such cases, we may wish to explore other data structures which are designed <u>not</u> to store null- or zero-data. Such structures include linked lists and trees.

3.3.1.2 Linked Lists and Trees

A linked list is a list of records in which each record contains, in addition to its data, a field used to hold a pointer to the next record in the list. A tree is a data structure whose root node contains, in addition to its data, links to two or more child nodes. Each child node contains data plus links to other child nodes, and so forth in recursive fashion. Linked lists and trees may be created, enlarged, and reduced dynamically (that is to say, at runtime) as opposed to array structures, whose size is fixed at the time the program is compiled.

By and large, lists and trees are more difficult and time-consuming to create and maintain than are ordinary arrays. Each time a new node is required, memory must be allocated and links established. Likewise, when a node is no longer needed, its memory must be deallocated and returned to the operating system. Further, lists and trees are not randomly addressable; whenever a node is to be accessed, the list or tree must be traversed in linear or some other order until the desired node is found.

As mentioned, for a scene in which the segmented region is very small in relation to the rest of the scene, most of the array will be used to store zeros in order to represent pixels which were not selected during the segmentation process. This memory space could be regarded as wasted, but is an unavoidable overhead cost to employing a fixed-size array. The use of lists and trees to hold segmented data represents an effort to reduce this wasted space, but their effectiveness is limited because of the extra memory required for each node in the list or tree to record pointers to its children and parent nodes.

One form of tree structure used for the storage of two-dimensional image data is called the quadtree. A quadtree divides a scene into four quadrants. The quadtree's root node points to each of the four top-level quadrants, which are classified depending upon whether the quadrant is full (i.e. all of the quadrant's pixels values meet the segmentation criteria), partially full (only some of the pixels meet the segmentation criteria), or empty (none of the pixels meet the segmentation criteria). Each quadrant is recursively divided into sub-quadrants and analyzed, down to the pixel level, where no further division is possible. (When working with three dimensional scenes, we may use a natural extension of the quadtree notion called the octree, which divides the scene into eight octants.)[1]

If a region's homogeneity is highly concentrated (i.e. many adjacent pixels meeting the criteria for segmentation), then, in terms of memory requirements, the quadtree can provide a fairly efficient method of data storage. In general, the amount of memory required is a function of the resolution (number of levels in the quadtree), the image size, and the region's position in the grid [Samet90A].

[1] Quadtrees are frequently used in the transmission of graphical data, where the interest is in enabling the receiving party to view the image at progressively better levels of resolution, and to terminate the transmission if and when it is decided that the picture is unwanted. At first, a brief amount of time is used to transmit a crude picture at low resolution. Then a longer amount of time is used to transmit a better-quality picture at a higher resolution, and so on, until, at the end of the transmission, the picture is reconstructed perfectly. This process may be interrupted at any time, and is useful for browsing operations [Samet90B].

If the region represented by a quadtree is highly convoluted (relatively few adjacent member pixels), then the efficiency of the quadtree decreases. Consider a scene whose pixels may have a value of either 0 or 1. In the worst case, that of a checkerboard pattern of dimensions 2^n x 2^n, then the number of nodes required to completely represent the region is

$$\sum_{k=0}^{n} 4^k = 4^0 + 4^1 + 4^2 \cong \left(\frac{4}{3}\right) * 16 = \left(\frac{4}{3}\right) 4^n$$

Note that each node in the quadtree must store certain data, for example:

- the value of this pixel, if a leaf node (2 bytes)
- a status field (full, partially full, or empty) (1 byte)
- the upper-left coordinates of the quadrant (2 bytes)
- the size of the quadrant's side, in pixels (1 byte)
- a pointer to the node's parent (2 bytes)
- four pointers to each of the node's children (4 x 2 bytes)

By this reckoning, then, each node would need to be allocated 16 bytes of storage, and the 2^n x 2^n region would require (in the worst case)

$$16 * \left(\frac{4}{3}\right) 4^n$$

bytes of storage.

In the best case (where all the pixels of the region are members), only one node (at 16 bytes) would be required to completely represent the region. However, these remarks hold only for two-valued (black-and-white) images. If a scene's pixels are multi-valued (e.g. grayshade values or density data), pixels with a relatively wide range of values may be considered members of a segmented region and yet may not be able to be grouped efficiently in a quadtree because their values are not exactly the same. In other words, the quadtree assigns nodes on the basis of a two-state condition: is the pixel a member of the region or not? Whereas the segmentation algorithm usually assigns membership less restrictively, admitting a wide range of values.

Note that a fixed-size array of dimensions 2^n x 2^n would require only $2 * 2^{2n}$ bytes (at 2 bytes per array element), regardless of the condition of the region, and regardless of the conditions for segmentation; in other words, the best and worst cases are the same.

3.3.2 File Formats

The binary image data stored in the RAW and PostScript formats is organized by spatial occupancy enumeration in row-major order. That is, the values in the first *n*-element row of the memory array are stored in the first *n* positions of the file; the values from the second row of the array are stored in the next *n* positions of the file; an so on.

3.3.2.1 RAW Format

The RAW file format was originally used for storing 256-shade grayscale data. As implemented in the MIDTERM image-processing program [Bell94A], an image of dimensions 256 x 256 pixels could be stored in a file of 65,536 bytes, each byte being capable of representing a number in the range 0 to 255. There is no particular reason why the RAW format could not be used for storing any kind of data (e.g. density levels), provided that the range of numbers could fit into a single byte.

When GETCOORD reads data from the slice-file, it must quantize (scale) the data from the range 0-4095 to the range 0-15, in order for the VGA to be able to display the image. Since MIDTERM expects from a RAW file a value in the range 0-255, GETCOORD must re-quantize the screen-buffer gray-scale values from the range 0-15 to the range 0-255 before storing the values in the RAW file.

3.3.2.2 Encapsulated PostScript Format

GETCOORD can save both the segmented image and a copy of the original image to an encapsulated PostScript (EPS) file. All of the information relating to the active settings is saved: filenames, x-y-z coordinates, thresholding and floodfilling methods, etc. The EPS format is quite similar to ordinary PostScript, but permits the image to be imported into many popular word-processing and page-layout programs.[2]

3.4 Image Presentation

Since MRI data represents relative RF signal intensities (which are, in turn, correlated to hydrogen density), some transformations are necessary in order to obtain a useful display on a computer monitor. We discuss two techniques below.

[2] In order to convert an ordinary PostScript file to EPS, using a text editor, simply add the following lines to the beginning of the PostScript file:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: llx lly urx ury
%%Creator: cccccc
%%Title: tttttt
%%CreationDate: mmddyy
```

where llx and lly are the lower-left x and y coordinates and urx and ury are the upper-right x and y coordinates of the image's bounding box. The Creator, Title, and CreationDate fields are non-printing optional comments which will appear in the bounding box on the screen (in lieu of a thumbnail picture, whose format is not included here) [Holzgang92].

3.4.1 Histogram Stretching

The GETCOORD program records the minimum and maximum values present in the slice-image. It also permits the user to specify lower and upper bounds to be used in stretching or compressing the histogram of intensities in the slice-image. When the histogram is modified, the values in the slice-image are adjusted so as to scale the entire range of values to a different range, resulting in improved contrast and a more usable image. For a system of 256 grayshades, the value 0 equals black and 255 equals white, with progressively lighter shades of gray in between. If most of a picture's pixel values are low, then the picture will be dark. Stretching the histogram can lighten the picture and make more detail visible.

3.4.2 Color Quantization

Color quantization is the process of scaling a range of pixel color values to a range which is more suited to the display device being used. The range of input values from the 3DHEAD VDS has been determined to be from -128 to 3955. The range of valid grayshade values for a standard VGA adapter is 0 (black) to 15 (high-intensity white). Therefore, the input values must be scaled and sorted into 16 "buckets" corresponding to each of the available VGA grayshades. It may be necessary to perform histogram stretching and quantization together in order to obtain a usable image.

CHAPTER 4

TESTING AND RESULTS IN TWO DIMENSIONS

In this chapter, we discuss the tests performed with the 2-D segmentation program, focusing on these particular areas:

- image quality
- choice of seedpoint
- density ranges
- density-gradient threshold
- gradient-approximation methods
- region-growing methods
- stack frame and memory requirements
- image resolution
- run time needed to create an image

The notion of "image quality" is subjective and deserves some elaboration. To some extent, image quality is constrained by the resolution of the original dataset. In two dimensions, the quality of an image is a reflection of how well a particular structure is able to be segmented, that is, separated from neighboring tissue. Does the region have clear boundaries? Does it include what it is "supposed to"? Does it exclude other undesired areas? Is the image pocked or pitted to a degree which is out of proportion to the fundamental nature of the object represented?

Image quality certainly depends upon the completeness of the image. This completeness is related to computer resource requirements in that, the more pixels which are visited, the more time and stack frames will be needed for the final image to be generated. But more resource commitments do not necessarily result in a better image.

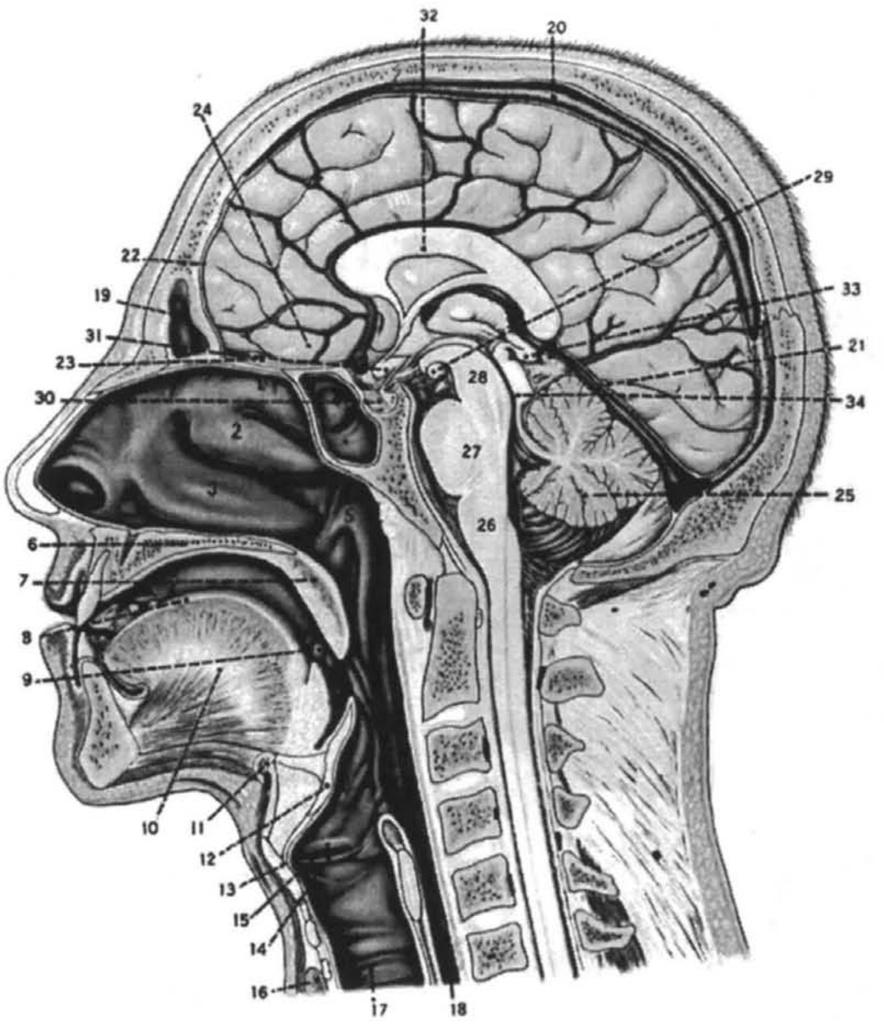
4.1 Benchmark Seedpoints

Specific locations within the human head were used as "benchmark seedpoints" for the segmentation process. Taken from the median sagittal section (3DHEAD slice #54) of the head, these benchmark points were chosen because they represent (in the opinion of the author) distinct and significant structures which would lend themselves well to segmentation. Names of the skull's internal structures are taken from [Frohse61] (figure 4.1-1). Figure 4.1-2 and table 4.1-1 may be used together to locate the benchmark seedpoints.

From these seedpoints, "canonical" images of segmented structures were derived (see section 4.3). Although the notion of what is canonical is a matter of opinion, these images will provide a useful means of comparison with other segmented images.

structure (fig.)	(x, y)	ref.	T	LDL	UDL
corpus callosum (4.3-1)	(120, 103)	A	1000	1200	2000
pons (4.3-2)	(155, 140)	B	800	1100	1500
worm of cerebellum (4.3-3)	(181, 147)	C	1800	900	1800
cerebrum (4.3-4)	(151, 75)	D	900	900	2000
scalp (4.3-5)	(199, 55)	E	8000	1200	2800
superior sagittal sinus (4.3-6)	(200, 75)	F	1400	100	900
brain (cerebrum, cerebellum, stem) (4.3-7)	(179, 142)	G	1500	500	2000
head (4.3-8)	(127, 127)	H	10000	200	3000
(all Prewitt segmentation) (all Recursive 4-connected, except fig. 4.3-8, Iterative)					

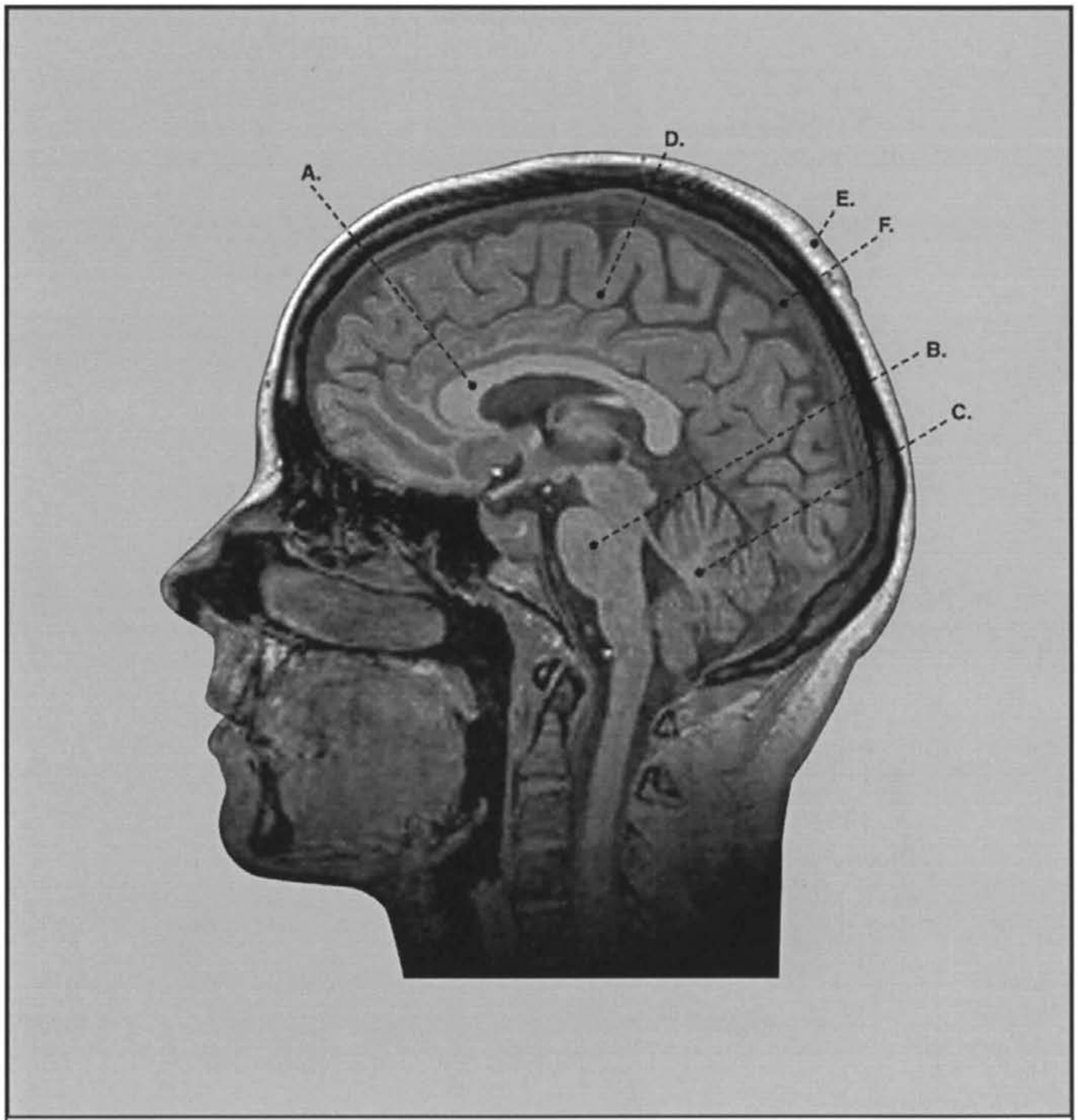
Table 4.1-1: Benchmark Seedpoints



THE HEAD
MEDIAN SECTION

1. Superior turbinated bone *(Concha nasalis superior)*
2. Middle turbinated bone *(Concha nasalis media)*
3. Inferior turbinated bone *(Concha nasalis inferior)*
4. Sphenoidal sinus *(sinus sphenoidalis)*
5. Tubal protruberance *(Torus tubarius)*
6. Hard palate *(Palatum durum)*
7. Soft palate *(Palatum molle)*
8. Back of tongue *(Dorsum linguae)*
9. Tonsil *(Tonsilla palatina)*
10. Genioglossal muscle *(M. genioglossus)*
11. Hyoid bone *(Os hyoideum)*
12. Epiglottis *(Epiglottis)*
13. Thyroid cartilage *(Cartilago thyreoidea)*
14. Vocal fold *(Plica vocalis)*
15. Ventricular fold *(Plicca vertricularis)*
16. Thyroid gland *(Glandula thyreoidea)*
17. Windpipe *(Trachea)*
18. Gullet *(Oesophagus)*
19. Frontal sinus *(Sinus frontalis)*
20. Superior sagittal sinus *(sinus sagittalus superior)*
21. Strainght sinus *(Sinus rectus)*
22. Dura mater *(Dura mater)*
23. Olfactory bulb *(Bulbus olfactorius)*
24. Frontal lobe *(Lobus frontalis superior)*
25. Worm of cerebellum *(Vermis cerebelli)*
26. Oblong medulla *(Medulla oblongata)*
27. Pons (Pons)
28. Leg of cerebellum *(Crus cerebri)*
29. Mamillary body *(Corpus mamillare)*
30. Pituitary body *(Hypophysis)*
31. Optic chiasma *(Chiasma nervi optici)*
32. Great commissure *(Corpus callosum)*
33. Pineal body *(Corpus pineale)*
34. Quadrigeminal bodies *(Corpora quadrigemina)*

Figure 4.1-1: Anatomical Illustration of the Head (Median Section)



A. Corpus Callosum
B. Pons
C. Worm of Cerebellum
D. Cerebrum
E. Scalp
F. Superior Sagittal Sinus

Figure 4.1-2: Benchmark Seedpoint Locations

4.2 Factors Affecting Segmentation Results

4.2.1 The Seedpoint

The choice of seedpoint is important in performing a successful segmentation. If a seedpoint is chosen which lies on or near the boundary between two types of tissue, then the gradient will be very high there, and the segmentation will immediately fail the gradient-threshold test. Likewise, if the density of the chosen seedpoint is outside of the specified range, then the density-range test will fail.

In order to address this possibility, the program displays during the seedpoint-selection process a 3 x 3 matrix of density values surrounding the pixel underneath the mouse-cursor, as well as the gradient and density at that pixel. This information is updated dynamically as the mouse-cursor is moved, allowing the user to "explore" a region of interest before committing to a particular seedpoint.

4.2.2 The Density Range

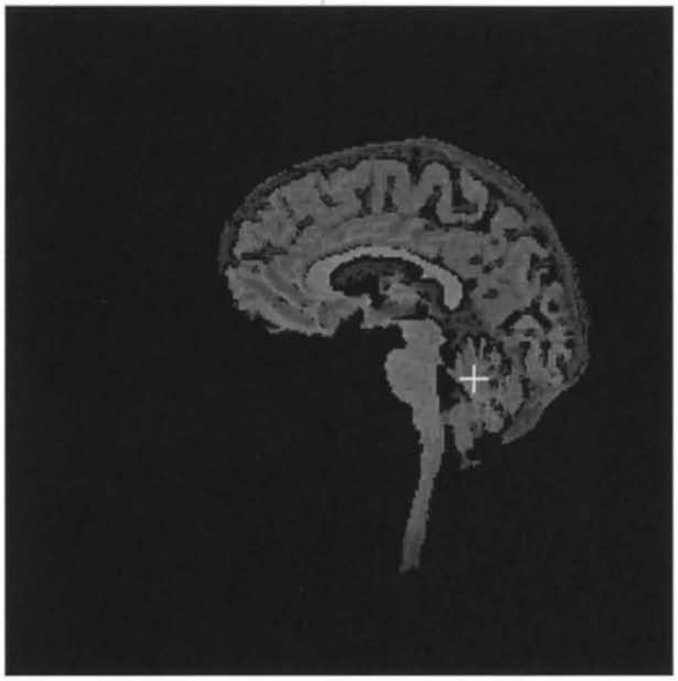
As the density range limits LDL and UDL are made smaller, fewer points will be considered part of the region to be segmented. Some data may be lost, but the degree of separation between different types of tissue may be improved. Using the "exploration" feature of the program, the user may observe the range of densities in the area he wishes to segment, and will be able to set LDL and UDL accordingly.

To observe the effects of changes in density range, we use the canonical image of the brain (figure 4.2.2-1) as a basis for comparison. (The settings for this and the other canonical images were arrived at through a process of intelligent trial-and-error involving such changes as we describe here.) The seedpoint for this image lies in the worm of the cerebellum, at coordinates (179,142). The density at this point is 1370; we know, then, that the LDL must be set to some number less than 1370, and the UDL to some greater number, for any region-growing to take place at all. (As a rule, in the following tests, we work with this same seedpoint, using iterative region-growing, and varying from the canonical only those settings whose behavior we are studying.)

Holding the UDL at the maximum of 4095, we begin by varying the LDL from 100 to 1300 in increments of 100. As table 4.2.2-1 shows, as the LDL in gradually increased, fewer and fewer pixels are accumulated into the region grown. Figure 4.2.2-2 shows how excluding pixels of density 500-800 results in an image of the cerebrum which is eroded and less distinct. Figure 4.2.2-3 demonstrates an interesting phenomenon: as the LDL is increased by just 100, there is a drastic decline in the area of the grown region. Apparently there is a "bridge" of pixels with density 800-900 which connect the worm of the cerebellum with the rest of the brain; when this bridge is eliminated, region-growing is inhibited.

Looking at the upper side of the density range, as the UDL is increased upwards from 1370 (and the LDL is held constant at 500), some pixels are added to the image, but ultimately the growth of the region is constrained by the density gradient threshold.

Output filename: FIG4221.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Iterative
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 2957
Pixels accumulated: 9137

Input filename: SAG-54.SLC

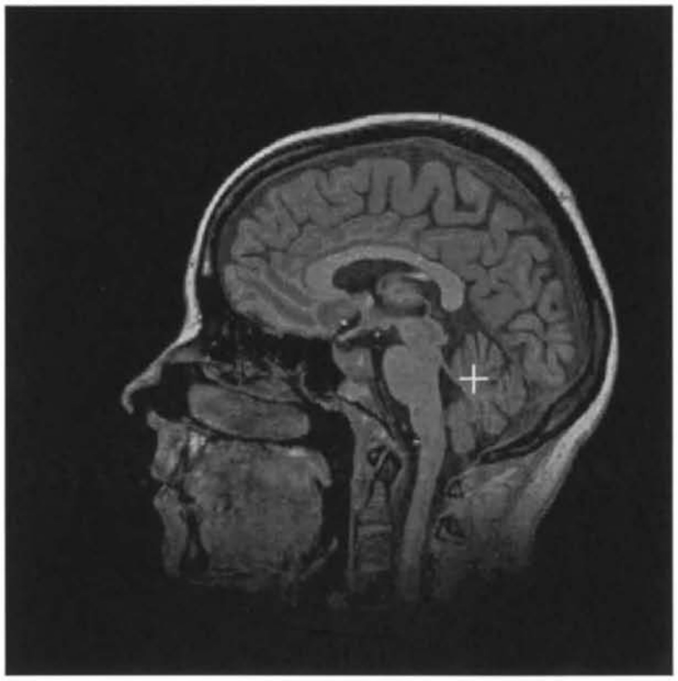
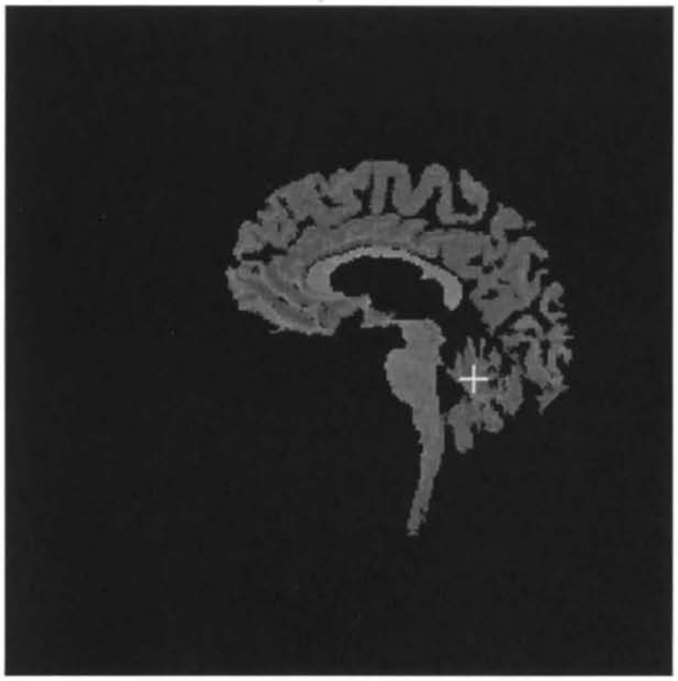


Figure 4.2.2-1: Canonical Image of the Brain

Output filename: FIG4222.EPS



GETCOORD - Bill Bell - Fall 1996

```
x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Iterative
LDL: 800
UDL: 4095
Threshold: 1500
Stack frames used: 2139
Pixels accumulated: 6605
```

Input filename: SAG-54.SLC

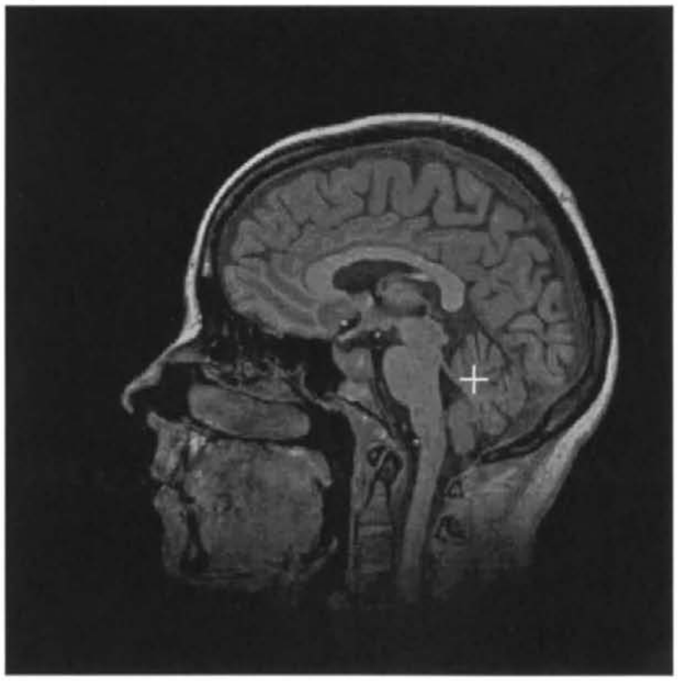
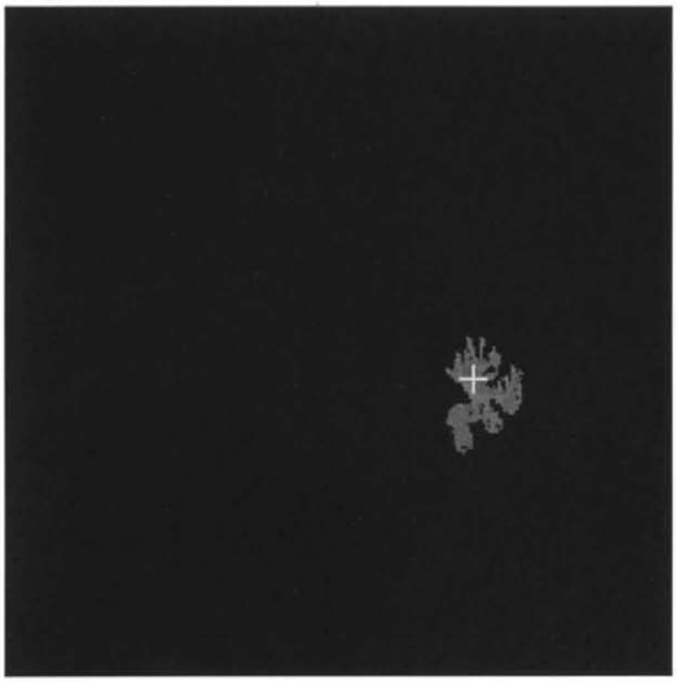


Figure 4.2.2-2: The Canonical Brain, LDL Increased to 800

Output filename: FIG4223.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Iterative
LDL: 900
UDL: 4095
Threshold: 1500
Stack frames used: 218
Pixels accumulated: 611

Input filename: SAG-54.SLC

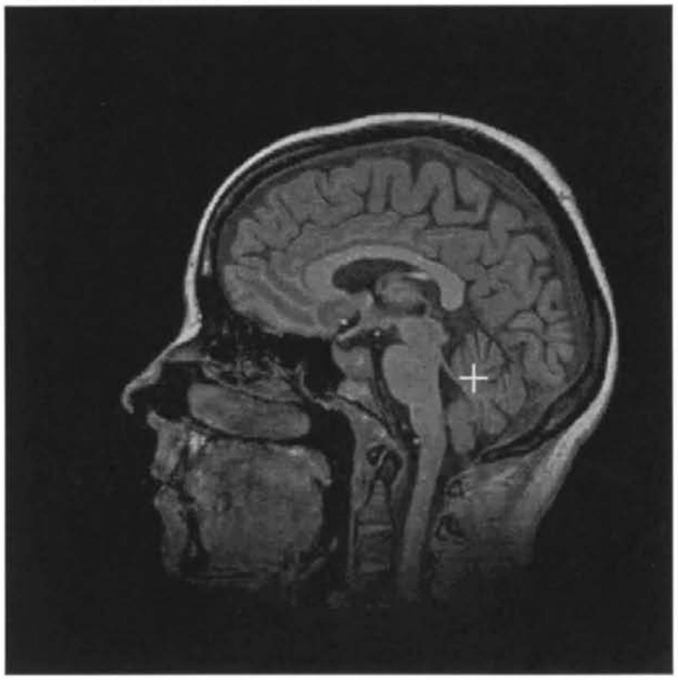


Figure 4.2.2-3: The Canonical Brain, LDL Increased to 900

LDL	UDL	recursive calls	pixels accumulated	remarks (image quality, structures included)
100	4095	4562	20895	brain, frontal and sagittal sinuses
200	4095	3113	14108	brain, superior sagittal sinus, back of neck
300	4095	3325	12578	brain, superior sagittal sinus, back of neck
400	4095	3191	10976	brain, superior sagittal sinus, back of neck
500	4095	2957	9137	brain only
600	4095	2601	7986	brain, folds of cerebrum eroded
700	4095	2512	7120	brain, folds of cerebrum eroded
800	4095	2139	6605	brain, cerebrum eroded, no pineal
900	4095	218	611	only worm of cerebellum shown
1000	4095	108	515	eroded worm of cerebellum
1100	4095	88	232	eroded worm of cerebellum
1200	4095	39	79	eroded worm of cerebellum (almost gone)
1300	4095	20	39	eroded worm of cerebellum (almost gone)
1369	4095	2	1	one pixel
4095	4095	1	0	end of chart

Table 4.2.2-1: Effects of Adjustments to Lower Density Limit

LDL	UDL	recursive calls	pixels accumulated	remarks (image quality, structures included)
500	1371	1982	8553	brain, corpus callosum missing
500	1400	3275	8664	brain, corpus callosum missing
500	1500	2911	8880	brain, corpus callosum begins to appear
500	1600	2957	9133	brain complete
500	1700	2957	9137	brain complete
500	1800	2957	9137	brain complete
500	1900	2957	9137	brain complete
500	2000	2957	9137	brain complete
500	2100	2957	9137	brain complete
500	2200	2957	9137	brain complete
500	2300	2957	9137	brain complete
500	2400	2957	9137	brain complete
500	2500	2957	9137	brain complete
500	4095	2957	9137	end of chart

Table 4.2.2-2: Effects of Adjustments to Upper Density Limit

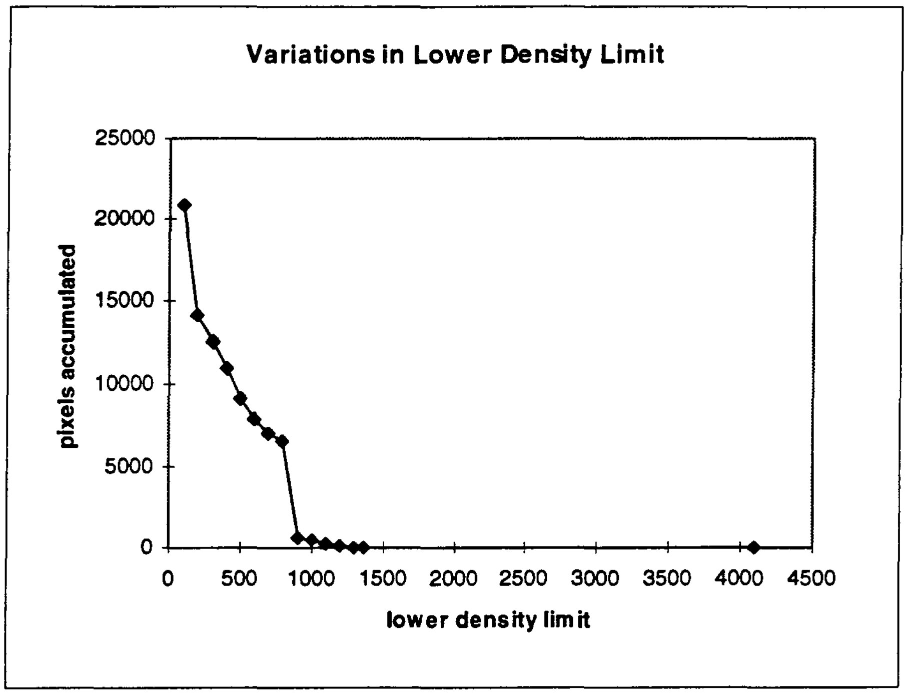


Figure 4.2.2-4: Effects of Adjustments to Lower Density Limit

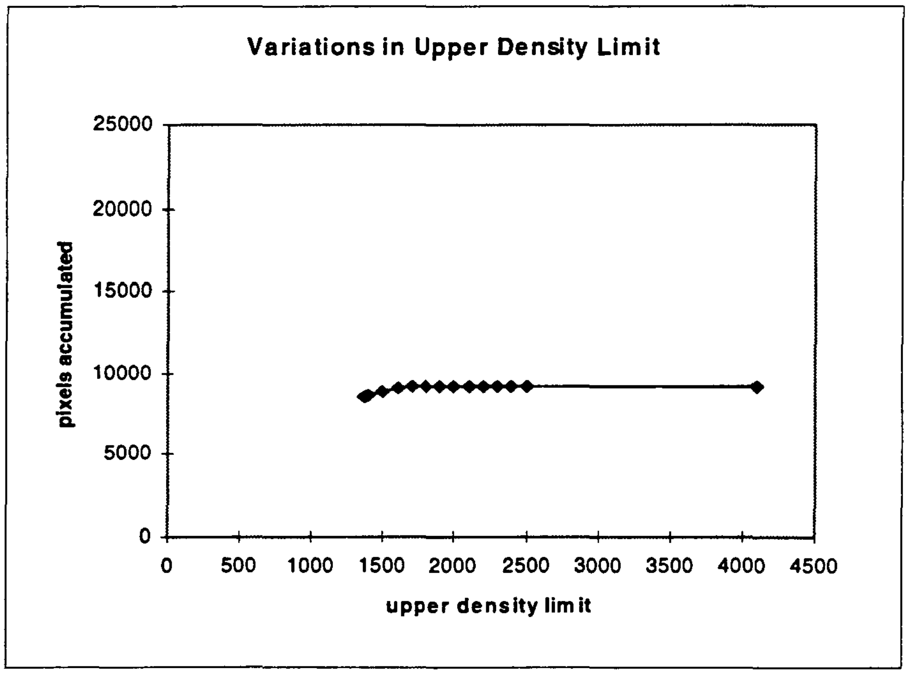


Figure 4.2.2-5: Effects of Adjustments to Upper Density Limit

4.2.3 The Gradient Threshold

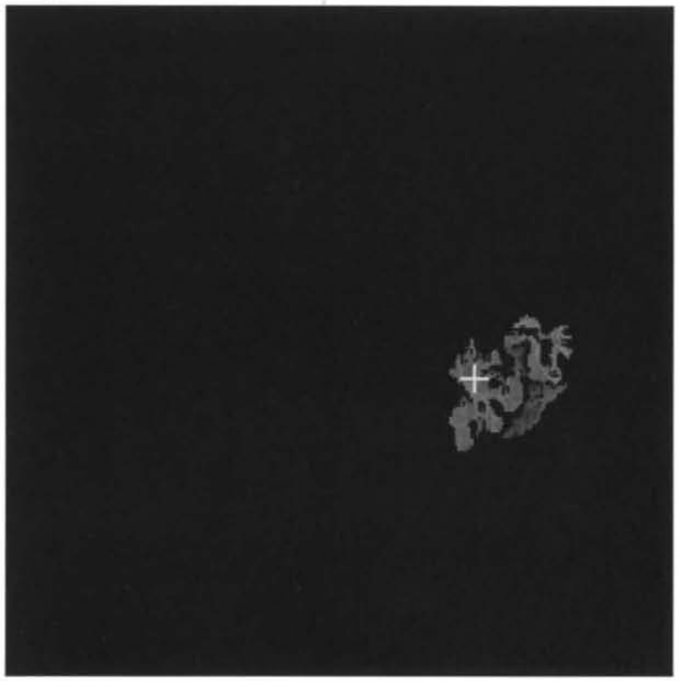
As the gradient-threshold T is made smaller, fewer points will be considered part of the region to be segmented. Some data may be lost, but the degree of separation between different types of tissue may be improved. Using the "exploration" feature of the program, the user may observe the range of gradients in the area he wishes to segment. Keeping in mind the fact that the gradient is high in the area of the boundary between tissue types, the user can determine (approximately) the largest threshold value which will acquire member pixels of the region to be segmented, without impairing the program's ability to accurately differentiate between tissue types.

Segmentation by gradient-threshold may be complicated where the seedpoint is located in very porous tissue which has an inherently high gradient, such as bone or lung tissue. In such cases, one may be better off selecting a very high gradient-threshold (INT_MAX, in this case 32,767) and relying upon density-range alone for results. One must be sure that the density-range selected is that of the porous tissue itself, not of whatever substance lies inside the pores.

The (Prewitt) gradient approximation at the seedpoint is 597. Again using the canonical brain image for comparison (figure 4.2.3-1), we hold the LDL and UDL steady while varying the density-gradient threshold in increments. As table 4.2.3-1 shows, as the threshold is increased, this relaxation in the criteria for region membership results in the addition of more pixels. Figures 4.2.3-2 and 4.2.3-3 demonstrate a drastic accumulation of pixels when the threshold is increased from 1000 to 1100; the higher threshold was needed to permit the addition of pixels representing the cerebrum to those of the worm of the cerebellum, wherein the seedpoint lies. If the threshold is permitted to grow too large, entirely different structures may be added to the region. Ultimately, region growth is constrained by the density range limits.

Output filename: FIG4232.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [179, 142, 54]

Gradient-approximation method:

Prewitt

Region-growing method:

Iterative

LDL: 500

UDL: 2000

Threshold: 1000

Stack frames used: 395

Pixels accumulated: 1042

Input filename: SAG-54.SLC

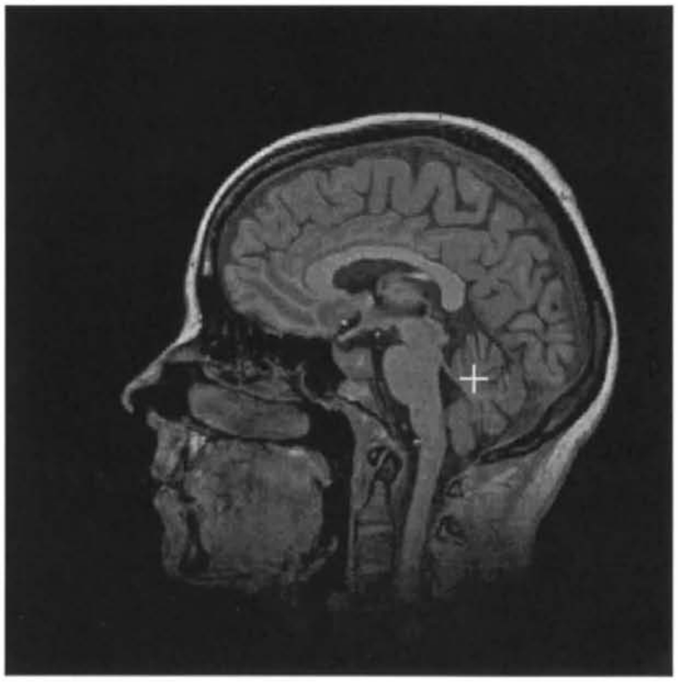
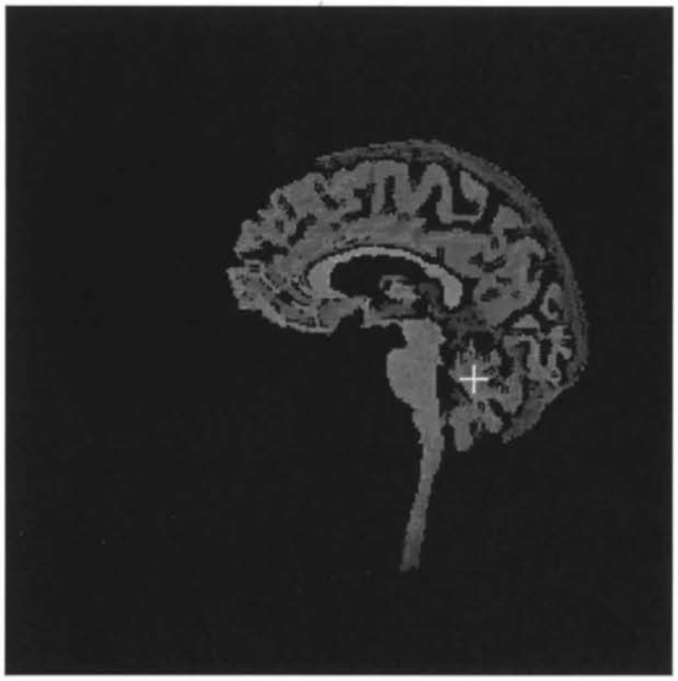


Figure 4.2.3-2: The Canonical Brain, Density Gradient Threshold Reduced to 1000

Output filename: FIG4233.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Iterative
LDL: 500
UDL: 2000
Threshold: 1100
Stack frames used: 2280
Pixels accumulated: 7416

Input filename: SAG-54.SLC

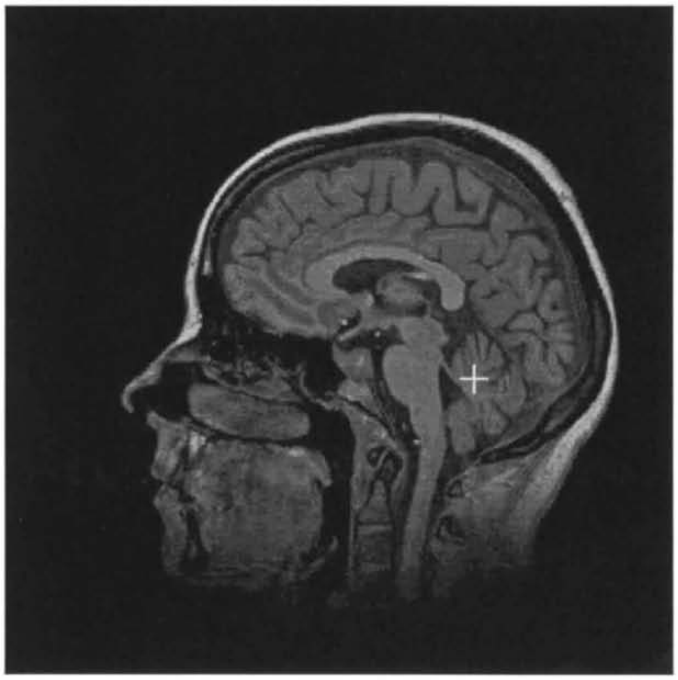


Figure 4.2.3-3: The Canonical Brain, Density Gradient Threshold Reduced to 1100

threshold	recursive calls	pixels accumulated	remarks (image quality, structures included)
600	105	196	eroded worm of cerebellum
700	131	259	eroded worm of cerebellum
800	274	470	eroded worm of cerebellum, some cerebrum
900	334	804	eroded worm of cerebellum, some cerebrum
1000	395	1042	complete worm of cerebellum, some cerebrum
1100	2280	7416	brain, eroded
1200	2336	7916	brain, eroded
1300	2734	8369	brain, eroded
1400	3231	8760	brain, eroded
1500	2957	9137	the canonical image
1600	3492	10561	brain, back of neck
1700	2914	10882	brain, back of neck
1800	3238	11114	brain, back of neck
1900	3441	11327	brain, back of neck
2000	3760	11824	brain, back of neck
2500	3746	12481	brain, back of neck
3000	3531	17133	brain, back of neck, frontal sinuses
4000	3599	17498	brain, back of neck, frontal sinuses, vertebrae
5000	3665	17565	brain, back of neck, frontal sinuses, vertebrae
10000	3665	18508	brain, back of neck, frontal sinuses, vertebrae, scalp
15000	3665	18508	brain, back of neck, frontal sinuses, vertebrae, scalp
20000	3665	18508	brain, back of neck, frontal sinuses, vertebrae, scalp
32767	3665	18508	end of chart

Table 4.2.3-1: Effects of Adjustments to Density Gradient Threshold

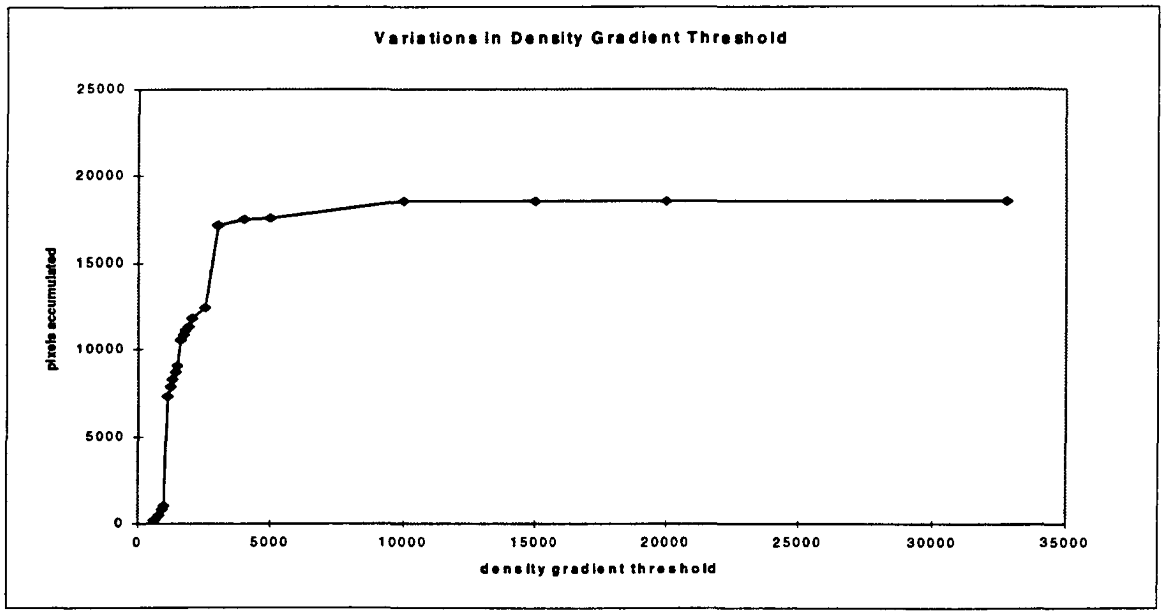


Figure 4.2.3-4: Effects of Adjustments to Density Gradient Threshold

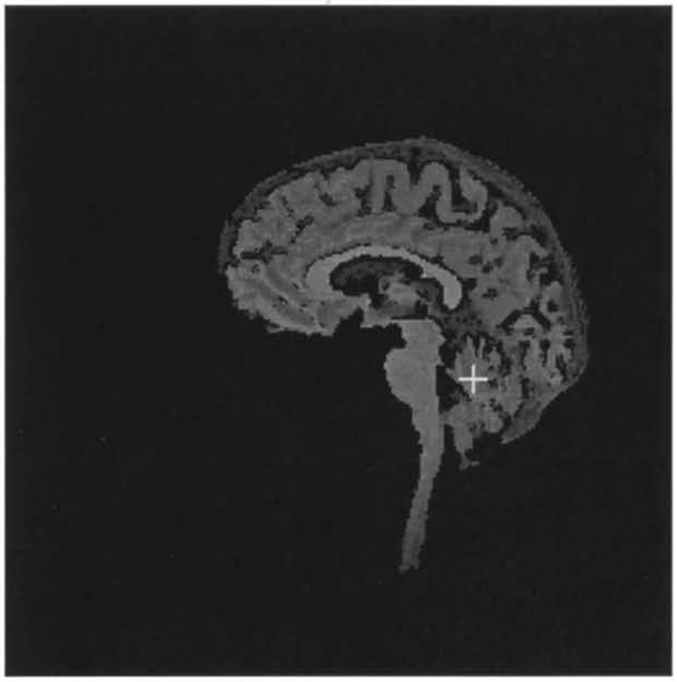
4.2.4 The Gradient Approximation Method

In order to compare the effectiveness of the three gradient-approximation methods used in this study, we perform three otherwise identical segmentations on the canonical image of the brain. Figure 4.2.4-1 demonstrates the use of the Prewitt kernel (the canonical Brain); figure 4.2.4-2, the Sobel kernel; and figure 4.2.4-3, the SSR formula.

Observe that the image created using the Sobel method appears somewhat eroded, with structures either incomplete (e.g. the superior sagittal sinus) or missing entirely (e.g. the pineal body). The image created using the SSR method is, by contrast, much too complete; it includes a great many undesired structures into the region.

In order more closely to examine and analyze the behavior of each method, a spreadsheet program was used to create sets of test matrices (tables 4.2.4-1 to 4.2.4-5). Each matrix set describes a 3 x 3 matrix as the values of its constituent cells gradually change (in nine steps) to reflect a change in density from one uniform level to another. At each increment, the density gradient is recorded for each gradient-approximation method. Table 4.2.4-6 synopsizes the results of the previous tables in this section; figure 4.2.4-4 displays these results in graphical form.

Output filename: FIG4241.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Iterative
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 2957
Pixels accumulated: 9137

Input filename: SAG-54.SLC

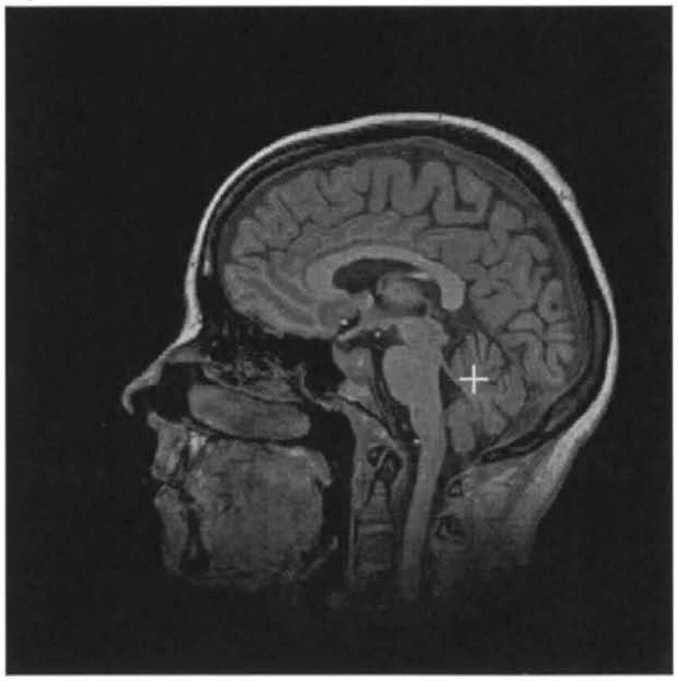
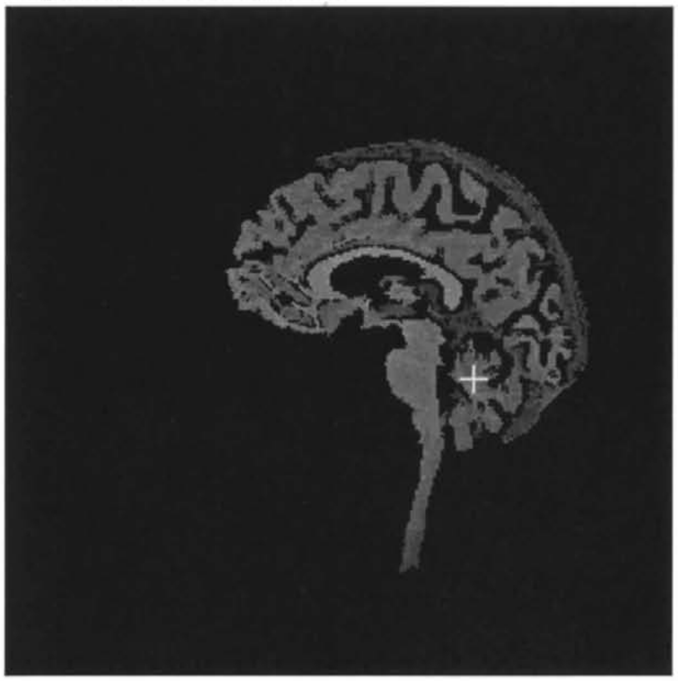


Figure 4.2.4-1: The Canonical Brain (Prewitt Gradient Approximation)

Output filename: FIG4242.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
Sobel
Region-growing method:
Iterative
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 2262
Pixels accumulated: 7447

Input filename: SAG-54.SLC

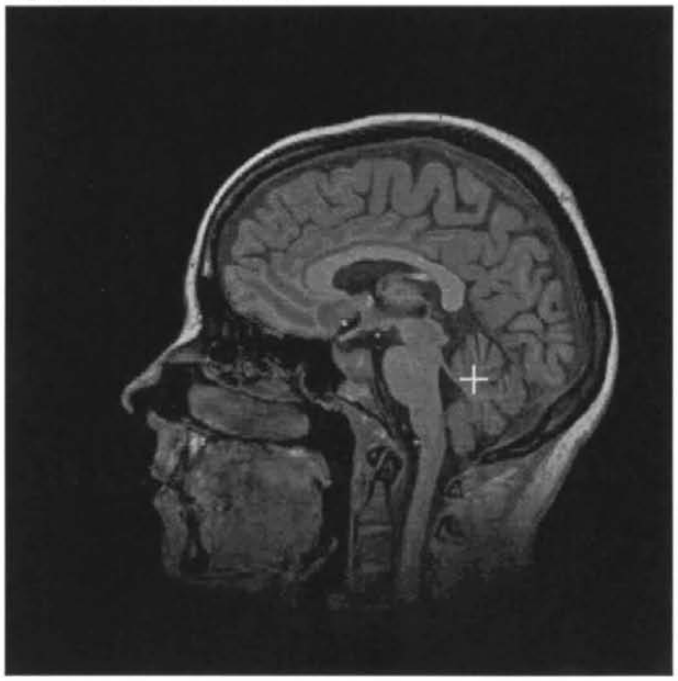
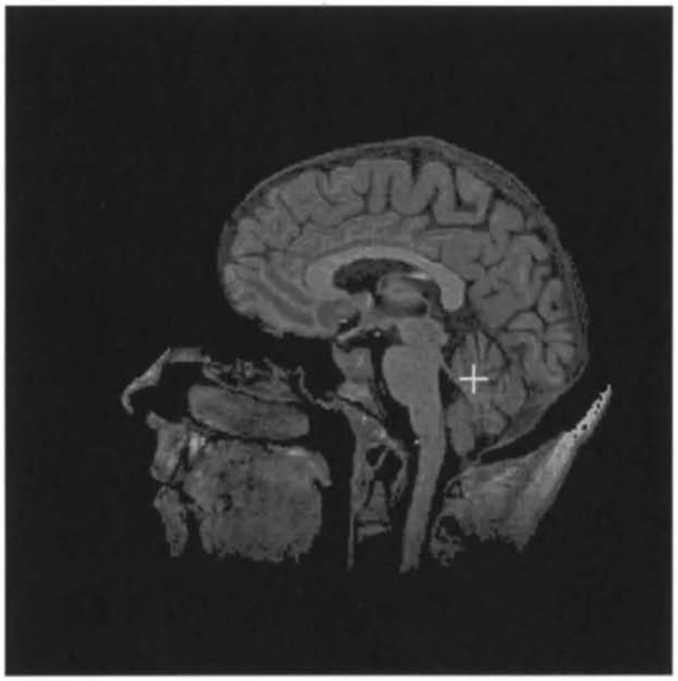


Figure 4.2.4-2: The Canonical Brain (Sobel Gradient Approximation)

Output filename: FIG4243.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
SSR
Region-growing method:
Iterative
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 3665
Pixels accumulated: 17541

Input filename: SAG-54.SLC

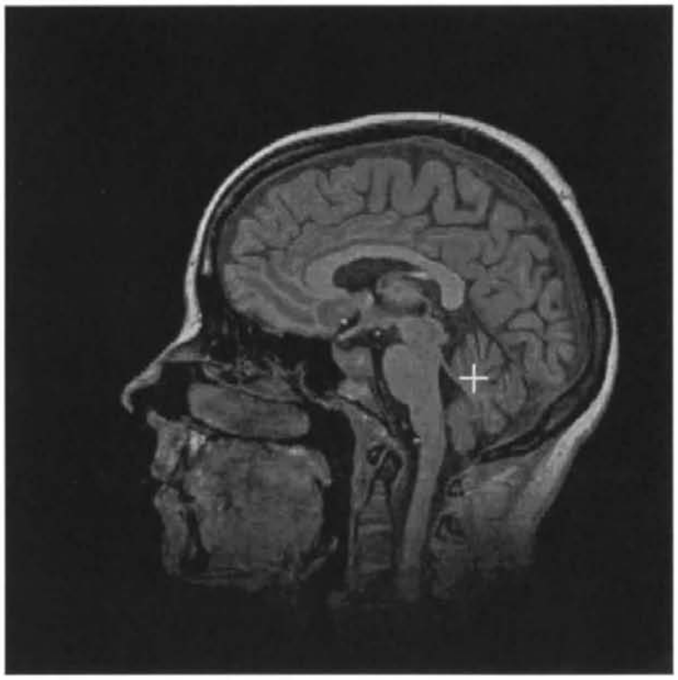


Figure 4.2.4-3: The Canonical Brain (Square-Sum-Root Gradient Approximation)

number of cells covered	test matrix	Prewitt	Sobel	SSR
0	$\begin{array}{\|ccc\|}\hline 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ \hline\end{array}$	0	0	0
1	$\begin{array}{\|ccc\|}\hline 1200 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ \hline\end{array}$	400	400	0
2	$\begin{array}{\|ccc\|}\hline 1200 & 1000 & 1000 \\ 1200 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ \hline\end{array}$	600	800	200
3	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1000 \\ 1200 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ \hline\end{array}$	800	1200	283
4	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1000 \\ 1200 & 1200 & 1000 \\ 1000 & 1000 & 1000 \\ \hline\end{array}$	800	1200	283
5	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1000 \\ 1200 & 1200 & 1000 \\ 1200 & 1000 & 1000 \\ \hline\end{array}$	800	1200	283
6	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1200 \\ 1200 & 1200 & 1000 \\ 1200 & 1000 & 1000 \\ \hline\end{array}$	800	1200	283
7	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1200 \\ 1200 & 1200 & 1000 \\ 1200 & 1200 & 1000 \\ \hline\end{array}$	600	800	200
8	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1200 \\ 1200 & 1200 & 1200 \\ 1200 & 1200 & 1000 \\ \hline\end{array}$	400	400	0
9	$\begin{array}{\|ccc\|}\hline 1200 & 1200 & 1200 \\ 1200 & 1200 & 1200 \\ 1200 & 1200 & 1200 \\ \hline\end{array}$	0	0	0

Table 4.2.4-1: Gradient Test Matrix Set (1.2:1)

number of cells covered	test matrix	Prewitt	Sobel	SSR
0	$\begin{matrix} 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 \end{matrix}$	0	0	0
1	$\begin{matrix} 1400 & 1000 & 1000 \\ 1000 & 1000 & 1000 \\ 1000 & 1000 & 1000 \end{matrix}$	800	800	0
2	$\begin{matrix} 1400 & 1000 & 1000 \\ 1400 & 1000 & 1000 \\ 1000 & 1000 & 1000 \end{matrix}$	1200	1600	400
3	$\begin{matrix} 1400 & 1400 & 1000 \\ 1400 & 1000 & 1000 \\ 1000 & 1000 & 1000 \end{matrix}$	1600	2400	566
4	$\begin{matrix} 1400 & 1400 & 1000 \\ 1400 & 1400 & 1000 \\ 1000 & 1000 & 1000 \end{matrix}$	1600	2400	566
5	$\begin{matrix} 1400 & 1400 & 1000 \\ 1400 & 1400 & 1000 \\ 1400 & 1000 & 1000 \end{matrix}$	1600	2400	566
6	$\begin{matrix} 1400 & 1400 & 1400 \\ 1400 & 1400 & 1000 \\ 1400 & 1000 & 1000 \end{matrix}$	1600	2400	566
7	$\begin{matrix} 1400 & 1400 & 1400 \\ 1400 & 1400 & 1000 \\ 1400 & 1400 & 1000 \end{matrix}$	1200	1600	400
8	$\begin{matrix} 1400 & 1400 & 1400 \\ 1400 & 1400 & 1400 \\ 1400 & 1400 & 1000 \end{matrix}$	800	800	0
9	$\begin{matrix} 1400 & 1400 & 1400 \\ 1400 & 1400 & 1400 \\ 1400 & 1400 & 1400 \end{matrix}$	0	0	0

Table 4.2.4-2: Gradient Test Matrix Set (1.4:1)

number of cells covered	test matrix	Prewitt	Sobel	SSR
0	1000 1000 1000 1000 1000 1000 1000 1000 1000	0	0	0
1	1600 1000 1000 1000 1000 1000 1000 1000 1000	1200	1200	0
2	1600 1000 1000 1600 1000 1000 1000 1000 1000	1800	2400	600
3	1600 1600 1000 1600 1000 1000 1000 1000 1000	2400	3600	849
4	1600 1600 1000 1600 1600 1000 1000 1000 1000	2400	3600	849
5	1600 1600 1000 1600 1600 1000 1600 1000 1000	2400	3600	849
6	1600 1600 1600 1600 1600 1000 1600 1000 1000	2400	3600	849
7	1600 1600 1600 1600 1600 1000 1600 1600 1000	1800	2400	600
8	1600 1600 1600 1600 1600 1600 1600 1600 1000	1200	1200	0
9	1600 1600 1600 1600 1600 1600 1600 1600 1600	0	0	0

Table 4.2.4-3: Gradient Test Matrix Set (1.6:1)

number of cells covered	test matrix	Prewitt	Sobel	SSR
0	$\begin{matrix}1000 & 1000 & 1000\\ 1000 & 1000 & 1000\\ 1000 & 1000 & 1000\end{matrix}$	0	0	0
1	$\begin{matrix}1800 & 1000 & 1000\\ 1000 & 1000 & 1000\\ 1000 & 1000 & 1000\end{matrix}$	1600	1600	0
2	$\begin{matrix}1800 & 1000 & 1000\\ 1800 & 1000 & 1000\\ 1000 & 1000 & 1000\end{matrix}$	2400	3200	800
3	$\begin{matrix}1800 & 1800 & 1000\\ 1800 & 1000 & 1000\\ 1000 & 1000 & 1000\end{matrix}$	3200	4800	1131
4	$\begin{matrix}1800 & 1800 & 1000\\ 1800 & 1800 & 1000\\ 1000 & 1000 & 1000\end{matrix}$	3200	4800	1131
5	$\begin{matrix}1800 & 1800 & 1000\\ 1800 & 1800 & 1000\\ 1800 & 1000 & 1000\end{matrix}$	3200	4800	1131
6	$\begin{matrix}1800 & 1800 & 1800\\ 1800 & 1800 & 1000\\ 1800 & 1000 & 1000\end{matrix}$	3200	4800	1131
7	$\begin{matrix}1800 & 1800 & 1800\\ 1800 & 1800 & 1000\\ 1800 & 1800 & 1000\end{matrix}$	2400	3200	800
8	$\begin{matrix}1800 & 1800 & 1800\\ 1800 & 1800 & 1800\\ 1800 & 1800 & 1000\end{matrix}$	1600	1600	0
9	$\begin{matrix}1800 & 1800 & 1800\\ 1800 & 1800 & 1800\\ 1800 & 1800 & 1800\end{matrix}$	0	0	0

Table 4.2.4-4: Gradient Test Matrix Set (1.8:1)

number of cells covered	test matrix	Prewitt	Sobel	SSR
0	1000 1000 1000 1000 1000 1000 1000 1000 1000	0	0	0
1	2000 1000 1000 1000 1000 1000 1000 1000 1000	2000	2000	0
2	2000 1000 1000 2000 1000 1000 1000 1000 1000	3000	4000	1000
3	2000 2000 1000 2000 1000 1000 1000 1000 1000	4000	6000	1414
4	2000 2000 1000 2000 2000 1000 1000 1000 1000	4000	6000	1414
5	2000 2000 1000 2000 2000 1000 2000 1000 1000	4000	6000	1414
6	2000 2000 2000 2000 2000 1000 2000 1000 1000	4000	6000	1414
7	2000 2000 2000 2000 2000 1000 2000 2000 1000	3000	4000	1000
8	2000 2000 2000 2000 2000 2000 2000 2000 1000	2000	2000	0
9	2000 2000 2000 2000 2000 2000 2000 2000 2000	0	0	0

Table 4.2.4-5: Gradient Test Matrix Set (2.0:1)

Prewitt:	ratio				
cells covered	1.2:1	1.4:1	1.6:1	1.8:1	2.0:1
0	0	0	0	0	0
1	400	800	1200	1600	2000
2	600	1200	1800	2400	3000
3	800	1600	2400	3200	4000
4	800	1600	2400	3200	4000
5	800	1600	2400	3200	4000
6	800	1600	2400	3200	4000
7	600	1200	1800	2400	3000
8	400	800	1200	1600	2000
9	0	0	0	0	0

Sobel:	ratio				
cells covered	1.2:1	1.4:1	1.6:1	1.8:1	2.0:1
0	0	0	0	0	0
1	400	800	1200	1600	2000
2	800	1600	2400	3200	4000
3	1200	2400	3600	4800	6000
4	1200	2400	3600	4800	6000
5	1200	2400	3600	4800	6000
6	1200	2400	3600	4800	6000
7	800	1600	2400	3200	4000
8	400	800	1200	1600	2000
9	0	0	0	0	0

SSR:	ratio				
cells covered	1.2:1	1.4:1	1.6:1	1.8:1	2.0:1
0	0	0	0	0	0
1	0	0	0	0	0
2	200	400	600	800	1000
3	283	566	849	1131	1414
4	283	566	849	1131	1414
5	283	566	849	1131	1414
6	283	566	849	1131	1414
7	200	400	600	800	1000
8	0	0	0	0	0
9	0	0	0	0	0

Table 4.2.4-6: Gradient Approximation Method Responses for Test Matrices

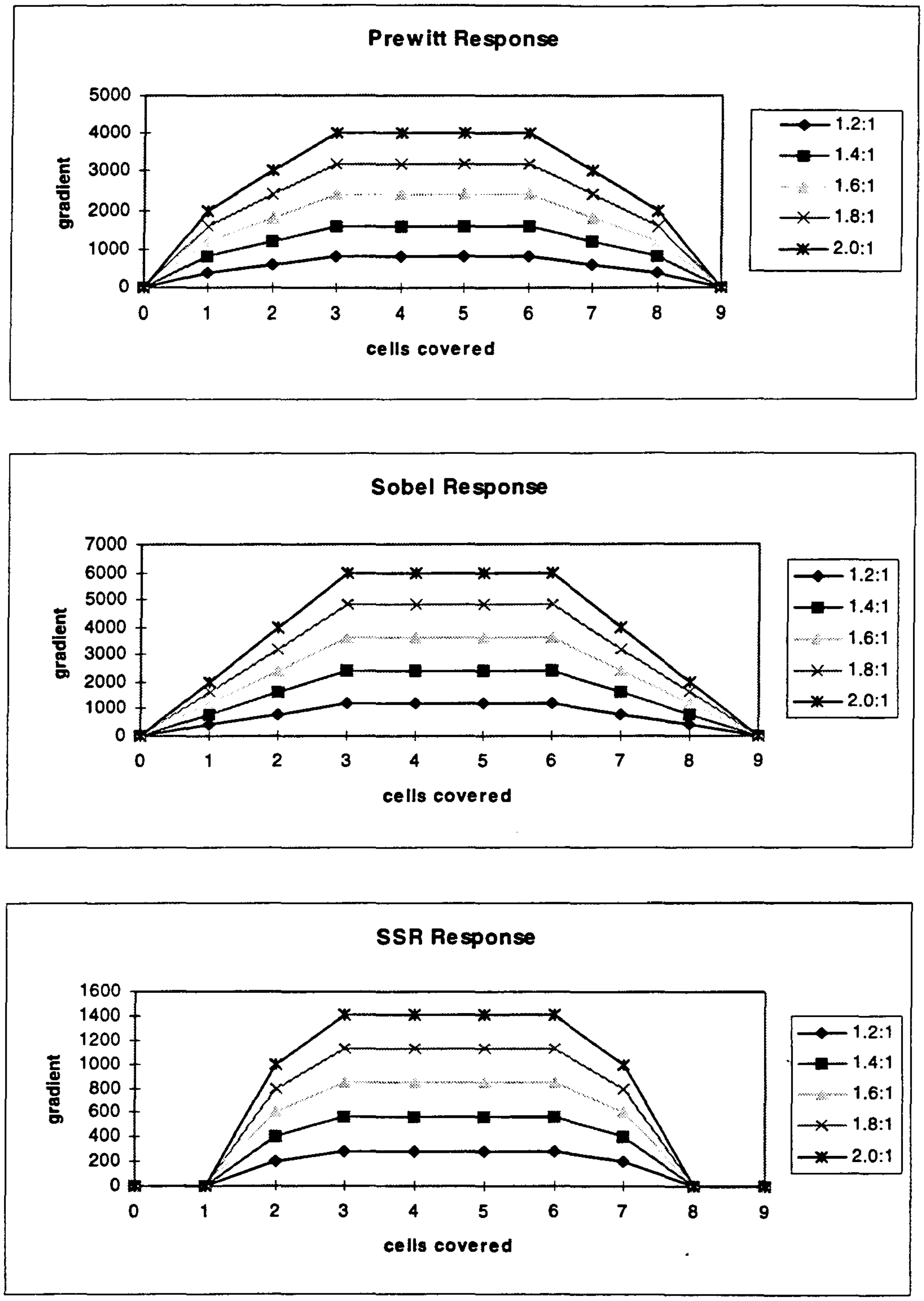


Figure 4.2.4-4: Gradient Approximation Method Responses for Test Matrices

We observe two fundamental differences in the curves shown in the graph: first, the range of density gradients encountered; and second, the shapes of the curves. The range of the curves for the Prewitt method is 0 to 4000; the range for the Sobel method is 0 to 6000; and for the SSR method, 0 to 1414.

Whereas the absolute gradient range by itself is unimportant, it becomes significant if one does not take it into account when assigning an appropriate gradient threshold for the gradient-approximation method being used. For example, a threshold of 1500 was appropriate for segmenting the canonical Brain using the Prewitt method; such a number ensures that no undesired areas will be included into the segmented image. However, this same number is higher than most of the gradients which are computed during SSR segmentation, making the gradient threshold largely irrelevant. (As we shall see, the SSR threshold should be, in this case, about a third of the Prewitt threshold in order to create a reasonably good image.)

In a similar fashion, because the Sobel kernel magnifies the effect of an encounter with an edge, a threshold which works for a Prewitt segmentation may not be high enough for a Sobel segmentation in order accurately to include all parts of a desired region.

The shape of a gradient-approximation response curve may also play a part in the results of segmentation (figure 4.2.4-4). Over a range of a given kernel's cell-coverings, the response curves of the different methods may be compared in order to answer certain questions: How quickly does a particular method respond to a change in density? How strongly does it respond? Does it display any anomalies in its response (e.g. dips or flat spots in the graph)? How do these characteristics manifest themselves in terms of image quality?

Tables 4.2.4-7 and 4.2.4-8, and their accompanying graphs, figures 4.2.4-5 and 4.2.4-6, are presented for evaluation. We first normalize the graph of the gradients computed for each method at a given starting-to-ending-density ratio (in this case, 2.0:1). By normalizing the numbers, we scale them to the same range of values to ensure a fairer comparison. In this case, the SSR numbers were multiplied by a constant of 2.86 in order to scale them to the level of the Prewitt numbers. Then, the modified SSR numbers and the Prewitt numbers were multiplied by 1.5 in order to scale them to the level of the Sobel numbers.

Normalized Density Gradients			
cells covered	Prewitt	Sobel	SSR
0	0	0	0
1	3000	2000	0
2	4500	4000	4290
3	6000	6000	6066
4	6000	6000	6066
5	6000	6000	6066
6	6000	6000	6066
7	4500	4000	4290
8	3000	2000	0
9	0	0	0

Table 4.2.4-7: Normalized Density Gradients of Gradient Test Matrix (2.0:1)

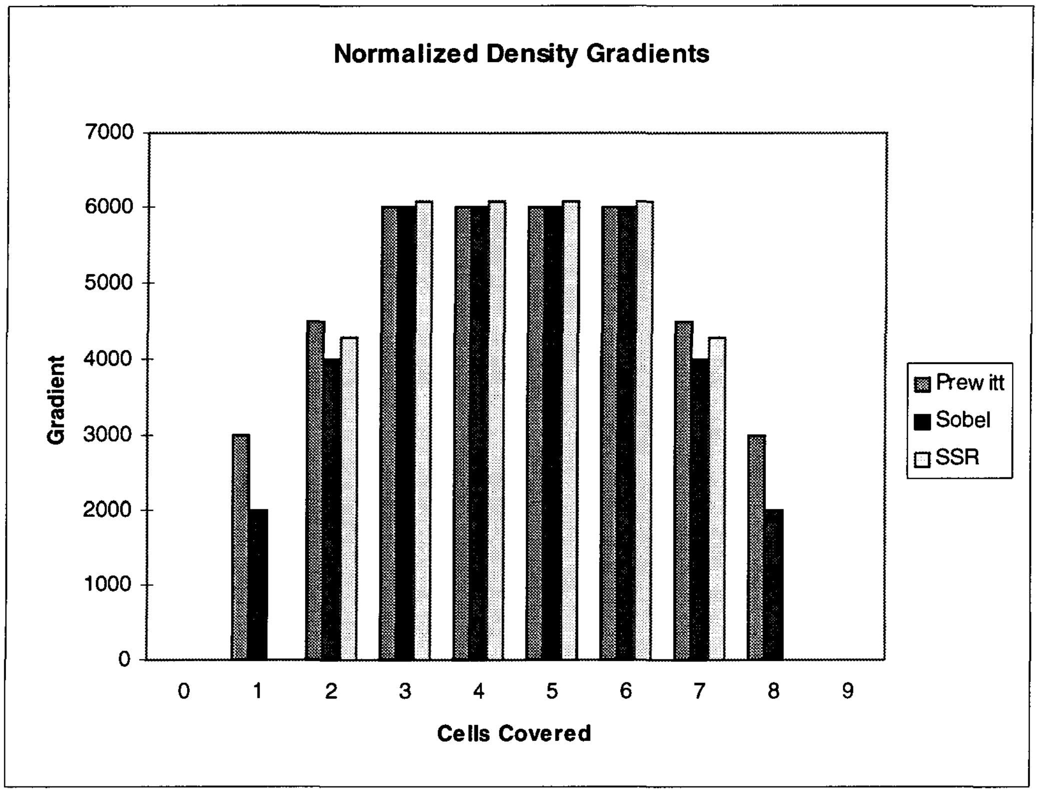


Figure 4.2.4-5: Normalized Density Gradients of Gradient Test Matrix (2.0:1)

First Derivative of Normalized Density Gradients			
cells covered	Prewitt	Sobel	SSR
0	0	0	0
1	3000	2000	0
2	1500	2000	4290
3	1500	2000	1776
4	0	0	0
5	0	0	0
6	0	0	0
7	-1500	-2000	-1776
8	-1500	-2000	-4290
9	-3000	-2000	0

Table 4.2.4-8: First Derivative of Normalized Densities of Gradient

Test Matrix (2.0:1)

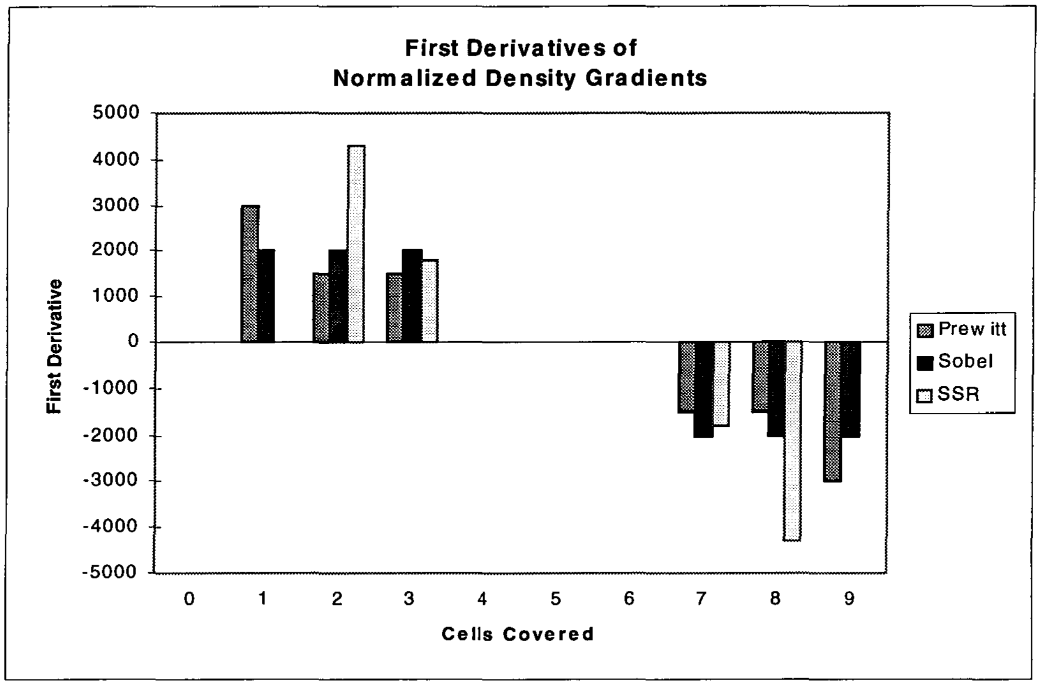


Figure 4.2.4-6: First Derivative of Normalized Density Gradients of Gradient

Test Matrix (2.0:1)

Next, we look at the rate of change of gradient with respect to the number of cells covered; in effect, the first derivative of the normalized gradients. (Note that, since the SSR method generates no gradient for the first, second, and ninth intervals, those intervals are not useful for evaluating this method. Also note that symmetry exists about the fifth interval: the fourth and sixth, third and seventh, second and eighth, and first and ninth intervals are the negatives of each other.)

After all of this effort, it seems anticlimactic to report that the differences in gradient first-derivatives is fairly small for the three approximation methods studied. Using the highly simplified test matrices, the three methods yield respond in an approximately equal manner. When comparing images created using each method for the canonical seedpoints (and adjusting the threshold accordingly), we observe that the results are not radically different. In general, the differences are either poor or overzealous region growth, or degraded image quality due to erosion of boundary areas.

4.2.4.1 Prewitt Gradient Approximation

The Prewitt method was used to generate the so-called canonical images displayed in section 4.3. As such, they are used as a basis for comparison in evaluating the effectiveness of the other gradient-approximation methods.

4.2.4.2 Sobel Gradient Approximation

As has been noted, a threshold which works for a Prewitt segmentation may not be high enough for a Sobel segmentation. We observe this in practice by performing canonical segmentations using the Sobel method instead of the Prewitt, but retaining the canonical (Prewitt) gradient threshold. When the region to be segmented is fairly small (e.g. the corpus callosum or the pons), the difference in quality of segmentation may not be noticeable to the eye, although the degree of segmentation is different, as measured by the number of recursive calls made and the number of pixels accumulated into the segmented region (see table 4.2.4.2-1 and figure 4.2.4.2-1).

As the size of the desired region increases, however, boundary erosion and poor region growth become evident (see figure 4.2.4.2-2). The remedy is to increase the threshold by some amount, from perhaps 20% to as much as 80% of the Prewitt threshold. Even then, the quality of the region boundaries may leave a little to be desired.

	Sobel			Sobel			Prewitt			
Structure	Prewitt Threshold	Stack Frames	Pixels Accumulated	Modified Threshold	Stack Frames	Pixels Accumulated	Prewitt Threshold	Stack Frames	Pixels Accumulated	Remarks on Sobel Quality
corpus callosum	1000	211	310	1250	262	347	1000	257	354	as good as Prewitt
pons	800	248	613	1000	272	633	800	272	640	as good as Prewitt
worm of cerebellum	1800	217	581	2400	358	659	1800	346	663	as good as Prewitt
cerebrum	900	404	923	1225	963	2730	900	970	2720	poor growth
scalp	8000	464	866	15000	516	1224	8000	516	1214	poor growth
superior sagittal sinus	1400	399	1041	1900	635	1758	1400	666	1770	poor growth
brain	1500	2262	7447	2100	2754	9117	1500	2957	9137	mediocre growth, eroded
head	10000	11775	25214	11000	13395	25281	10000	13395	25358	cranium not fully formed

Table 4.2.4.2-1: Results of Canonical Image Generation Using

Sobel Gradient Approximation

	SSR			SSR			Prewitt			
Structure	Prewitt Threshold	Stack Frames	Pixels Accumulated	Modified Threshold	Stack Frames	Pixels Accumulated	Prewitt Threshold	Stack Frames	Pixels Accumulated	Remarks on SSR Quality
corpus callosum	1000	377	735	250	259	339	1000	257	354	includes dense areas of frontal lobe
pons	800	512	2448	250	297	646	800	272	640	includes frontal lobe
worm of cerebellum	1800	1986	6791	575	330	663	1800	346	663	includes brain, stem, c.c.
cerebrum	900	1623	6681	279	800	2668	900	970	2720	includes brain, stem, c.c.
scalp	8000	516	1224	8000*	516	1224	8000	516	1214	almost identical to canonical
superior sagittal sinus	1400	5613	16858	414	577	1645	1400	666	1770	includes sup. sag. sinus other intracerebral material
brain	1500	3665	17541	465	3052	8917	1500	2957	9137	includes brain, neck, jaw, face
head	10000	13395	25358	10000**	13395	25358	10000	13395	25358	identical to canonical

* maximum gradient crossing the cranium -= 1800; density<400 on either side

** probably constrained by LDL and UDL

Table 4.2.4.3-1: Results of Canonical Image Generation Using

SSR Gradient Approximation

Output filename: FIG42421.EPS

GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [181, 147, 54]

Gradient-approximation method:

Sobel

Region-growing method:

Iterative

LDL: 900

UDL: 1800

Threshold: 1800

Stack frames used: 217

Pixels accumulated: 581

Input filename: SAG-54.SLC

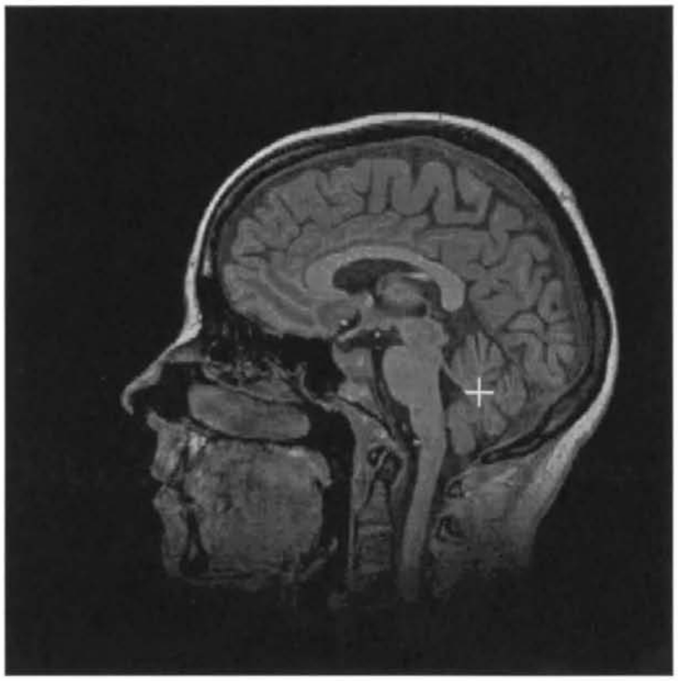
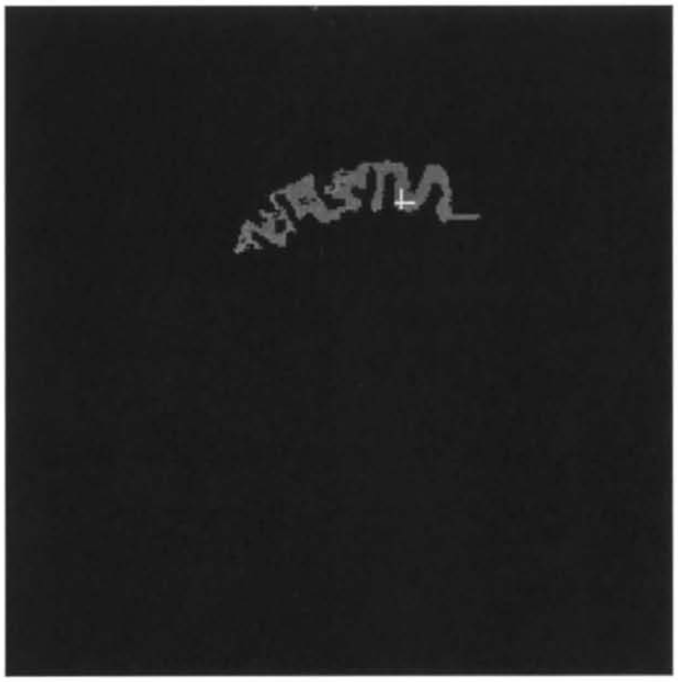


Figure 4.2.4.2-1: Canonical Worm of Cerebellum Using Sobel Gradient Approximation

Output filename: FIG42422.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [151, 75, 54]
Gradient-approximation method:
Sobel
Region-growing method:
Iterative
LDL: 900
UDL: 2000
Threshold: 900
Stack frames used: 404
Pixels accumulated: 923

Input filename: SAG-54.SLC

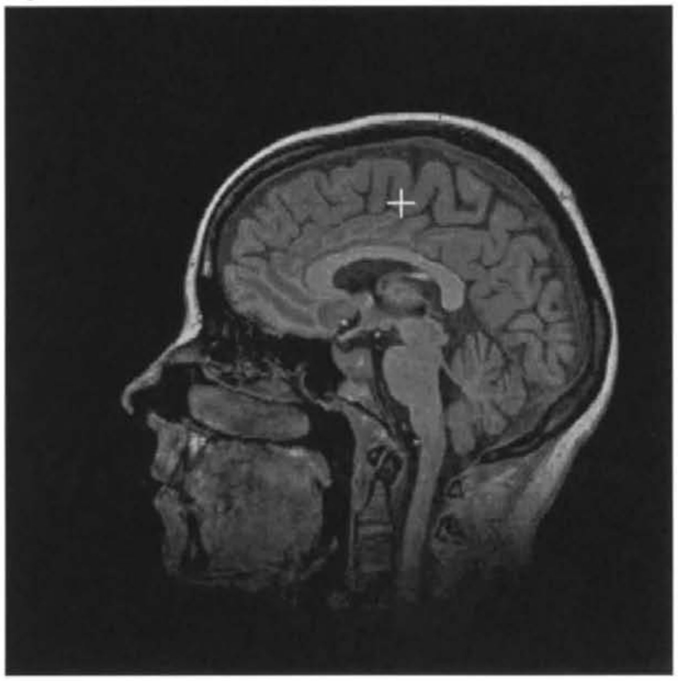
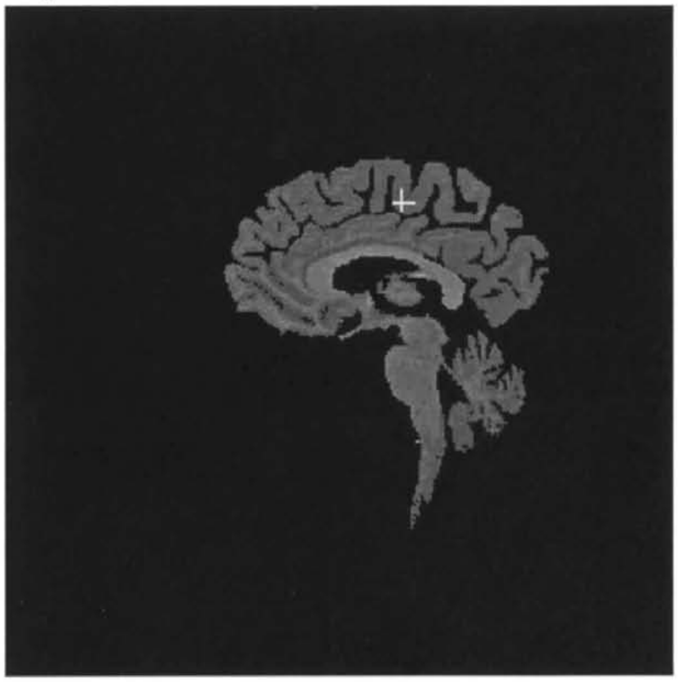


Figure 4.2.4.2-2: Canonical Cerebrum Using Sobel Gradient Approximation

4.2.4.3 Square-Sum-Root Gradient Approximation

Since the gradient approximations computed by the SSR method are uniformly much lower than those computed by the other methods studied, it comes as no surprise that the use of a threshold which is successful with the Prewitt method should fail miserably (most of the time) with the SSR method. Many undesired areas are included in the segmented region due to an excessively high threshold. The remedy is, of course, to reduce the threshold to a more appropriate level, perhaps to a third or a fourth of the Prewitt value. Table 4.2.4.3-1 synopsizes the stack-frame/pixel-accumulation data recorded for the SSR trials; figures 4.2.4.3-1 and 4.2.4.3-2 show before-and-after attempts to segment the cerebrum using the SSR method.

Output filename: FIG42431.EPS



GETCOORD - Bill Bell - Fall 1996

```
x-y-z coordinates: [151, 75, 54]
Gradient-approximation method:
    SSR
Region-growing method:
    Iterative
LDL: 900
UDL: 2000
Threshold: 900
Stack frames used: 1623
Pixels accumulated: 6681
```

Input filename: SAG-54.SLC

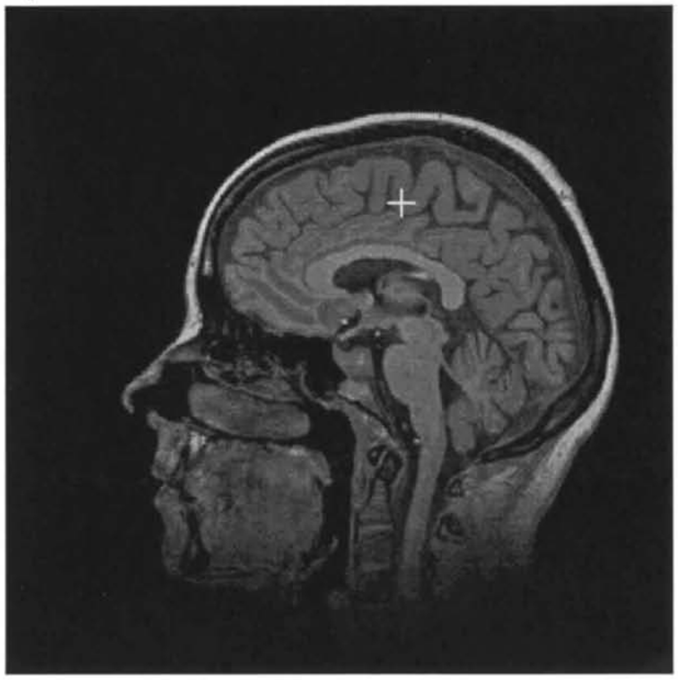
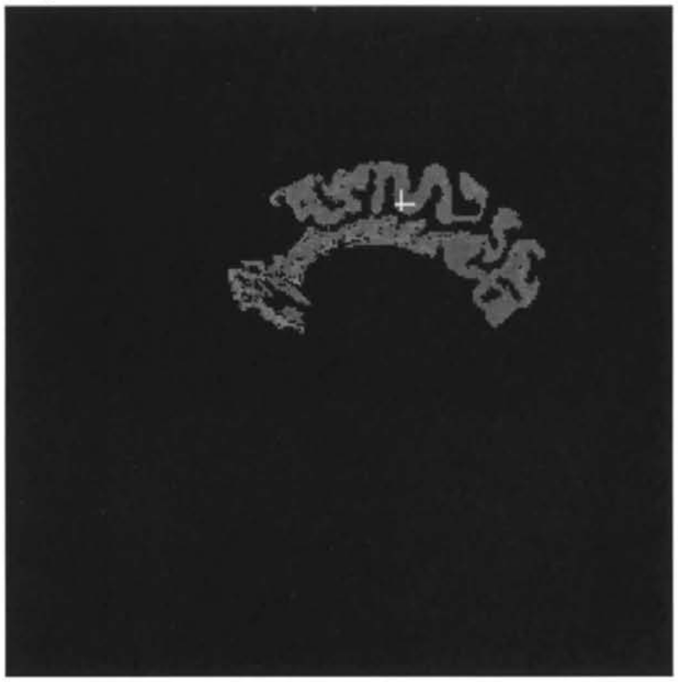


Figure 4.2.4.3-1: Canonical Cerebrum Using SSR Gradient Approximation

Output filename: FIG42432.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [151, 75, 54]
Gradient-approximation method:
SSR
Region-growing method:
Iterative
LDL: 900
UDL: 2000
Threshold: 279
Stack frames used: 800
Pixels accumulated: 2668

Input filename: SAG-54.SLC

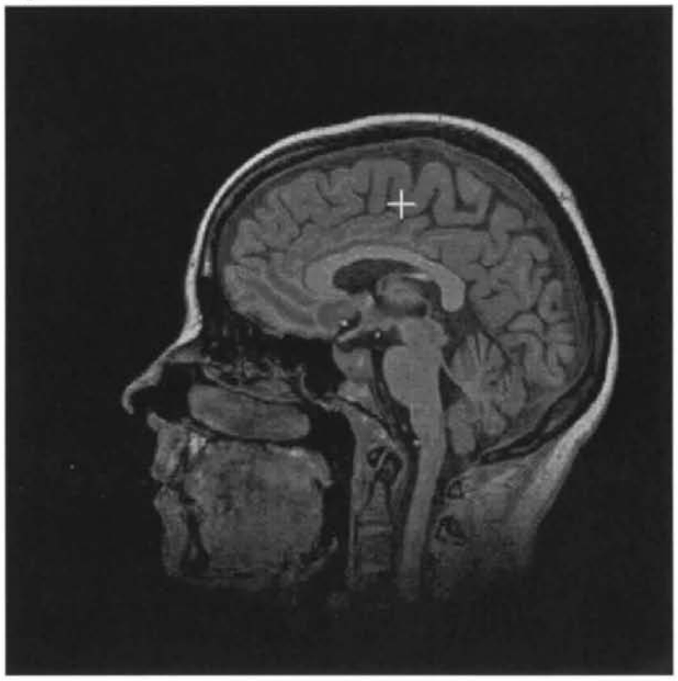


Figure 4.2.4.3-2: Canonical Cerebrum Using SSR Gradient Approximation (Threshold Adjusted)

4.2.5 The Region-growing Method

Our primary objective in performing segmentation is to create a beautiful and informative image of a structure. We must recognize also the limitations of space and time; and so we explore different region-growing algorithms in order to determine the characteristics of each algorithm with respect to memory usage and runtimes, as well as to its ability to segment effectively.

4.2.5.1 4-connected and 8-connected Recursion

The program GETCOOR8 was used to generate the images displayed in this section. It is exactly the same as the GETCOORD program, except that a compiler directive is enabled which expands the 4-connected region-growing function (do_floodfill_rec()) to an 8-connected function. To assess the effects of 8-connected region-growing, we recreate the canonical images with GETCOOR8 and compare the results with those created with GETCOORD.

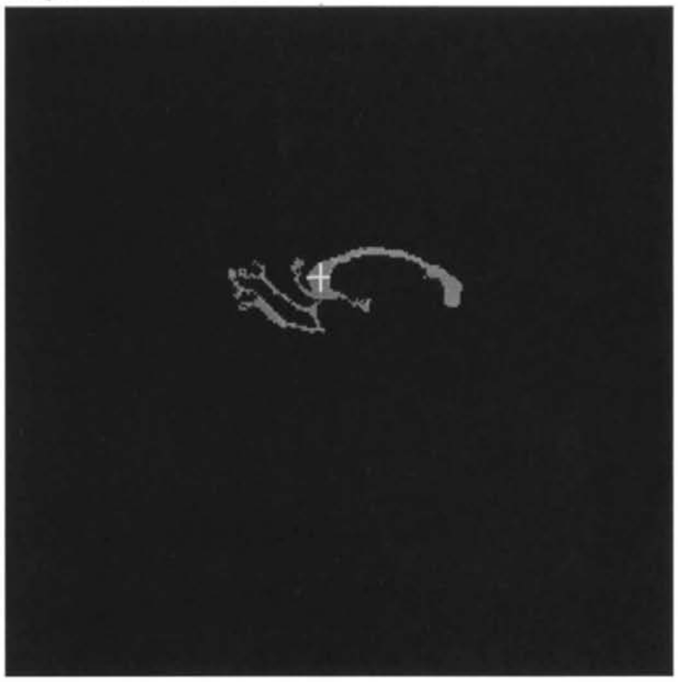
Compare the 8-connected image of the corpus callosum (figure 4.2.5.1-1) with its 4-connected counterpart (figure 4.3-1). 8-connectedness causes dense areas of the frontal lobe to be included with the corpus callosum itself. Reducing the gradient threshold to 800 from 1000 removes most of the extra frontal lobe area (figure 4.2.5.1-2). Increasing the lower density limit to 1400 from 1200 removes the frontal lobe entirely, although the remaining image appears somewhat eroded (figure 4.2.5.1-3).

Figure 4.2.5.1-4 shows the drastic effect of 8-connectedness on the 4-connected canonical image of the cerebrum (figure 4.3-4). In this case, 8-connectedness causes the corpus callosum, pons, and oblong medulla to be included with the cerebrum. Reducing the gradient threshold to 750 from 900 helps improve the segmentation, although at some cost to the quality of the remaining image (figure 4.2.5.1-5); especially so, since the gradient in the cerebrum, a highly convoluted structure, tends to be on the high side.

By contrast, the appearance of some structures does not change much with the application of 8-connected region-growing. For example, figure 4.2.5.1-6 shows the result of 8-connectedness on the image of the scalp. Comparing this picture to the canonical scalp in figure 4.3-5, we see that the only addition of matter appears at the bridge of the nose. We conjecture that the scalp is, due to the relatively high gradient surrounding it, already well-segmented; 8-connectivity doesn't have much of an effect under these circumstances. Similar remarks hold for the superior sagittal sinus (8-connected, figure 4.2.5.1-7; canonical 4-connected, figure 4.3-6).

Output filename: FIG42511.EPS

GETCOOR8 - Bill Bell - Fall 1996



```
x-y-z coordinates: [120, 103, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Recursive
LDL: 1200
UDL: 2000
Threshold: 1000
Stack frames used: 262
Pixels accumulated: 596
```

Input filename: SAG-54.SLC

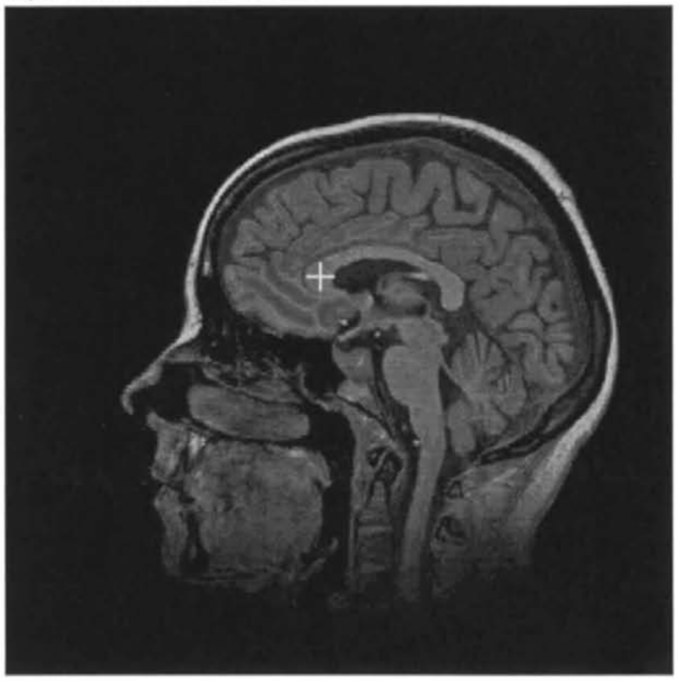
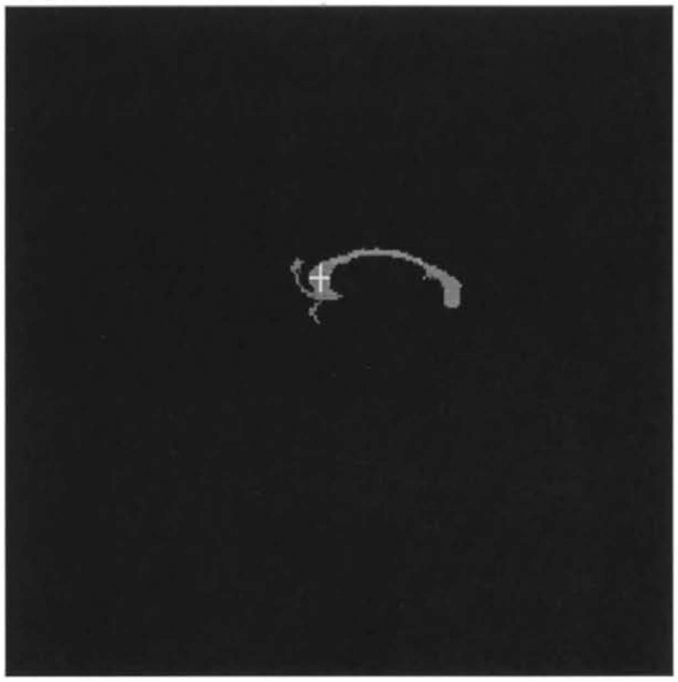


Figure 4.2.5.1-1: Corpus Callosum Using 8-connected Segmentation

Output filename: FIG42512.EPS

GETCOOR8 - Bill Bell - Fall 1996



x-y-z coordinates: [120, 103, 54]

Gradient-approximation method:
Prewitt

Region-growing method:
Recursive

LDL: 1200

UDL: 2000

Threshold: 800

Stack frames used: 226

Pixels accumulated: 356

Input filename: SAG-54.SLC

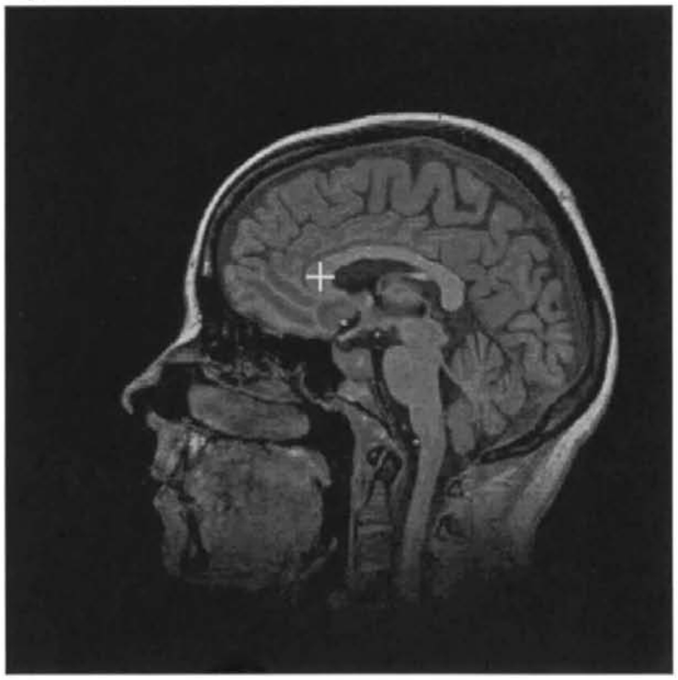
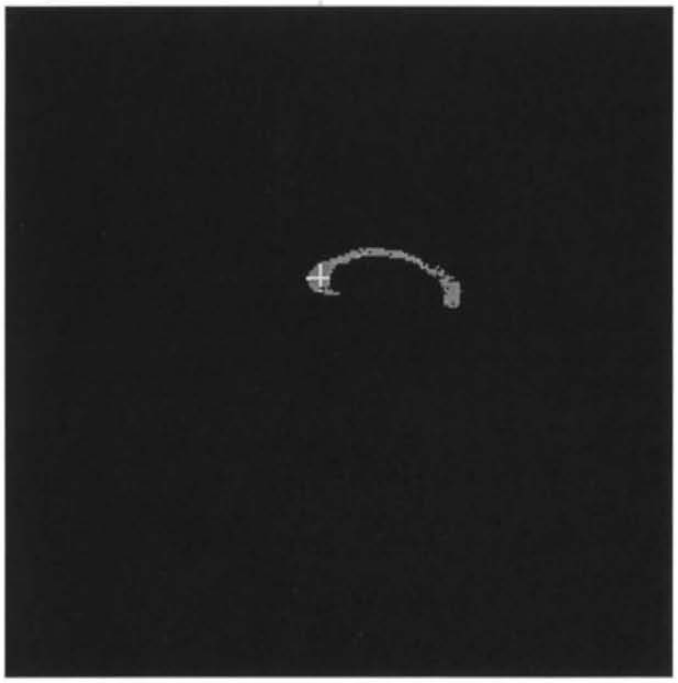


Figure 4.2.5.1-2: Corpus Callosum Using 8-connected Segmentation
(reduced gradient threshold)

Output filename: FIG42513.EPS



GETCOOR8 - Bill Bell - Fall 1996

x-y-z coordinates: [120, 103, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 1400
UDL: 2000
Threshold: 1000
Stack frames used: 198
Pixels accumulated: 276

Input filename: SAG-54.SLC

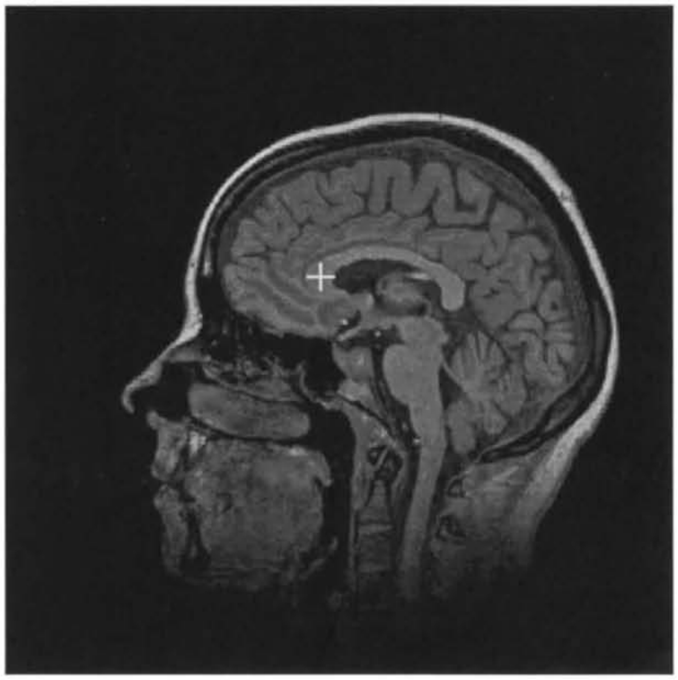
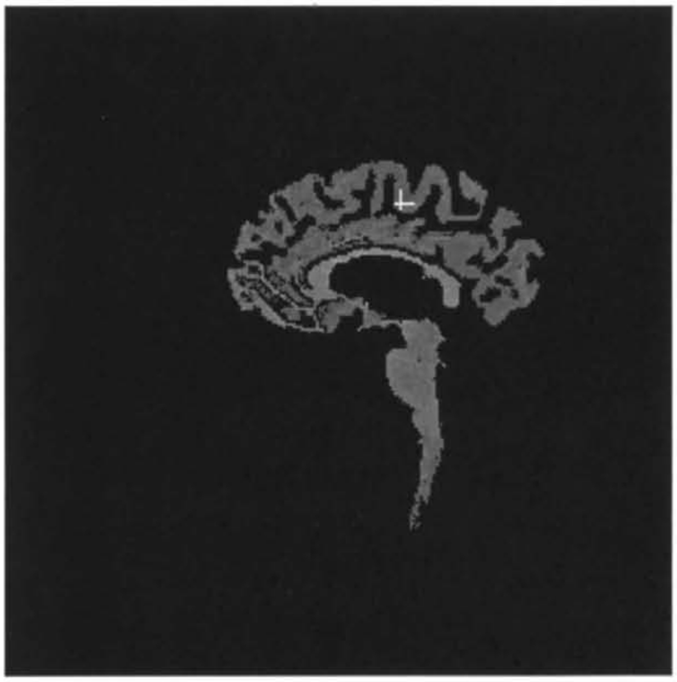


Figure 4.2.5.1-3: Corpus Callosum Using 8-connected Segmentation (increased LDL)

Output filename: FIG42514.EPS

GETCOOR8 - Bill Bell - Fall 1996



x-y-z coordinates: [151, 75, 54]

Gradient-approximation method:
  Prewitt

Region-growing method:
  Recursive

LDL: 900

UDL: 2000

Threshold: 900

Stack frames used: 1398

Pixels accumulated: 4216

Input filename: SAG-54.SLC

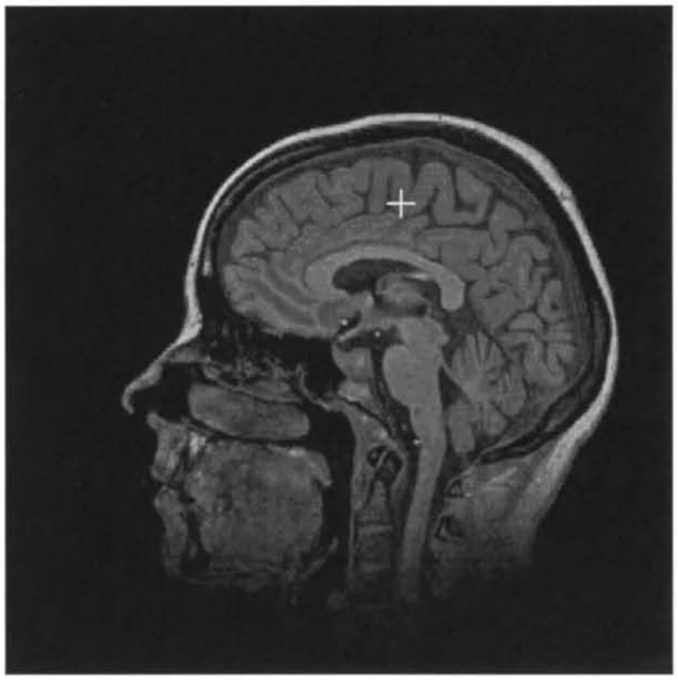
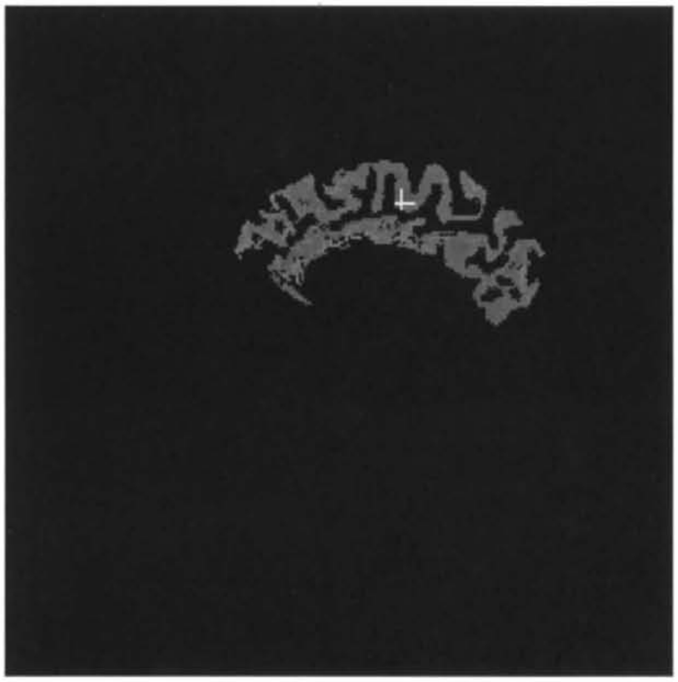


Figure 4.2.5.1-4: Cerebrum Using 8-connected Segmentation

Output filename: FIG42515.EPS



GETCOOR8 - Bill Bell - Fall 1996

x-y-z coordinates: [151, 75, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 900
UDL: 2000
Threshold: 750
Stack frames used: 1202
Pixels accumulated: 2345

Input filename: SAG-54.SLC

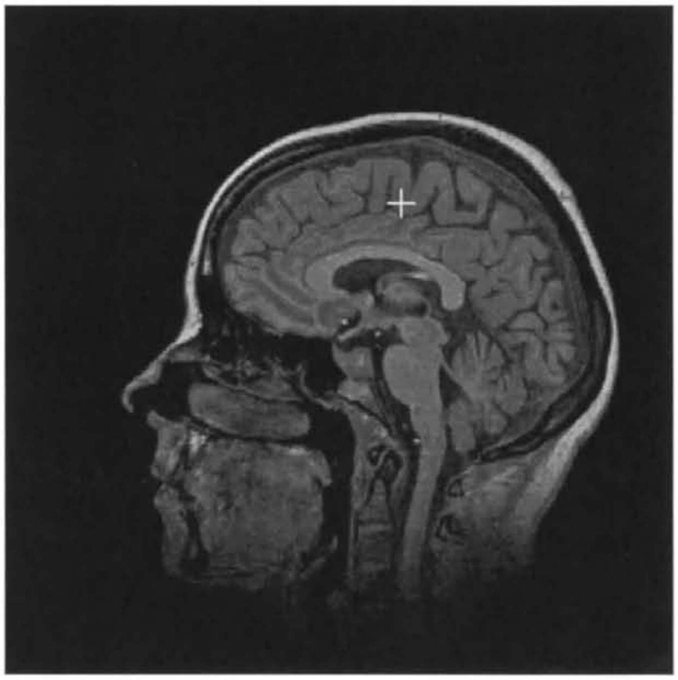
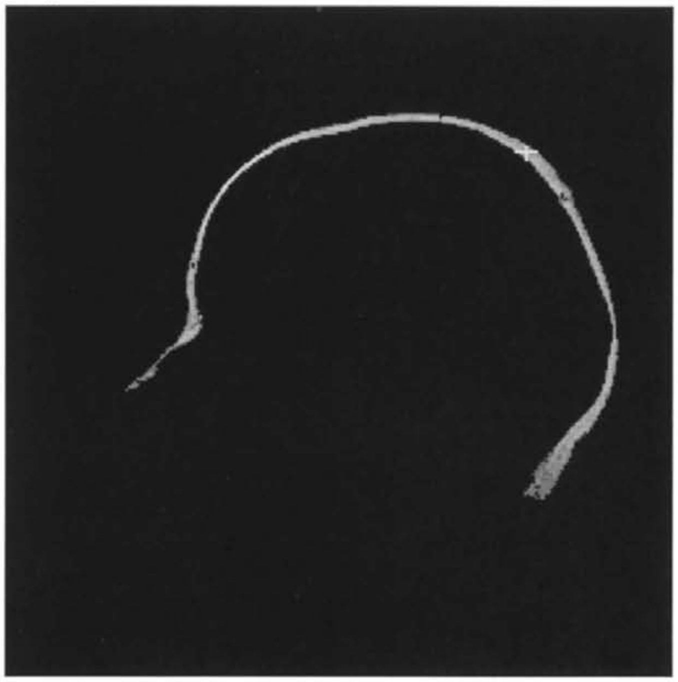


Figure 4.2.5.1-5: Cerebrum Using 8-connected Segmentation
(reduced gradient threshold)

Output filename: FIG42516.EPS



GETCOOR8 - Bill Bell - Fall 1996

x-y-z coordinates: [199, 55, 54]

Gradient-approximation method:
Prewitt

Region-growing method:
Recursive

LDL: 1200

UDL: 2800

Threshold: 8000

Stack frames used: 548

Pixels accumulated: 1247

Input filename: SAG-54.SLC

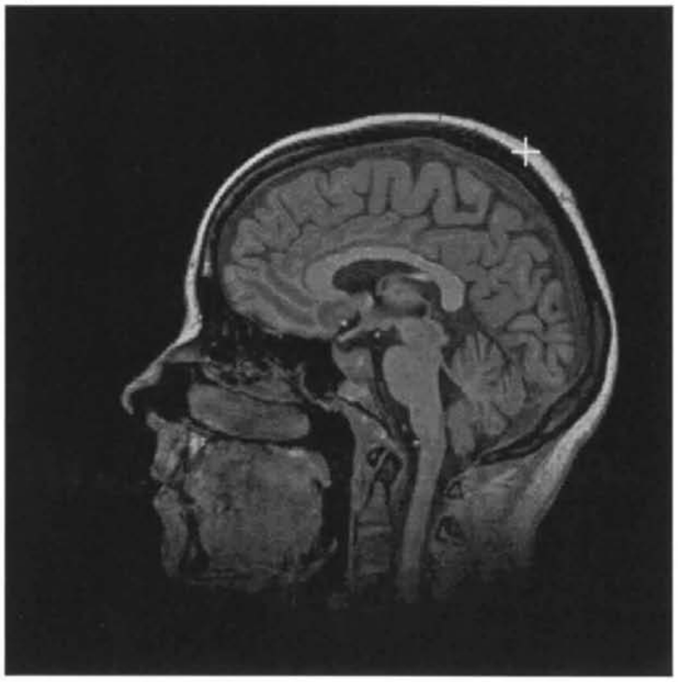
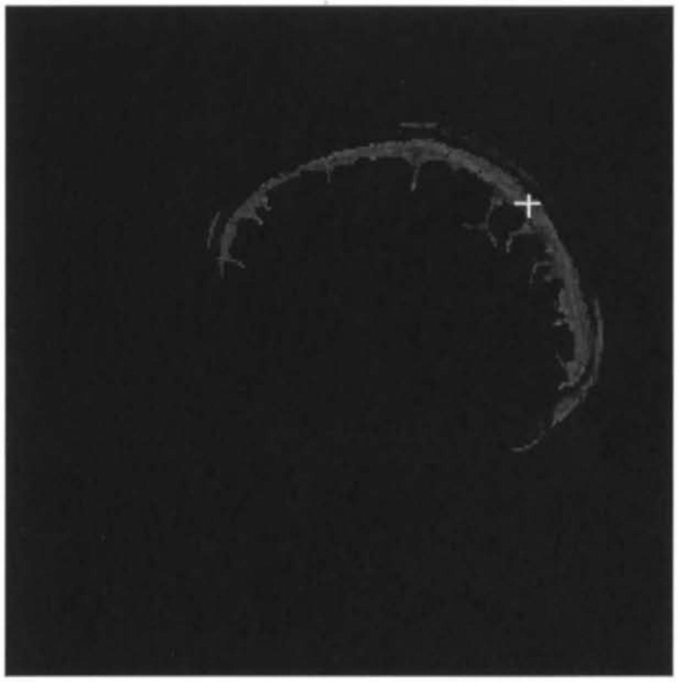


Figure 4.2.5.1-6: Scalp Using 8-connected Segmentation

Output filename: FIG42517.EPS

GETCOOR8 - Bill Bell - Fall 1996



x-y-z coordinates: [200, 75, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 100
UDL: 900
Threshold: 1400
Stack frames used: 704
Pixels accumulated: 2240

Input filename: SAG-54.SLC

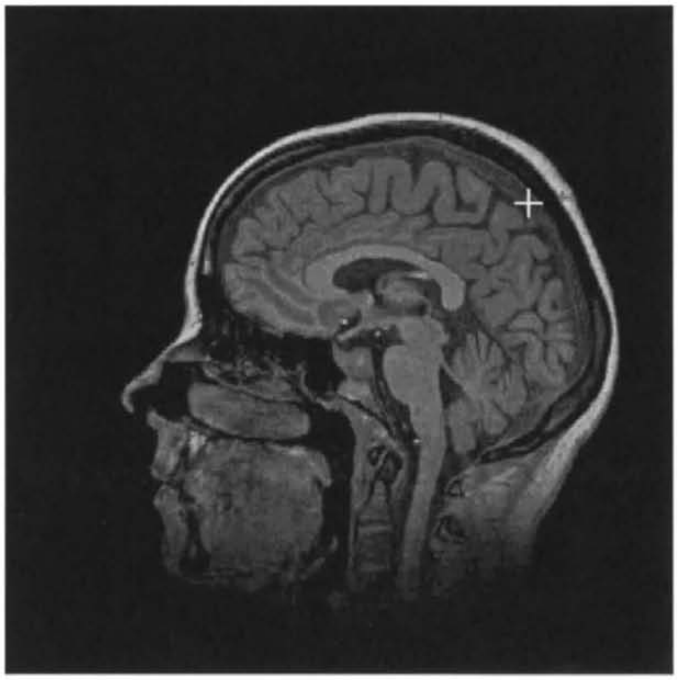


Figure 4.2.5.1-7: Superior Sagittal Sinus Using 8-connected Segmentation

In retrospect, let us observe quantitatively the results and the resource usage of the 4-connected and 8-connected recursive region-growing algorithms. As table 4.2.5.1-1 shows, in each of four benchmark cases studied, the 8-connected algorithm has accumulated more pixels for its image, and uses more stack frames, than the 4-connected method. The images of the cerebrum, and, to a lesser extent, the corpus callosum, could be considered degenerate cases; the enhanced connectedness of the 8-connected method has caused undesired areas to be included in the segmentation. We shall have more to say on the subject of stack limitations in section 4.2.5.4.

structure	4-connected region-growing			8-connected region-growing		
	figure	pixels	frames	figure	pixels	frames
corpus callosum	4.3-1	257	354	4.2.5.1-1	262	596
cerebrum	4.3-4	970	2720	4.2.5.1-4	1398	4216
scalp	4.3-5	516	1214	4.2.5.1-6	548	1247
sup. sag. sinus	4.3-6	666	1770	4.2.5.1-7	704	2240

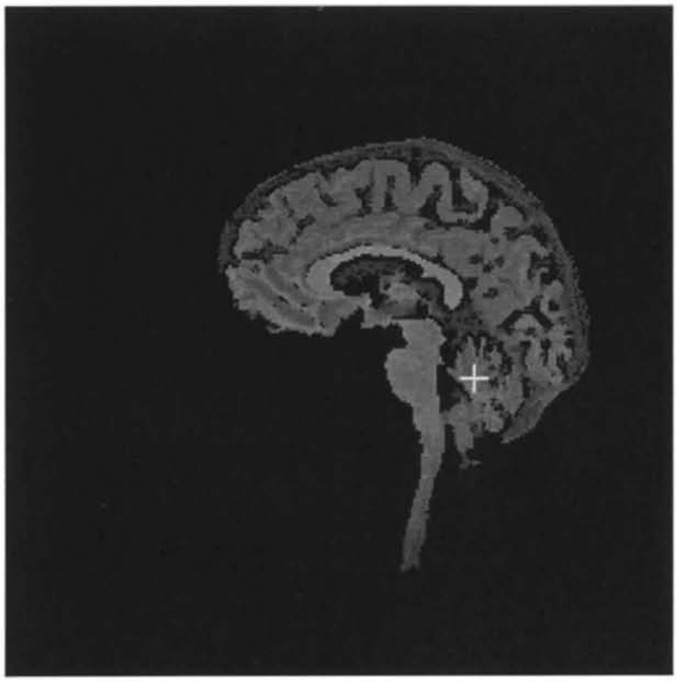
Table 4.2.5.1-1: Pixel Accumulation and Stack Frame Usage (4-connected vs. 8-connected Region-growing)

4.2.5.2 4-connected Simulated Recursion

The recursive method described above needs system stack space for its recursive calls as well as for its calls to incidental functions. As described in section 3.2.3, this iterative version simulates recursion by maintaining its own stack for the storage of passed arguments and return addresses. Since it requires much less memory than the purely recursive method to generate an image with the same settings, it has the potential to be able to perform segmentations of which the recursive method would be incapable. (See section 4.2.5.4, Stack Limitations.)

Since the iterative method is the result of a careful conversion of a recursive method [Tenenbaum90], it is reasonable to expect that (assuming enough stack space and identical settings) the two methods would yield exactly the same image. This is in fact the case for each of the canonical images used in this study (except the image of the entire head, in which case the stack ran out of space before the image was completed). We may verify this outcome in two ways. First, we compare the appearance of the two images. They should, of course, look the same; also, the number of stack frames and pixels accumulated should be equal. Compare figures 4.2.5.2-1 and 4.2.5.2-2 for an example. Second, we may save the output as RAW format files, and compare them byte-for-byte using an operating-system comparison utility.

Output filename: FIG42521.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 2957
Pixels accumulated: 9137

Input filename: SAG-54.SLC

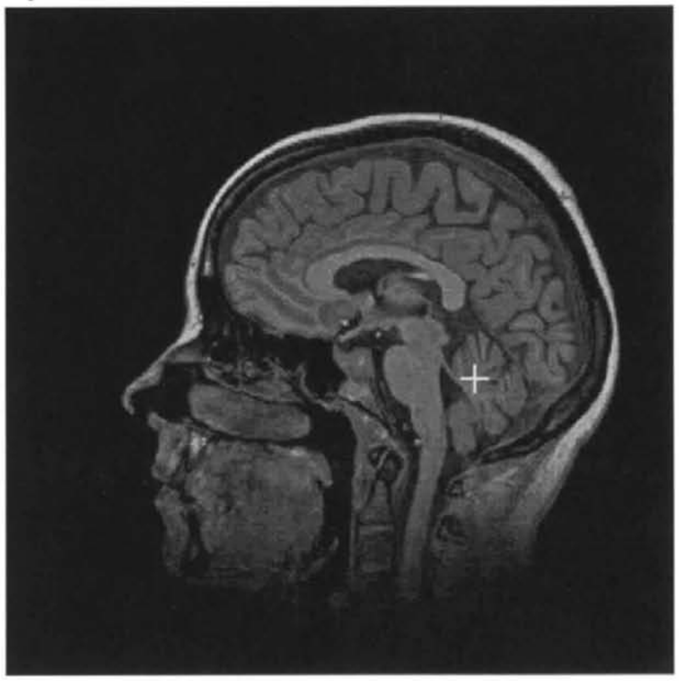
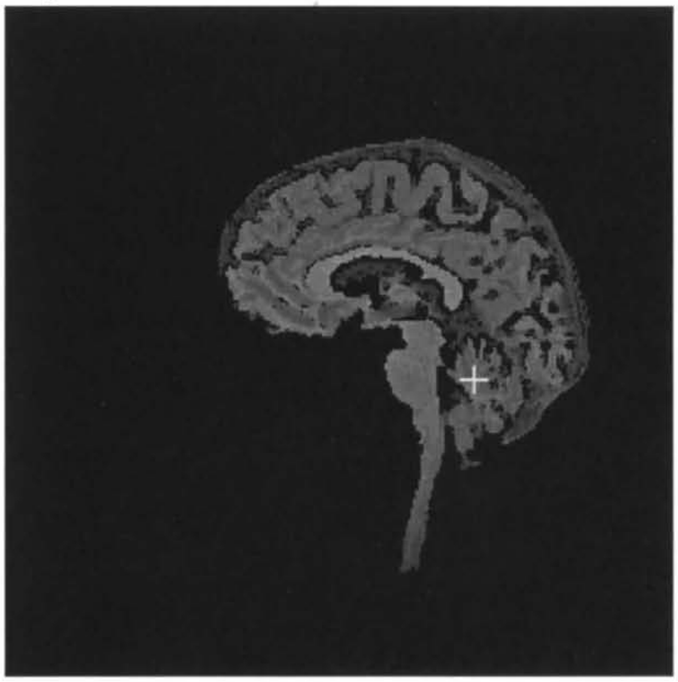


Figure 4.2.5.2-1: The Brain (Recursive Region-growing)

Output filename: FIG42522.EPS



GETCOORD - Bill Bell - Fall 1996

```
x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Iterative
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 2957
Pixels accumulated: 9137
```

Input filename: SAG-54.SLC

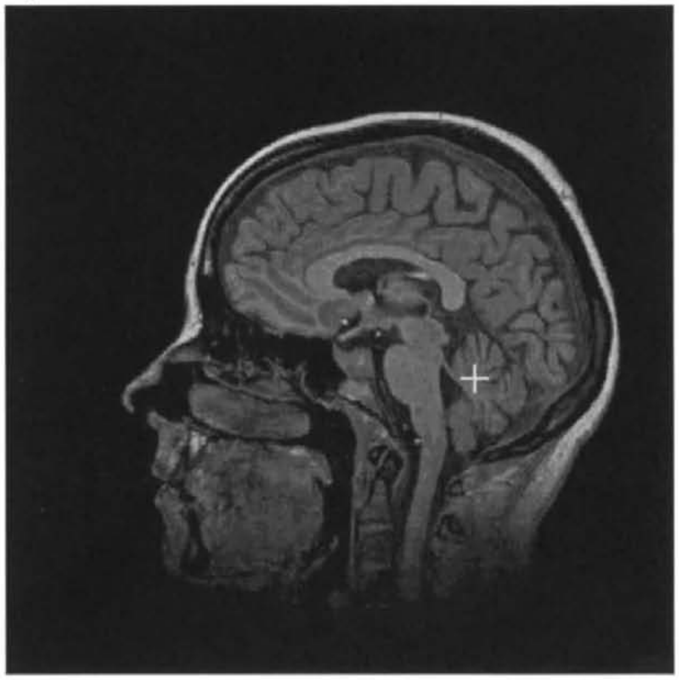


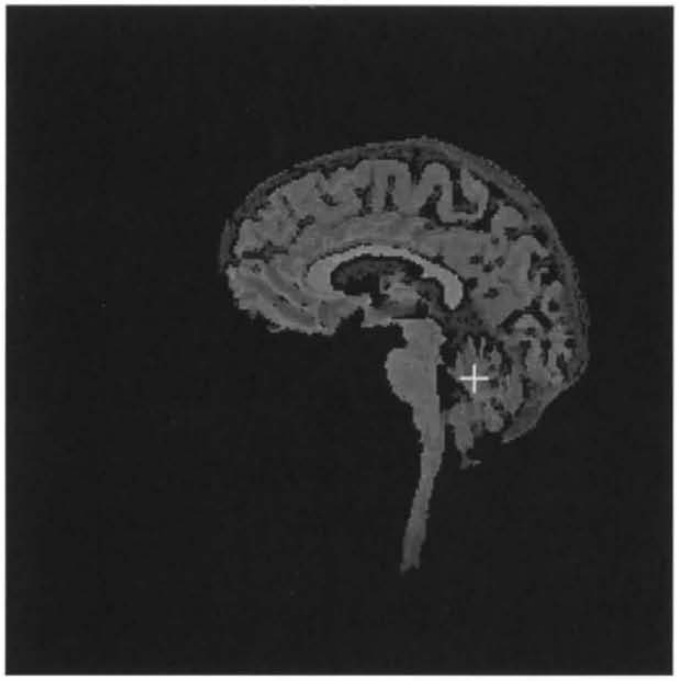
Figure 4.2.5.2-2: The Brain (Iterative, Simulated-recursive Region-growing)

4.2.5.3 Recursive Spanfilling

As explained in section 3.2.4, the spanfilling algorithm used in this study is much more efficient in its use of memory than the other region-growing algorithms used, because it does most of its work (that of adding pixels to a region) in an iterative manner, and relies on recursion only when moving to another span. We also note that the raw images generated with the spanfilling method are exactly the same as those generated by the other methods (except for the image of the entire head, which the spanfill method alone was successful in completing without running out of stack space).

Output filename: FIG42531.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Spanfill
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 331
Pixels accumulated: 9137

Input filename: SAG-54.SLC

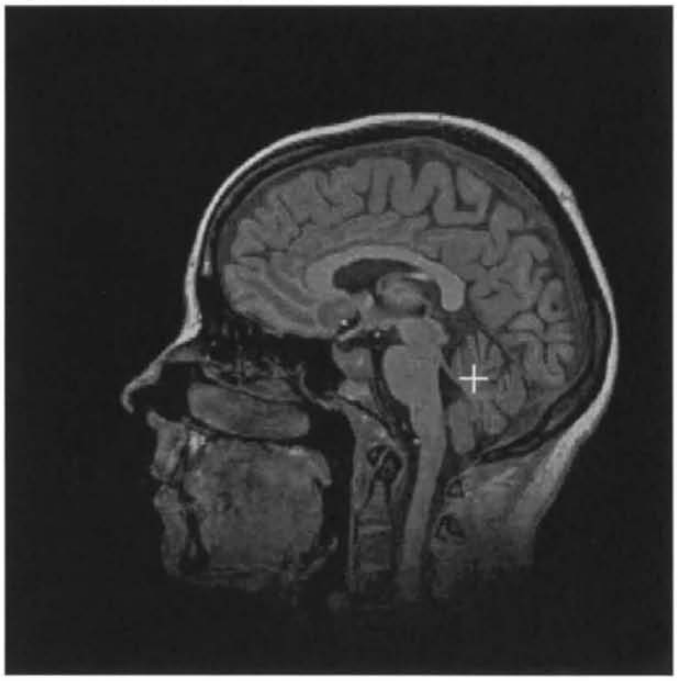


Figure 4.2.5.3-1: The Brain (Spanfill Region-growing)

4.2.5.4 Stack Limitations

The amount of memory available to the program as stack space, and the rate at which it is used, may be determined or limited by the operating system, the program, the compiler used to create the program, and the amount of physical memory in the computer. The implementation of recursive functions in general can be made more efficient by performing incidental functions in-line instead of calling other functions, refraining from declaring local variables, and restricting the number and size of passed arguments.

In order to determine the limits on stack space, we must "stress" the image-generating program, that is, provide it with a set of segmentation arguments which are broad enough to cause the stack to be filled. To this end, we attempt all-inclusive segmentation, in which the gradient threshold is set to the maximum (32767), and the density range limits are set to their extremes (1 and 4095). The seedpoint selected lies at the center of the input image. We would expect, given an unlimited amount of stack space, to view the image in its entirety. The results of this approach may be seen in figures 4.2.5.4-1, 4.2.5.4-2, and 4.2.5.4-3.

In the cases of recursion and simulated recursion, observe that, when the stack becomes full, no further segmentation is possible, and the image cannot be completely generated. However, the spanfill method has no problem in recreating the original image in its entirety. Table 4.2.5.4-1 synopsizes the numbers of stack frames used and pixels accumulated for each region-growing method for each of the canonical images. Since the recursive and simulate-recursive methods failed to completely recreate the entire input image, we regard the number of stack frames (i.e. recursive calls) used by these methods as the maximum available to the program.

Structure	Figure	Recursive		Simulated Recursion		Spanfill	
		Stack Frames	Pixels Accumulated	Stack Frames	Pixels Accumulated	Stack Frames	Pixels Accumulated
corpus callosum	4.3-1	257	354	257	354	30	354
pons	4.3-2	272	640	272	640	29	640
worm of cerebellum	4.3-3	346	663	346	663	24	663
cerebrum	4.3-4	970	2720	970	2720	104	2720
scalp	4.3-5	516	1214	516	1214	133	1214
sup. sag. sinus	4.3-6	666	1770	666	1770	85	1770
brain	4.3-7	2957	9137	2957	9137	331	9137
head	4.3-8	3404	20096*	20000	47351*	129	65280

* incomplete image

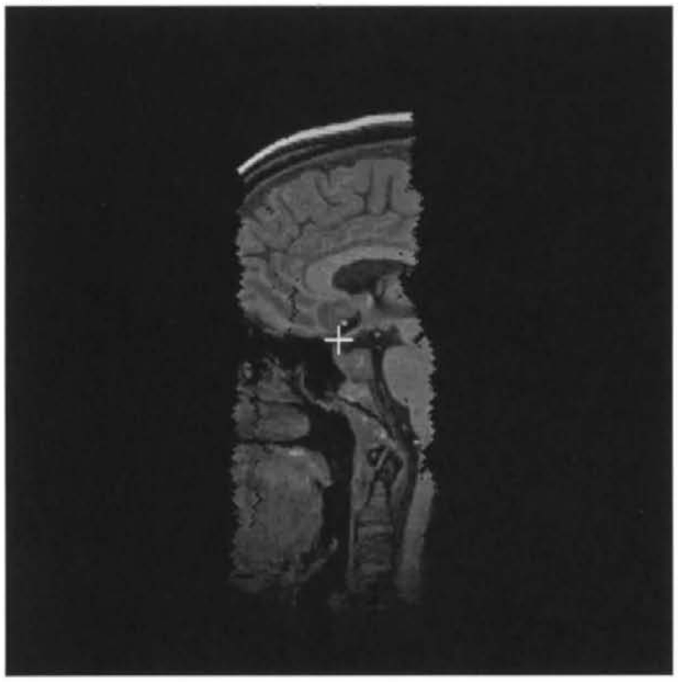
Table 4.2.5.4-1: Pixel Accumulation and Stack Frame Usage for Canonical Images

The recursive method uses the system's stack segment, which, for the GETCOORD program, was set at 55,000 bytes. (For the QuickC compiler, storage required for near data and the stack may not exceed 65,535 bytes. GETCOORD's near data take up the remaining 10,000 or so bytes.) The recursive function needs stack space not only for the many calls to itself, but also for all of the calls made to other functions which are incidental to the recursive function's operation. Observing figure 4.2.5.4-1, it is easy to see why it is the poorest performer in terms of memory usage for image quality gained.

The simulated-recursive method uses the system stack only for calls to incidental functions; otherwise, it uses a stack data structure which is built into, and managed by, the program itself; the maximum size of this structure (20,000 "stack frames" of 3 bytes each) is hard-coded into the program and is essentially limited by the maximum variable size which the compiler will accommodate (65,536 bytes). Since it uses memory much more efficiently, the image it generates (figure 4.2.5.4-2) is much more complete that the one created by its purely recursive cousin; however, in the end, it, too, proves unsatisfactory.

Although the spanfill method's stack maximum is theoretically dependent upon the same factors as the 4-connected recursive method, it clearly uses the stack much less than the other algorithms, and alone is able to completely recreate the entire input image. No single run of the GETCOORD program using the spanfill mode has yet been thwarted by lack of stack space; it seems likely that only the most highly perforated of input datasets (resembling, at the pixel level, something like a checkerboard) would cause such an outcome.

Output filename: FIG42541.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [127, 127, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 1
UDL: 4095
Threshold: 32767
Stack frames used: 3403
Pixels accumulated: 17994

Input filename: SAG-54.SLC

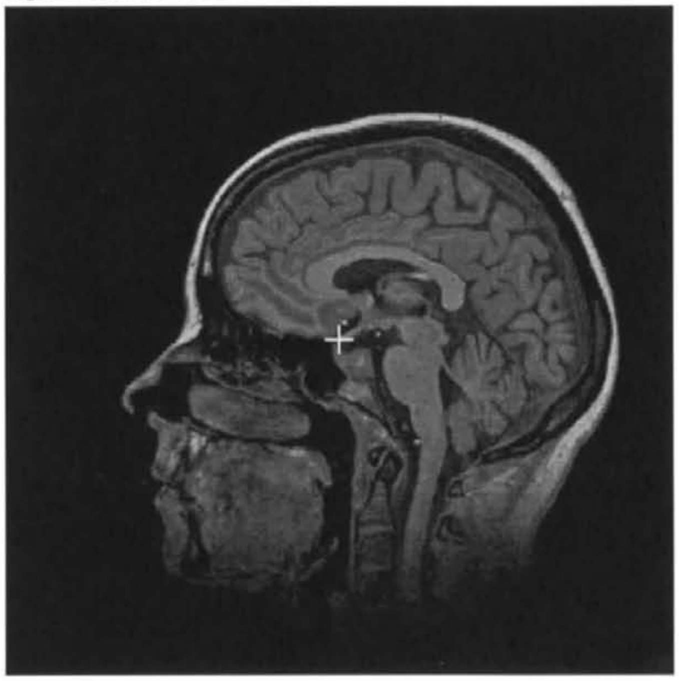
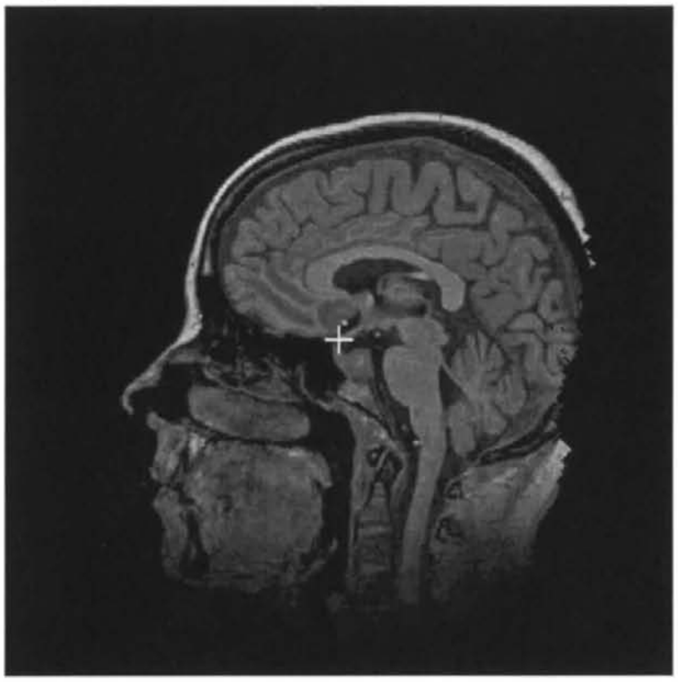


Figure 4.2.5.4-1: All-inclusive Segmentation Using the Recursive Region-growing Method

Output filename: FIG42542.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [127, 127, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Iterative
LDL: 1
UDL: 4095
Threshold: 32767
Stack frames used: 20000
Pixels accumulated: 47351

Input filename: SAG-54.SLC

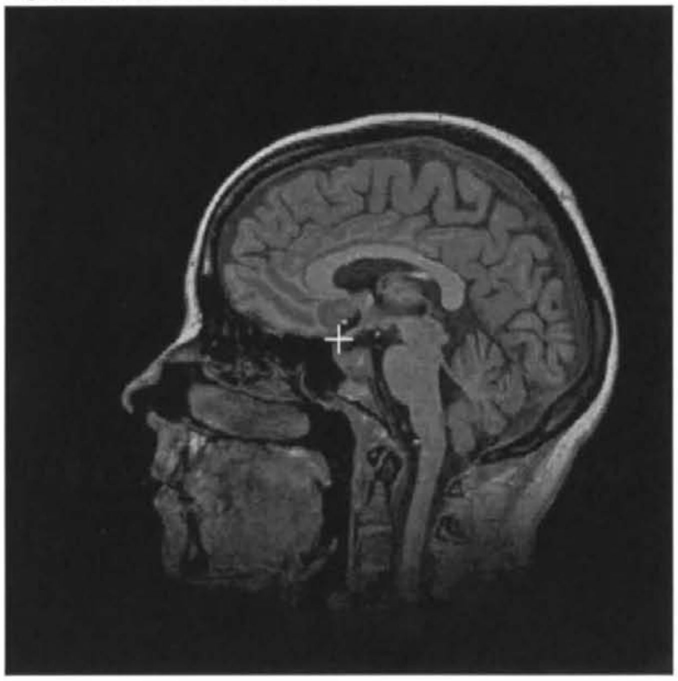
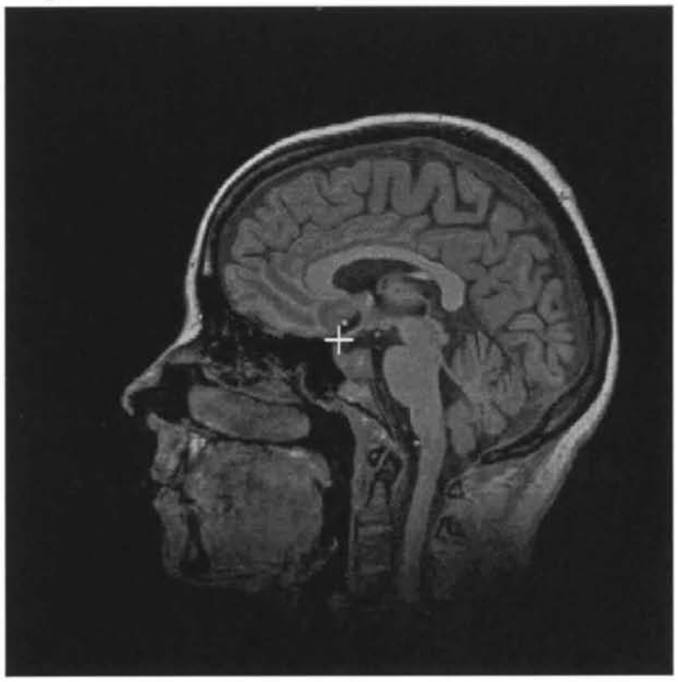


Figure 4.2.5.4-2: All-inclusive Segmentation Using the Simulated-recursive Region-growing Method

Output filename: FIG42543.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [127, 127, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Spanfill
LDL: 1
UDL: 4095
Threshold: 32767
Stack frames used: 129
Pixels accumulated: 65280

Input filename: SAG-54.SLC

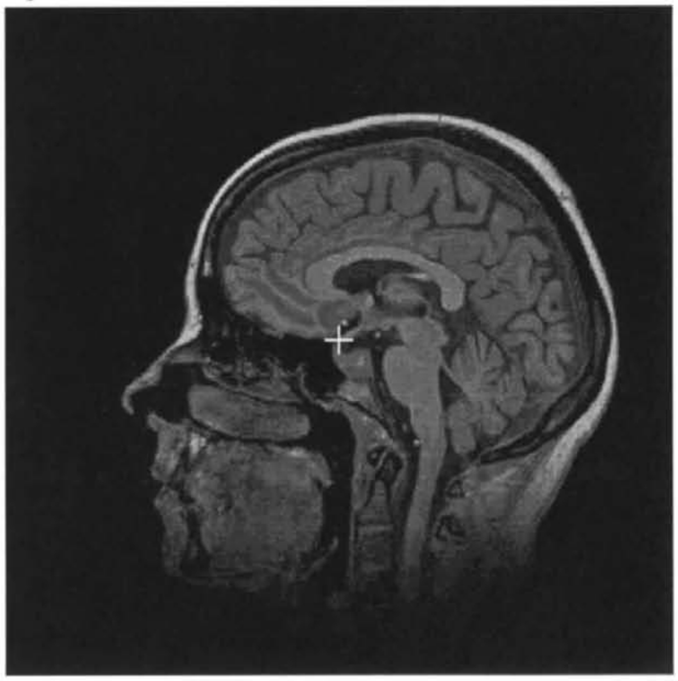


Figure 4.2.5.4-3: All-inclusive Segmentation Using the Spanfill Region-growing Method

4.2.6 Image Resolution

The 3DHEAD VDS represents an array of density data whose dimensions (in voxels) are 256 x 256 x 109. The physical dimensions of the area represented are roughly 36 cm x 36 cm x 15 cm.[3] It would appear that the resolution of the original dataset, roughly 7 pixels/cm (.7 pixels/mm), is relatively coarse, and may not admit of high-resolution segmented images of very small structures. Barring the use of a higher-resolution MRI scanning machine, some form of interpolation may be necessary.

4.2.7 Run Times

Before moving on to the next section, it is appropriate to say a few words about the time required to create the two-dimensional images presented here. These images were generated by a DOS-based program running on a 50-MHz 486 computer. In each case except for all-inclusive segmentation, run times were less than two or three seconds. For small structures such as the corpus callosum, the run time for image generation was essentially nil. The run time for an all-inclusive image using recursive region-growing finished at three seconds, although this represents the time required for the stack to fill up; the image created is incomplete. Likewise, all-inclusive region-growing using simulated recursion ran in 11 seconds, resulting in an incomplete image due to stack saturation. The spanfill method needed only eight seconds to create a complete image.

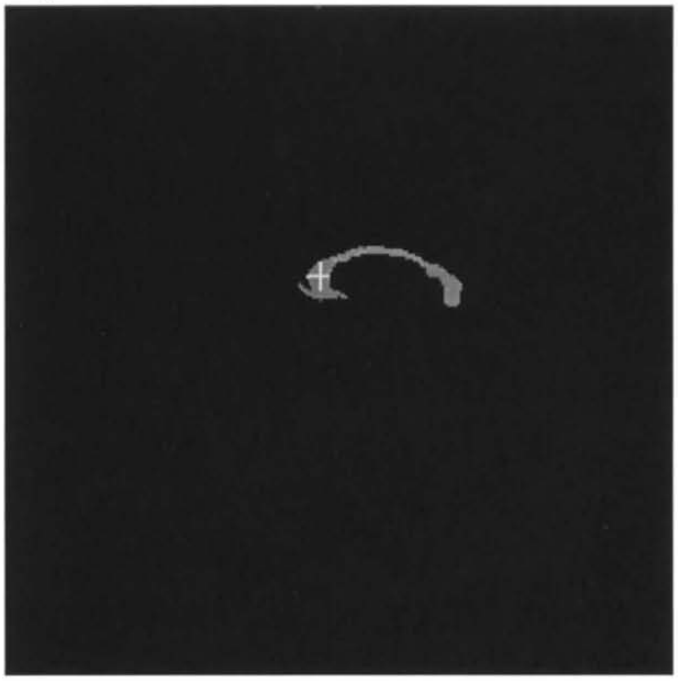
[3] These figures are based on a comparison of the 54th sagittal slice of the 3DHEAD VDS with measurements of the author's own skull: from the bridge of the nose to the back of the skull, about 23 cm; and from left temple to right temple, about 18 cm.

4.3 Canonical Images

The notion of a "canonical" image is taken to mean "a critical standard" by which other images may be judged or compared. To be sure, image quality is in the eye of the beholder; the reader is invited to derive better images than those displayed here. In any event, these images will provide a useful basis for assessing the effectiveness of different segmenting strategies.

Output filename: FIG431.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [120, 103, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 1200
UDL: 2000
Threshold: 1000
Stack frames used: 257
Pixels accumulated: 354

Input filename: SAG-54.SLC

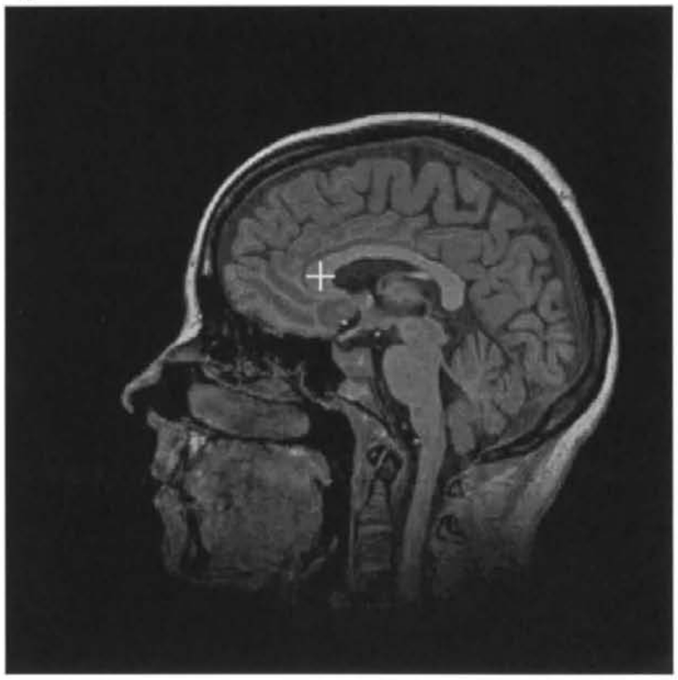
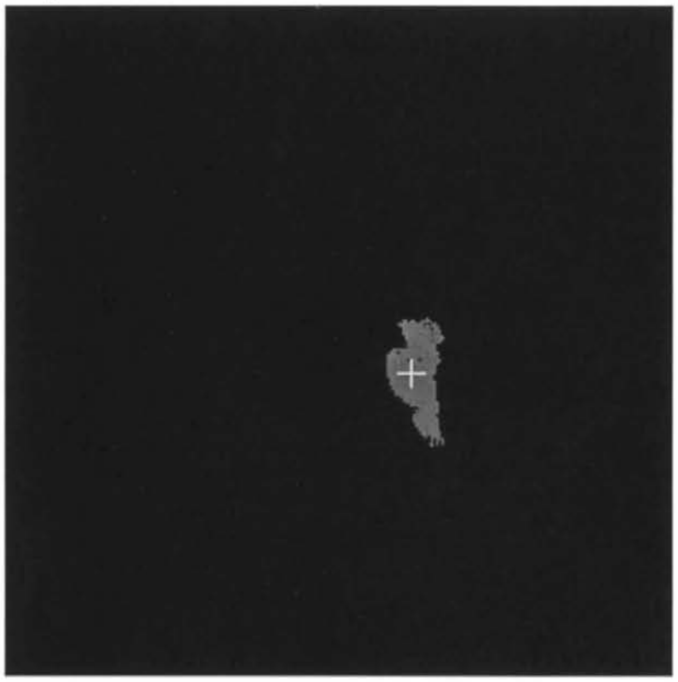


Figure 4.3-1: Canonical Image of the Corpus Callosum

Output filename: FIG432.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [155, 140, 54]

Gradient-approximation method:
    Prewitt

Region-growing method:
    Recursive

LDL: 1100

UDL: 1500

Threshold: 800

Stack frames used: 272

Pixels accumulated: 640

Input filename: SAG-54.SLC

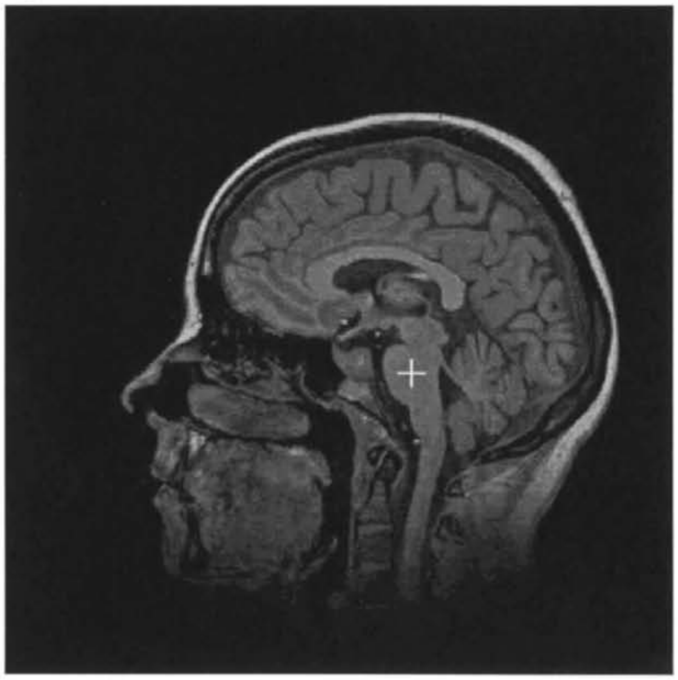
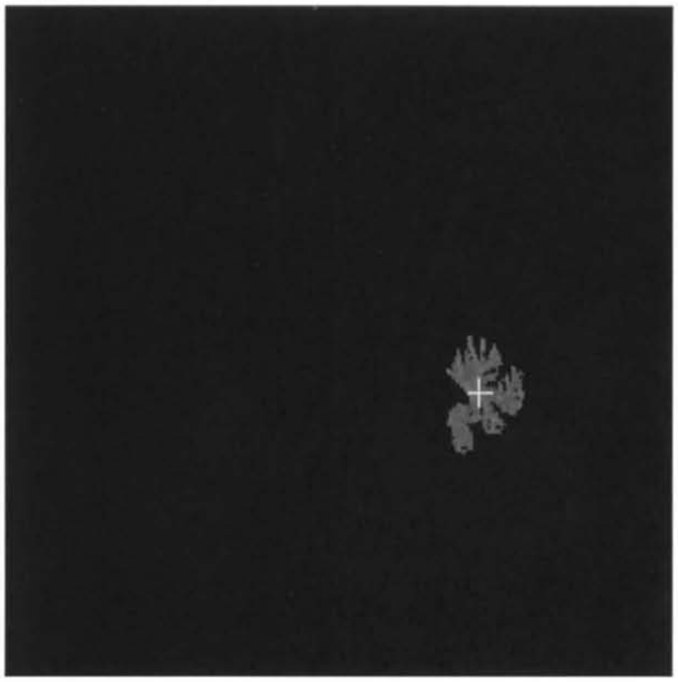


Figure 4.3-2: Canonical Image of the Pons

Output filename: FIG433.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [181, 147, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 900
UDL: 1800
Threshold: 1800
Stack frames used: 346
Pixels accumulated: 663

Input filename: SAG-54.SLC

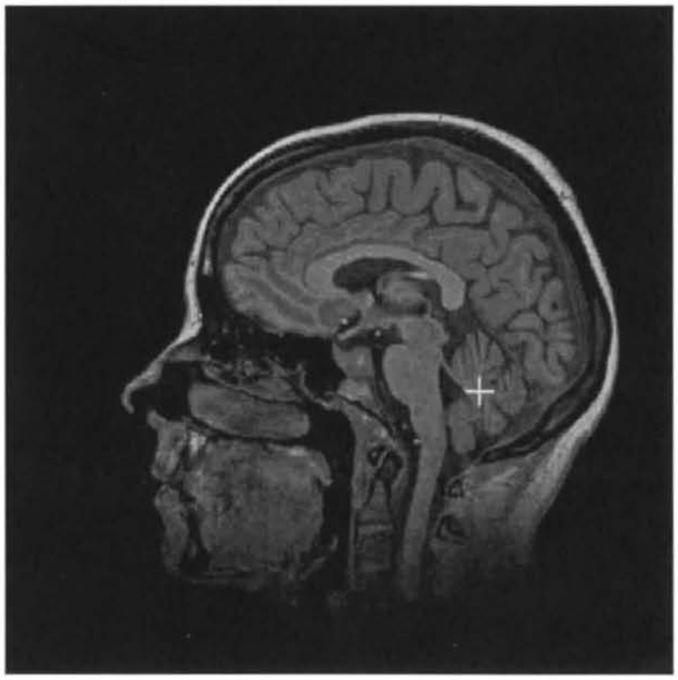
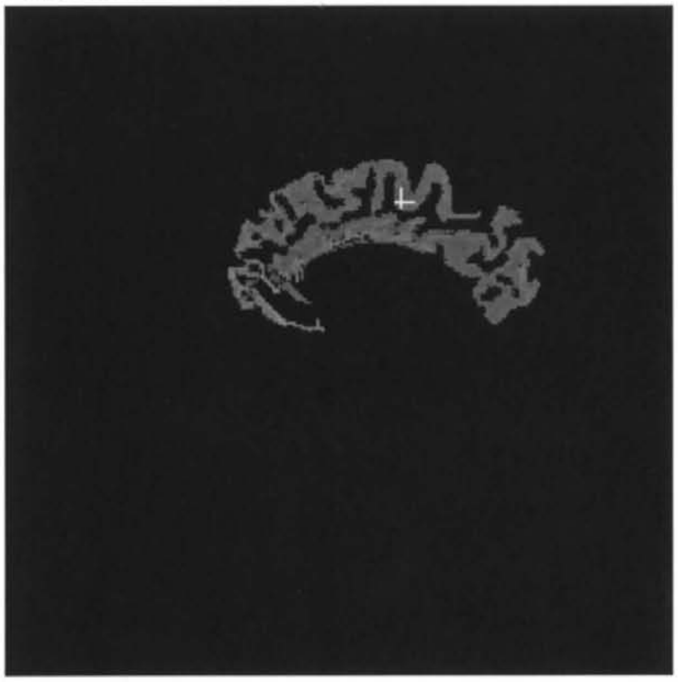


Figure 4.3-3: Canonical Image of the Worm of Cerebellum

Output filename: FIG434.EPS



GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [151, 75, 54]
Gradient-approximation method:
Prewitt
Region-growing method:
Recursive
LDL: 900
UDL: 2000
Threshold: 900
Stack frames used: 970
Pixels accumulated: 2720

Input filename: SAG-54.SLC

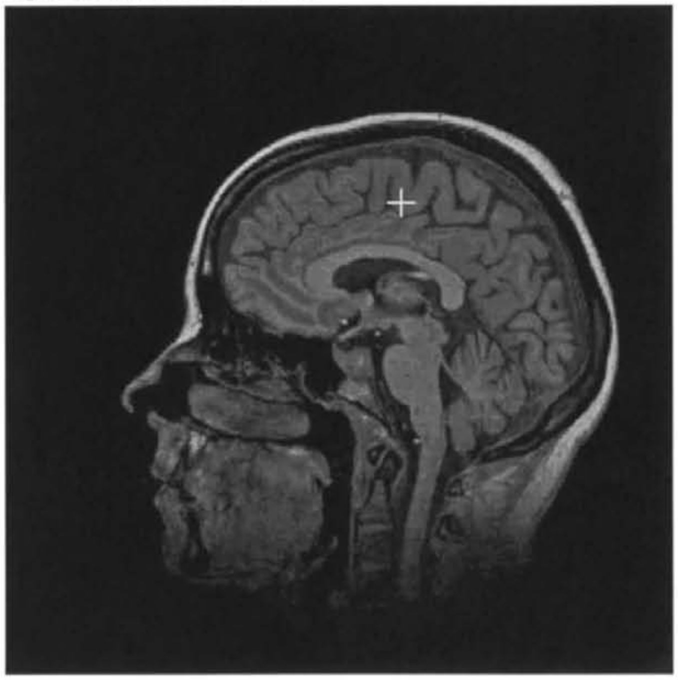
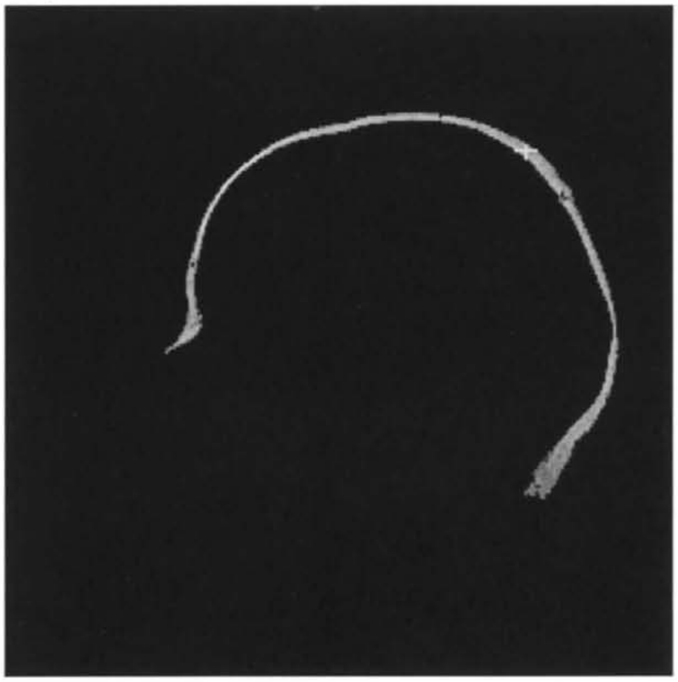


Figure 4.3-4: Canonical Image of the Cerebrum

Output filename: FIG435.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [199, 55, 54]

Gradient-approximation method:

Prewitt

Region-growing method:

Recursive

LDL: 1200

UDL: 2800

Threshold: 8000

Stack frames used: 516

Pixels accumulated: 1214

Input filename: SAG-54.SLC

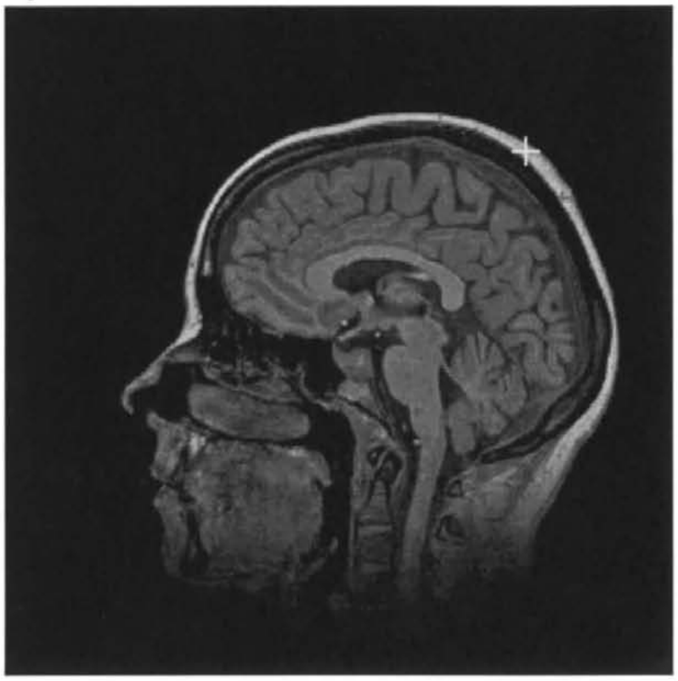


Figure 4.3-5: Canonical Image of the Scalp

Output filename: FIG436.EPS

GETCOORD - Bill Bell - Fall 1996

x-y-z coordinates: [200, 75, 54]

Gradient-approximation method:
Prewitt

Region-growing method:
Recursive

LDL: 100

UDL: 900

Threshold: 1400

Stack frames used: 666

Pixels accumulated: 1770

Input filename: SAG-54.SLC

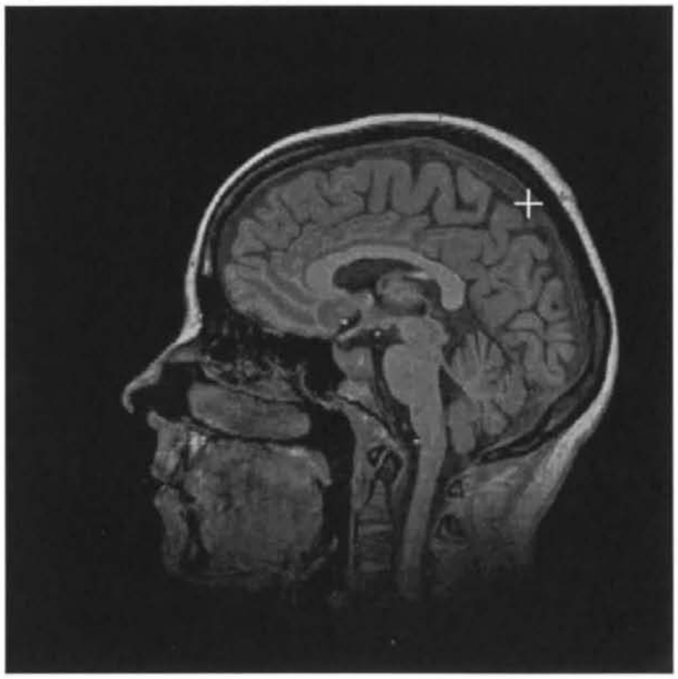
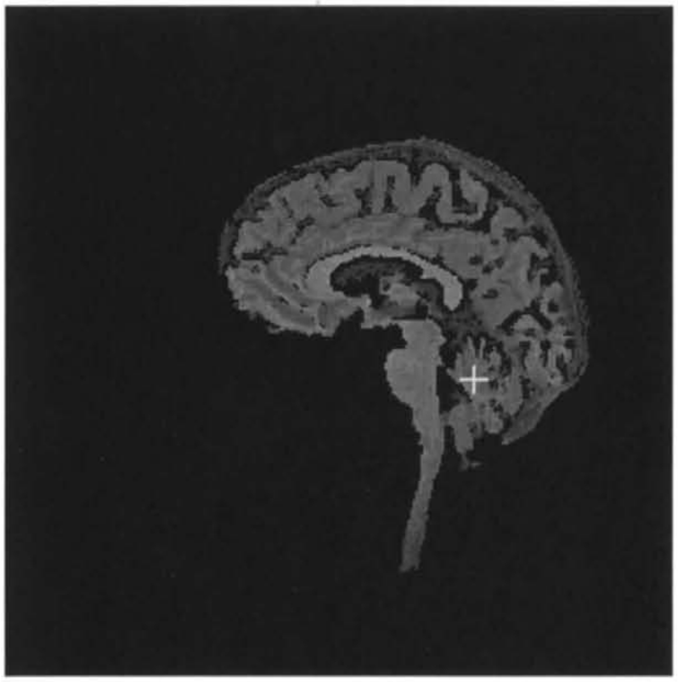


Figure 4.3-6: Canonical Image of the Superior Sagittal Sinus

Output filename: FIG437.EPS

GETCOORD - Bill Bell - Fall 1996



```
x-y-z coordinates: [179, 142, 54]
Gradient-approximation method:
    Prewitt
Region-growing method:
    Iterative
LDL: 500
UDL: 2000
Threshold: 1500
Stack frames used: 2957
Pixels accumulated: 9137
```

Input filename: SAG-54.SLC

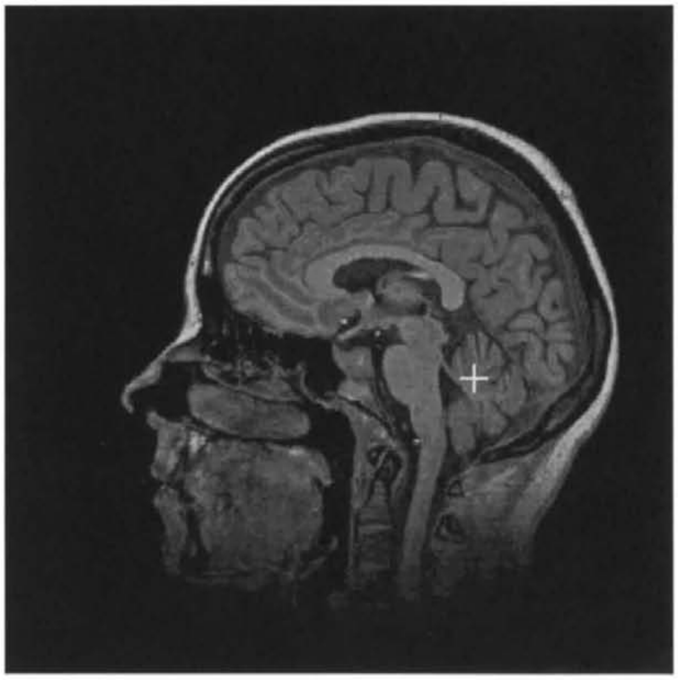
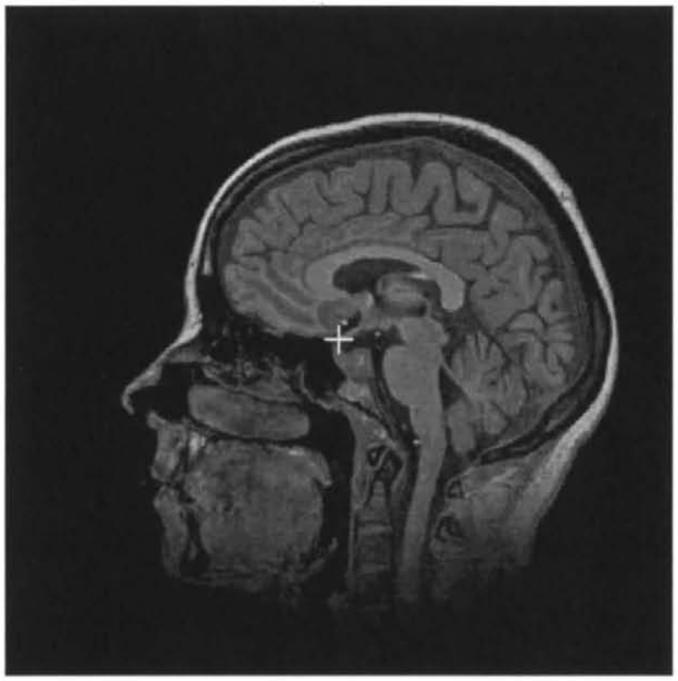


Figure 4.3-7: Canonical Image of the Brain

Output filename: FIG438.EPS

GETCOORD - Bill Bell - Fall 1996



x-y-z coordinates: [127, 127, 54]

Gradient-approximation method:
Prewitt

Region-growing method:
Iterative

LDL: 200

UDL: 3000

Threshold: 10000

Stack frames used: 13395

Pixels accumulated: 25358

Input filename: SAG-54.SLC

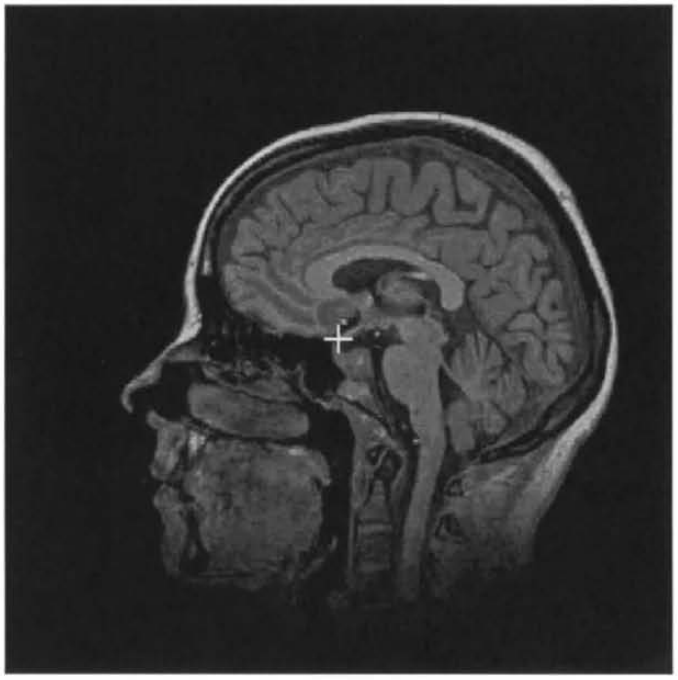


Figure 4.3-8: Canonical Image of the Head

CHAPTER 5

IMPLEMENTATION IN THREE DIMENSIONS

The MRI3D program was designed to perform segmentation on a volumetric data set (VDS) of hydrogen-ion density data gathered from a magnetic-resonance imaging scanner. Besides the VDS itself, MRI3D uses a configuration file (in ASCII format) which tells it how to segment the data. This configuration file contains the following information:

- seedpoint (x, y, z) coordinates
- region-growing algorithm
- lower- and upper-density limits
- gradient threshold
- gradient-approximation operator
- Z-buffer view
- Z-buffer file format and name
- VDS input and output file names
- illumination method
- Phong-shading settings

MRI3D may be set up to run in interactive mode or as part of a batch process (using the

"-b" option); due to the amount of data to be processed, the runs of this program often take a very long time to complete, and the ability to batch the jobs may make it more convenient to use. If being run in interactive mode, MRI3D allows the user to perform dynamic histogram-stretching in order to improve the contrast of the resulting image. The command-line usage for MRI3D is:

```
        MRI3D configfilename logfilename [-b]
```

When complete, MRI3D will have created a status log file, containing various statistics pertaining to the run of the program, as well as any errors which may have occurred. It will also, depending upon the wishes of the operator, have created 256-grayshade RAW, PostScript, or Encapsulated PostScript files of the Z-buffer which was generated. The RAW files' dimensions are 256x256 pixels and are readily convertible to other formats, as well as are able to be read directly by the MIDTERM image-processing program [Bell94A]. The PostScript files have embedded within them all of the settings pertaining to the run which they represent.

MRI3D, a DOS-based program, requires 16-color VGA and 640 kb of conventional memory to run. In addition, it requires at least 1 Mb of extended memory in order to maintain an XMS-based stack for certain region-growing algorithms; as much as 4 Mb of XMS is optimal. Furthermore, the input VDS file used for the experiments described here needs 14 Mb of disk space; and the spatial-occupancy enumeration method used to store output VDS data requires an additional 14 Mb. The intense disk input/output (I/O) activity which takes place during segmentation makes this a very slow and noisy program; a typical run could take many hours. Therefore, a large virtual disk (32 Mb) was created in RAM memory, which made run times tolerably short (as much as two to three hours in some cases).

MRI3D is made up of several component files; each deals with a different aspect of the overall program. In addition, two shareware libraries were used in the creation of MRI3D: VSA256, a VESA video graphics library, and XMSIF, a library of extended memory routines (see Chapter 2).

5.1 Thresholding Methods

As with GETCOORD, MRI3D allows the user to apply limits, or thresholds, to a segmentation run with respect either to hydrogen density or to the density gradient, or to both. The principles are the same as previously described, the only significant difference being that, when computing density gradients, we must extend our notion of the convolution kernel into three dimensions. In this project, we describe three types of gradient-approximation kernels: the Six-neighbor, the Frei-Chen, and the Pseudo-Sobel kernel. Whereas GETCOORD's approach was to compute the density gradient of a 3x3 matrix of density data in the direction of both the x- and y-axes , MRI3D's kernels evaluate a 3x3x3 matrix of density data and approximate the gradient in the direction of the z-axis as well. As before, these derivative operators are designed to yield a small number in response to volumes over which the change in density is small, and a high number in volumes demonstrating a large change in density (and, for our purposes, probably a change in tissue type).

For each 3x3x3 cube of data whose center lies at (x, y, z), MRI3D stores the density data in a 28-element array (the zeroth element is unused, for convenience). Array subscripts are assigned to the voxels of the data cube in a consistent manner, as shown in figure 5.1.1-1.

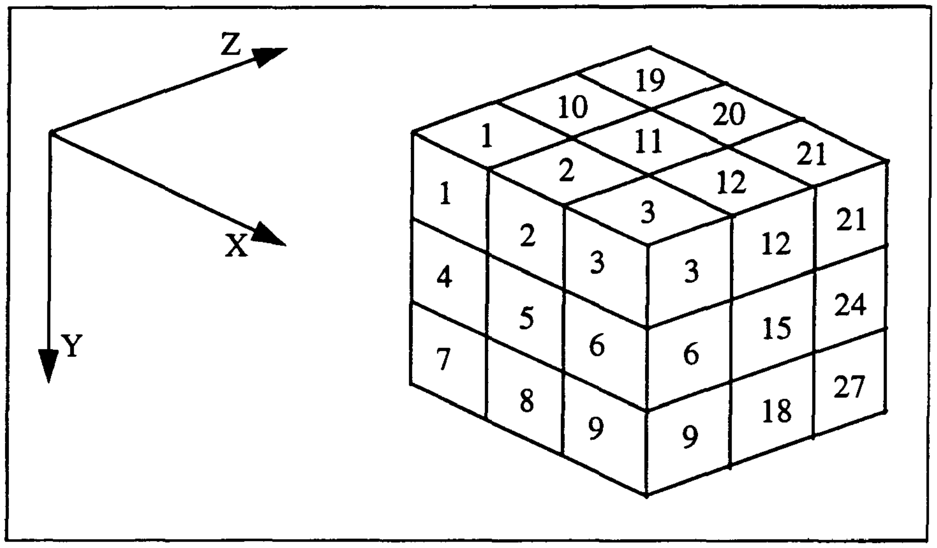


Figure 5.1.1-1: 3x3x3 Data Cube: Array Subscript Assignment

5.1.1 The Six-neighbor Kernels

The Six-neighbor method, described in [Kippenhan94], derives its name from the fact that it assesses the values of the six voxels which are neighbors in each axial direction to the voxel at the center of the cube. It uses the following three equations to compute the gradient in each direction:

$$\nabla_x = V_{(x-1,y,z)} - V_{(x+1,y,z)}$$

$$\nabla_y = V_{(x,y-1,z)} - V_{(x,y+1,z)}$$

$$\nabla_z = V_{(x,y,z-1)} - V_{(x,y,z+1)}$$

The magnitude of the gradient of the entire cube is approximated by adding the absolute values of the three axial gradients, thus:

$$G = |\nabla_x| + |\nabla_y| + |\nabla_z|$$

During segmentation, we compare the value of G at each voxel under consideration with the gradient threshold specified in the configuration file. Figure 5.1.1-2 shows the nature of the Six-neighbor method.

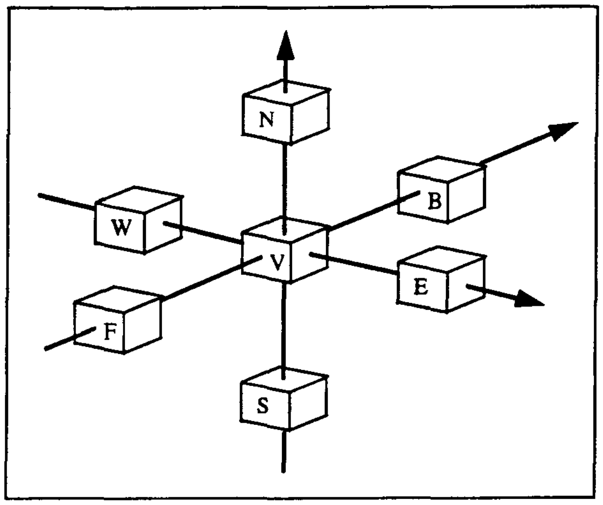


Figure 5.1.1-2: 6-neighbor Gradient Approximation

5.1.2 The Frei-Chen Kernels

The Frei-Chen kernels (also known as the 26-neighbor kernels) and described in [Kippenhan94], [Zucker81], and [Russ95], take into account all 26 of the voxels which are neighbors to the center voxel under consideration. The computations for the gradient are not conceptually complex; for the sake of brevity, we display only the equation for the gradient in the direction of the x-axis. We also introduce a way of computing the gradient which deviates from the notation used in [Kippenhan94], but which is algebraically equivalent and involves less floating-point arithmetic (always a good thing in computing!):

Let a = sqrt(3) / 3, b = sqrt(2) / 2, c = 1, and

let array M[27] contain the voxel density data, as described. Then

$$
\begin{aligned}
Gx = \ | & ((a * (M[3] + M[9] + M[21] + M[27])) + \\
& (b * (M[12] + M[18] + M[6] + M[24])) + \\
& (c * M[15])) - \\
& ((a * (M[1] + M[7] + M[19] + M[25])) + \\
& (b * (M[10] + M[16] + M[4] + M[22])) + \\
& (c * M[13])) \ |
\end{aligned}
$$

In the case of each axis, we consider the sum of densities (each multiplied by a constant which is related to the voxel's position relative to the center of the cube) of all voxels in the plane on one side of the center voxel, and the similar sum of the plane on the opposite side. The absolute value of the difference between these sums yields the gradient for this axis (see figure 5.1.2-1). The sum of all three axial gradients yields the approximation of the gradient for the entire cube.

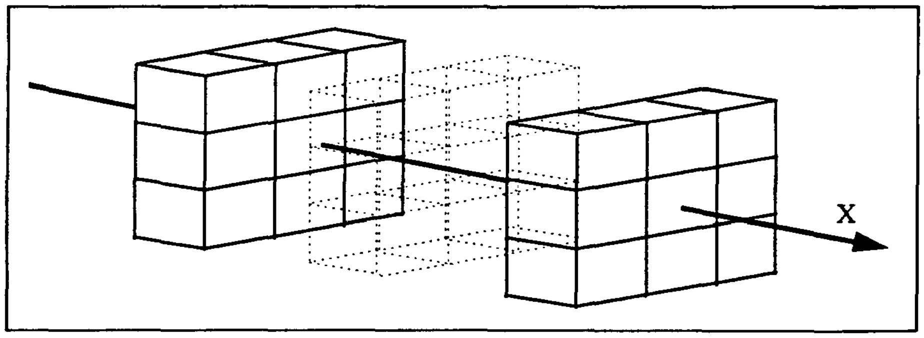


Figure 5.1.2-1: Frei-Chen (26-neighbor) Gradient Approximation (x-axis)

5.1.3 The Pseudo-Sobel Kernels

The Pseudo-Sobel kernels are loosely based on the Sobel kernels described earlier. They are computed in precisely the same way as the Frei-Chen kernels, except that the values of a, b, and c are 1, 2, and 4, respectively. As with the Sobel kernels, the idea is to reduce noise in the resulting image by doubling the center terms of the kernel.

5.2 Region-growing Methods

Most of the techniques described here for three-dimensional region-growing are simple extensions of the previously described algorithms for two-dimensional region-growing. We do, however, take advantage of the adjacency of voxels in the same plane and on the same row in order to reduce the number of Read operations, therefore improving the program's efficiency.

5.2.1 Recursive 6-connected Region-growing

The plain recursive region-growing algorithm used here is essentially the same as the one described in Chapter 3 for use in two dimensions, except that, in addition to the four recursive calls made for voxels to the north, south, east, and west (in the same plane), we make calls for the voxels to the "front" (the plane which precedes this one) and to the "back" (the plane which follows).

5.2.2 Recursive 26-connected Region-growing

This region-growing algorithm is an extension of the 6-connected algorithm which makes 26 recursive calls to all voxels which are neighbors of the voxel under consideration. It is analogous to two-dimensional 8-connectedness.

5.2.3 Iteration and Simulated Recursion

In order to make it possible to make more recursive calls without running out of stack space, we implement simulated recursion in three dimensions here as we did in two dimensions, as described in Chapter 3. However, the need for stack space in three dimensions is exponentially greater; we therefore employ extended memory (XMS) for our self-maintained stack. This concept will be discussed more fully in section 5.3.1.3.

5.2.4 A Stacked Spanfill Algorithm

The 2-D spanfilling algorithm described previously obtained a seedpoint (x, y) and filled the entire row y to the left and right so long as its pixels met the criteria of density and gradient for membership in the segmented set. Rows above and below this seedspan were treated similarly in subsequent recursive calls to the spanfilling function.

In order to extend this idea into three dimensions, we extract a single slice (containing two of the three orthogonal axes) through the input VDS. This slice is called the seedslice. We consider the coordinates of the rightmost voxel in each span in the seedslice to represent a seedpoint for a slice of the input VDS which is oriented perpendicularly to the seedslice. (These seedpoints are stored in an array in memory.) After using the seedpoints to spanfill the perpendicular slices in which they reside, we store the resulting segmented images in the output VDS. The values of the voxels nearest to the observer (as measured by the value of their z-coordinates) are stored in a Z-buffer for further image processing.

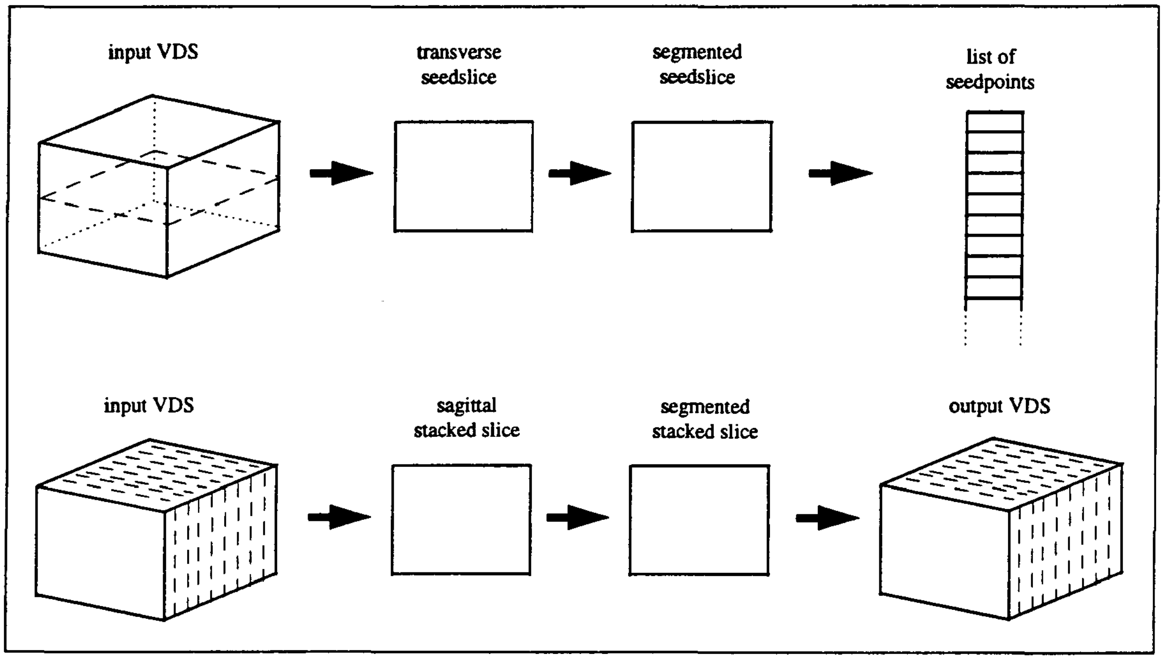


Figure 5.2.4-1: The Stacked Spanfill Algorithm Concept

The selection of orientation of the seedslice (and consequently of the stacked slices), in conjunction with the ordering of the VDS data, will have an effect on the runtime of the program. To read any slice of the VDS involves some number of Seek and Read operations: the Seek positions the file pointer to the correct offset into the VDS; the Read reads data which are found at that offset. A Seek operation may involve mechanical movement of a hard drive's read/write heads, something which takes a relatively long time in computer terms. Whether one is interested in reading one byte or a thousand, a Seek is always necessary; but once a Seek is performed, the amount of time required to read a large number of bytes or a small number is of little consequence. The number of Seeks and the number of bytes of data acquired with each Read, and the proportion of the one to the other, directly affects the amount of time needed to completely read a slice of data.

It would appear, then, that, in order to minimize the amount of time needed to fully process the input VDS, one would choose to read stacked slices in an order and orientation which both minimizes the number of Seeks and maximizes the number of bytes read for each Seek. (We don't care much about these considerations with respect to the seedslice, of which there is only one.) Let us observe quantitatively the different combinations we might consider, as in figure 5.2.4-2:

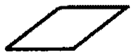
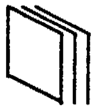
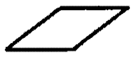
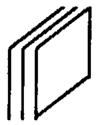
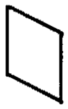
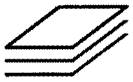
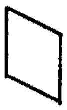
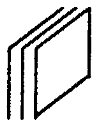
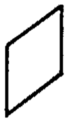
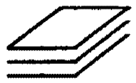
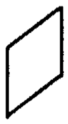
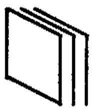
seedslice:			stacked slices:					
orientation	axes	seeks/slice	orientation	axes	seeks/slice	slices	stacked-slice seeks	total seeks
	x-z	109		x-y	1	109	109	218
	x-z	109		y-z	27904	256	7143424	7143533
	x-y	1		x-z	109	256	27904	27905
	x-y	1		y-z	27904	256	7143424	7143425
	y-z	27904		x-z	109	256	27904	55808
	y-z	27904		x-y	1	109	109	28013

Figure 5.2.4-2: Efficiency of Stacked Spanfill Slice-Reading Order

From figure 5.2.4-2, it is evident that the best strategy is to read a transverse seedslice, involving 109 Seeks and Reads of 256 integers each, and sagittal stacked slices, at 109 Seeks and Reads of 65,536 integers each. We take advantage of the natural storage order of the data in the VDS to optimize processing time.

5.3 Data Representation and Storage Methods

5.3.1 Spatial Occupancy Enumeration Using 3-D Arrays and Disk Files

The Spatial Occupancy Enumeration (SOE) method of storing data has already been adequately described in Chapter 3. As noted, SOE is very simple to implement; however, it was discerned early on that the number of Seek operations involved a very long time when working with VDS files stored on a hard disk. Therefore, it was decided to establish a large (32 Mb) virtual disk in memory: a "RAM disk." This action alone reduced runtimes by an order of magnitude without any change in the program's code (and also saved a great deal of wear and tear on the hard drive!).

5.3.2 The Z-buffer

Briefly mentioned in Chapter 1, the Z-buffer is a two-dimensional array which records, at buffer coordinates (x, y), the z-coordinate of that voxel V at (x, y, z) whose z-value is the smallest encountered thus far during the processing of the input VDS. For example, given two voxels V_1 at (x_1, y_1, z_1) and V_2 at (x_1, y_1, z_2) (both members of the segmented region), if z_1 is less than z_2, then V_1 is considered to be closer to the user's viewpoint than V_2. When the entire VDS has been processed, the Z-buffer holds what amounts to altitude data for the segmented region. Its contents may be saved by MRI3D to RAW, PostScript, and Encapsulated PostScript formats; and may also be stored in any of the three orthogonal views (sagittal, coronal, and transverse).

The input VDS called 3DHEAD contains 109 sagittal slices of density data (or, if you prefer, 256 coronal or transverse slices, although those are not the natural order of storage). Through some trickery involving manipulation of the video graphics board's color registers, we have squeezed out 189 gray- or pseudo-grayshades with which to color the pixels in the Z-buffer (thanks are due to Spyro Gumas, the creator of the VSA256 VESA library [see Chapter 2], for his idea). Some image-processing of the raw grayshade data may be needed to improve the brightness and contrast of the resulting image; to this end, a simple histogram-stretching feature is built in to permit the user to adjust the picture before saving it.

5.3.3 The Stack

The stack, its nature and the principles of its use, were discussed in Chapters 3 and 4. The recurring problem with stack-dependent algorithms is the fact that they are a finite resource, and if they are not large enough to accommodate all of the recursive calls made during segmentation, then the picture may appear incomplete. We attempted to circumvent the problem in section 3.2.3 by allocating and manipulating our own stack in conventional memory (i.e. under 640 kb), thus expanding the limits of tractable segmentations.

In three dimensions, however, the problem of insufficient stack space is much more severe. Therefore, we take advantage of the presence of physical memory beyond the 1 Mb boundary (in fact, up to 16 Mb) by making calls to functions which allocate extended memory (XMS). It is in XMS space that we create and maintain our stack for use with the same simulated-recursion approach described in section 3.2.3. For this project, the XMSIF library of C-interface routines (see Chapter 2) was used to allocate a handle to a block of extended memory, to write to and read from this block, and to free it when finished.

5.4 Rendering Techniques

In the following sections, we discuss the depth-cueing and ray-tracing/Phong-illumination methods of image rendering.

5.4.1 Depth-cueing

As mentioned in Chapter 1, depth-cueing provides a correlation between the altitude or proximity of a voxel and its perceived brightness. In MRI3D, we must ensure when mapping altitudes to grayshades that the correct constant of proportionality is used for the orientation of the view that was selected. Also, if there is little difference between the altitudes of adjacent structures (and therefore of their voxels' brightness), we may find it desirable to exaggerate those differences through the use of histogram-stretching in order to create a more pleasing and informative picture.

5.4.2 Ray-tracing and Phong Shading

The notion of ray-tracing to determine visible surfaces has already been treated in section 1.5.2. In this section, we describe specifically how such surfaces are determined, how the rays are computed, and how they are illuminated.

To begin with, we have a two-dimensional Z-buffer of altitude data. We traverse this array, row by row and column by column, and, for each pixel (at coordinates (x, y), imagine a small triangular "patch" whose vertices are at (x, y) (A), (x+1, y) (B), and (x, y+1) (C), as in figure 5.4.2-1).

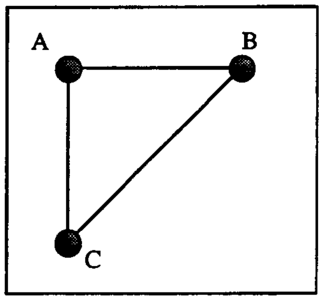


Figure 5.4.2-1: The Triangular Patch

We derive the (x, y, z) coordinates of each vertex from the row, column, and array element contents respectively; and use this information to compute some vectors:

(1) The line from A to C creates vector *X*; the line from A to B, vector *Y*. These two vectors describe a plane which includes all three vertices of the patch.

(2) Vectors *N* and N_u, derived from the cross-product of *X* and *Y*, are normal and unit-normal vectors of the patch, that is, vectors which are perpendicular to the patch.

(3) L_u is the unit vector of some point light source (whose coordinates are specified in the configuration file).

(4) R_u is the vector of light reflected off the patch from the point light source. (The angle between N and L is equal to the angle between N and R.)

(5) S, the sight vector, is the vector from the viewer's eye to the patch. (The z-component of this vector is an arbitrarily chosen number which represents the distance between the patch and the viewer.)

Now that all the vectors have been computed, we determine if the patch is visible to the viewer by computing the angle between vector R (the vector of reflected light) and vector S (the sight vector). If the angle is greater than or equal to 90 degrees, then the patch reflects no light towards the viewer and is therefore invisible (figure 5.4.2-2).

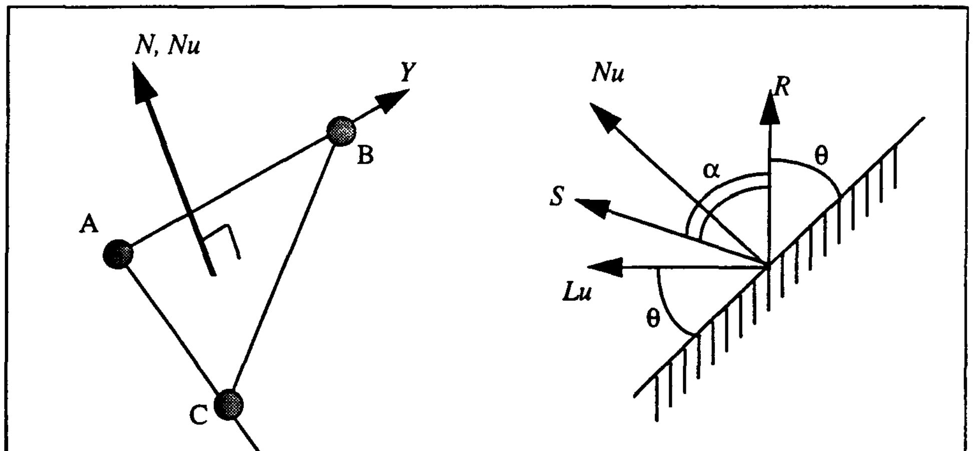


Figure 5.4.2-2: The Normal Vector; Normal, Light, Reflected, and Sight Vectors

If the patch <u>is</u> visible, then we compute the intensity of the reflected light according to the Phong illumination model [Foley90]:

$$I_r = I_a * K_a + (I_d * K_d * (N_u \bullet L_u)) + (I_d * w(\theta) * (\cos^n \alpha))$$

where:

I_r = reflected light intensity
I_a = ambient light intensity ($0 <= I_a <= 255$)
I_d = diffused light intensity ($0 <= I_d <= 255$)
K_a = ambient light coefficient ($0 < K_a < 1$)
K_d = diffused light coefficient ($0 < K_d < 1$)
N_u = unit vector normal to an illuminated patch
L_u = unit light vector (x, y, z)
$w(\theta)$ = a specular-reflection constant ($0 < w(\theta) < 1$)
n = a constant (>> 0 for a dull surface)
$\cos \alpha$ = the cosine of the angle between the sight vector S and the reflection vector R

The values I_a, K_a, I_d, K_d, $w(\theta)$, and n are specified in the configuration file; pleasing values are derived experimentally (i.e. through trial and error).

CHAPTER 6

TESTING AND RESULTS IN THREE DIMENSIONS

6.1 Canonical Images

In this chapter, we examine the results of three-dimensional segmentation using the MRI3D program. We begin by observing some canonical images; that is, images of certain structures in and of the head which are readily identifiable by the human eye as discrete structures, and which also lend themselves fairly well to segmentation. We then modify some of the settings for the manner in which region-growing, gradient-approximation, and rendering are performed, and we compare the results.

The following images are (in the judgment of the author) reasonably satisfactory, with the exception of those for the cerebellum (figure 6.1-3) and the cerebrum (figure 6.1-4). These two structures were found to be rather difficult to segment in two dimensions, and, despite exhaustive attempts, are apparently not separable in three (using the automatic methods described in this paper). It would appear that the consistency of the tissue from the two structures is sufficiently similar to defy automatic segmentation, and that some form of manual intervention would be necessary.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
=================================================================
SeedX:      120                       IlluminationMethod: Depth-cue
SeedY:      103                       PhongIa:    100.000000
SeedZ:      54                        PhongId:    100.000000
RGImplementation: Iterative           PhongKa:    0.800000
RGAlgorithm: 6-neighbor               PhongKd:    0.800000
LDL:        1300                      PhongLx:    -300.000000
UDL:        1700                      PhongLy:    300.000000
Threshold:  180                       PhongLz:    300.000000
ApproximationMethod: 6-neighbor       PhongWTheta: 0.800000
ZBufferView: Sagittal                 PhongN:     48.000000
Maxstacktop: 11159                    Voxels accumulated: 28775
Histogram settings: L1=30  U1=150  L2=20  U2=189
ZBufferFile: c:/thesis/mri3d/pix2/corpcall.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: corpus callosum
```

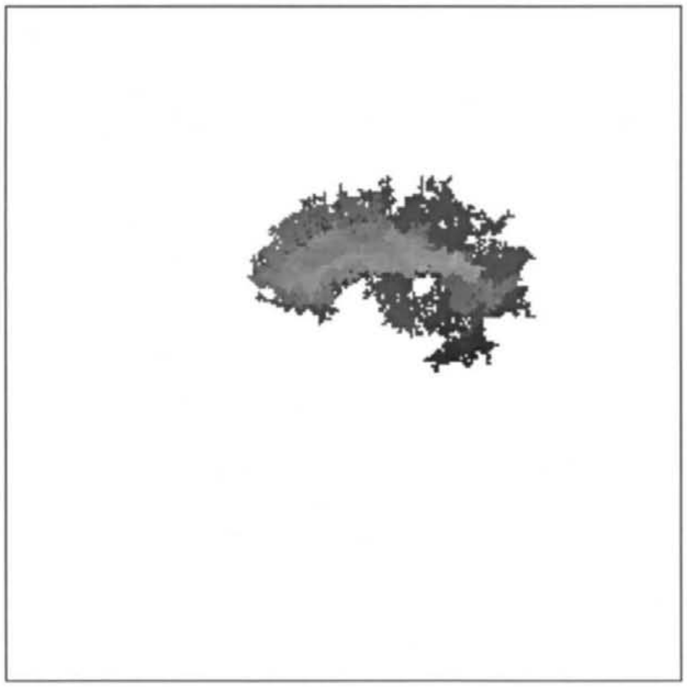


Figure 6.1-1: Corpus Callosum

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      155                        IlluminationMethod: Depth-cue
SeedY:      140                        PhongIa:     100.000000
SeedZ:      54                         PhongId:     100.000000
RGImplementation: Iterative            PhongKa:      0.800000
RGAlgorithm: 6-neighbor                PhongKd:      0.800000
LDL:        1100                       PhongLx:     -300.000000
UDL:        1500                       PhongLy:      300.000000
Threshold:  160                        PhongLz:      300.000000
ApproximationMethod: 6-neighbor        PhongWTheta: 0.800000
ZBufferView: Sagittal                  PhongN:      48.000000
Maxstacktop: 222                       Voxels accumulated: 768
Histogram settings: L1=80  U1=110  L2=50  U2=150
ZBufferFile: c:/thesis/mri3d/pix2/pons.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: pons
```

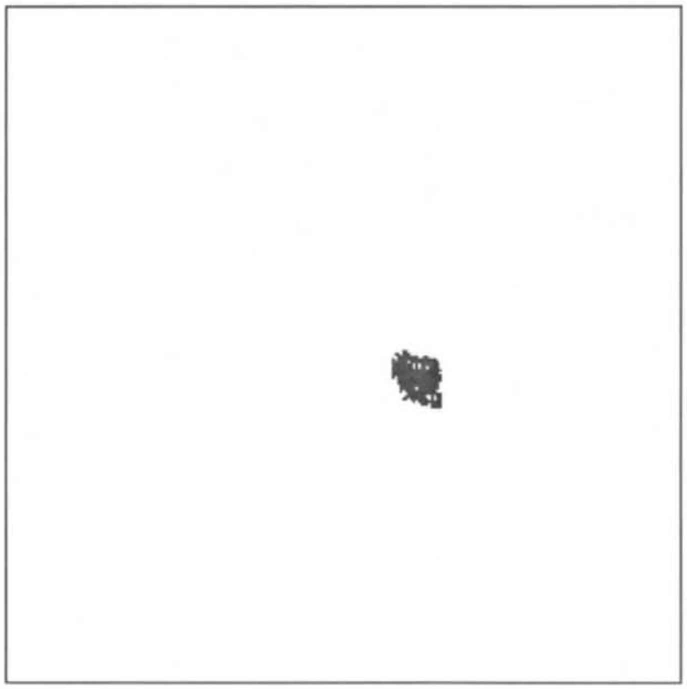


Figure 6.1-2: Pons

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      181                        IlluminationMethod: Depth-cue
SeedY:      147                        PhongIa:    100.000000
SeedZ:      54                         PhongId:    100.000000
RGImplementation: Iterative            PhongKa:     0.800000
RGAlgorithm: 6-neighbor                PhongKd:     0.800000
LDL:        1100                       PhongLx:    -300.000000
UDL:        1500                       PhongLy:     300.000000
Threshold:  250                        PhongLz:     300.000000
ApproximationMethod: 6-neighbor        PhongWTheta: 0.800000
ZBufferView: Sagittal                  PhongN:     48.000000
Maxstacktop: 32338                     Voxels accumulated: 106861
Histogram settings: L1=37  U1=170  L2=20  U2=189
ZBufferFile: c:/thesis/mri3d/pix2/crbllum.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: cerebellum
```

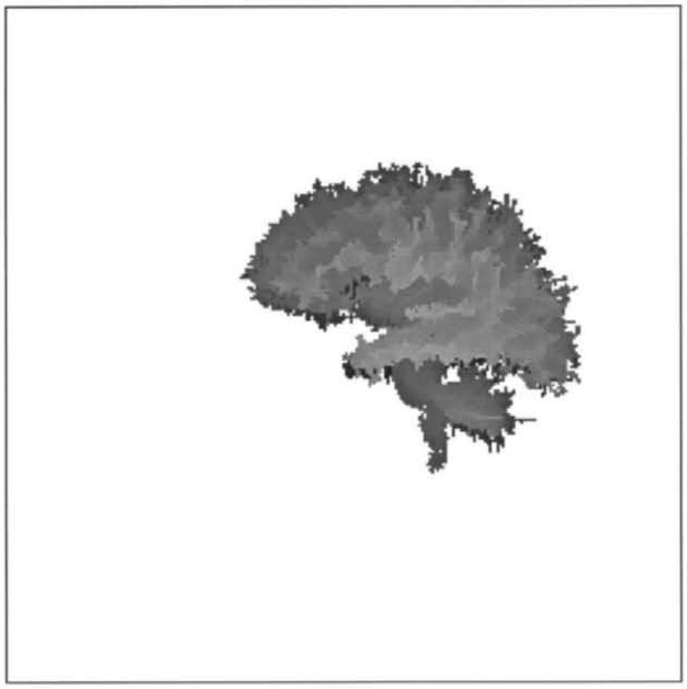


Figure 6.1-3: Cerebellum

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
================================================================
SeedX:      151                          IlluminationMethod: Depth-cue
SeedY:      75                           PhongIa:     100.000000
SeedZ:      54                           PhongId:     100.000000
RGImplementation: Iterative              PhongKa:     0.800000
RGAlgorithm: 6-neighbor                  PhongKd:     0.800000
LDL:        1000                         PhongLx:     -300.000000
UDL:        1800                         PhongLy:     300.000000
Threshold:  350                          PhongLz:     300.000000
ApproximationMethod: 6-neighbor          PhongWTheta: 0.800000
ZBufferView: Sagittal                    PhongN:      48.000000
Maxstacktop: 88191                       Voxels accumulated: 164043
Histogram settings: L1=37  U1=170  L2=37  U2=189
ZBufferFile: c:/thesis/mri3d/pix2/cerebrum.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: cerebrum
```

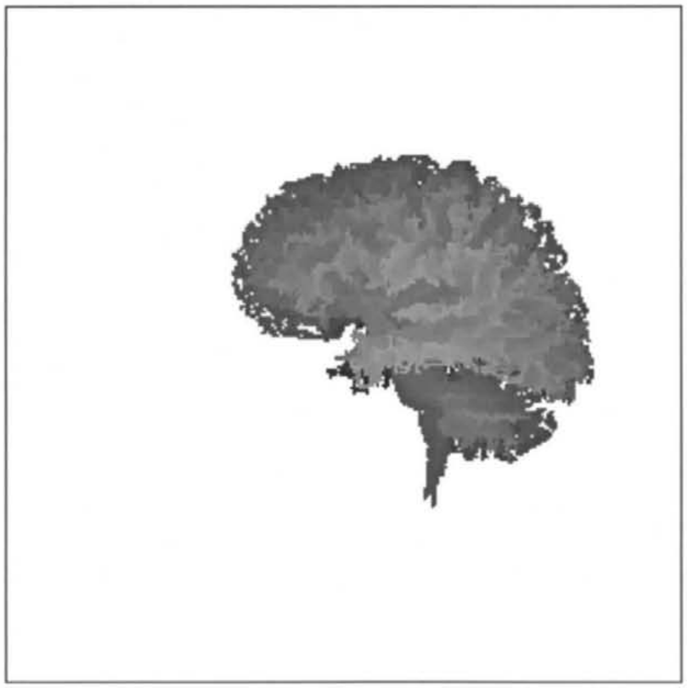


Figure 6.1-4: Cerebrum

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      151                          IlluminationMethod: Depth-cue
SeedY:      75                           PhongIa:     100.000000
SeedZ:      54                           PhongId:     100.000000
RGImplementation: Iterative              PhongKa:     0.800000
RGAlgorithm: 6-neighbor                  PhongKd:     0.800000
LDL:        900                          PhongLx:     -300.000000
UDL:        2000                         PhongLy:     300.000000
Threshold:  900                          PhongLz:     300.000000
ApproximationMethod: 6-neighbor          PhongWTheta: 0.800000
ZBufferView: Sagittal                    PhongN:      48.000000
Maxstacktop: 263879                      Voxels accumulated: 408407
Histogram settings: L1=60  U1=175  L2=25  U2=189
ZBufferFile: c:/thesis/mri3d/pix2/brain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain
```

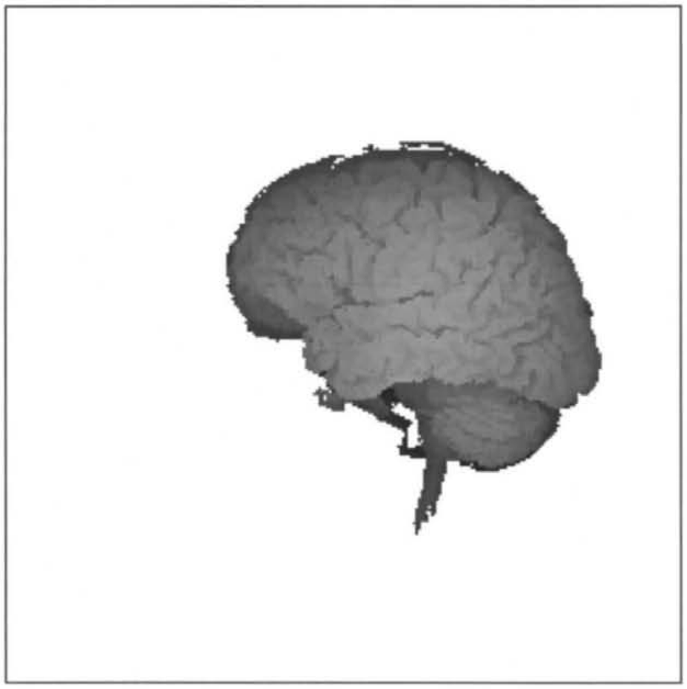


Figure 6.1-5: Brain

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
=============================================================
SeedX:      199                         IlluminationMethod: Depth-cue
SeedY:      55                          PhongIa:    100.000000
SeedZ:      54                          PhongId:    100.000000
RGImplementation: Iterative             PhongKa:     0.800000
RGAlgorithm: 6-neighbor                 PhongKd:     0.800000
LDL:        400                         PhongLx:    -300.000000
UDL:        2800                        PhongLy:     300.000000
Threshold:  8000                        PhongLz:     300.000000
ApproximationMethod: 6-neighbor         PhongWTheta: 0.800000
ZBufferView: Sagittal                   PhongN:     48.000000
Maxstacktop: 1092847                    Voxels accumulated: 1555035
Histogram settings: L1=75  U1=189  L2=15  U2=180
ZBufferFile: c:/thesis/mri3d/pix2/scalp.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: scalp
```

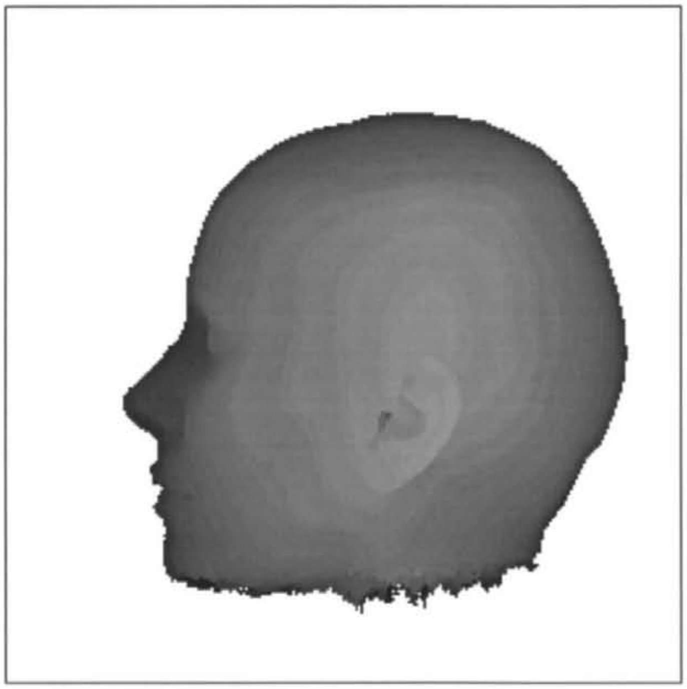


Figure 6.1-6: Scalp

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==================================================================
SeedX:      88                            IlluminationMethod: Phong
SeedY:      124                           PhongIa:     100.000000
SeedZ:      37                            PhongId:     100.000000
RGImplementation: Iterative               PhongKa:      0.800000
RGAlgorithm: 6-neighbor                   PhongKd:      0.800000
LDL:        50                            PhongLx:    -300.000000
UDL:        1500                          PhongLy:     300.000000
Threshold:  500                           PhongLz:     300.000000
ApproximationMethod: 6-neighbor           PhongWTheta: 0.800000
ZBufferView: Sagittal                     PhongN:      48.000000
Maxstacktop: 1314                         Voxels accumulated: 1754

ZBufferFile: c:/thesis/mri3d/pix2/eye1.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: left eye, sagittal, phong, 6n
```

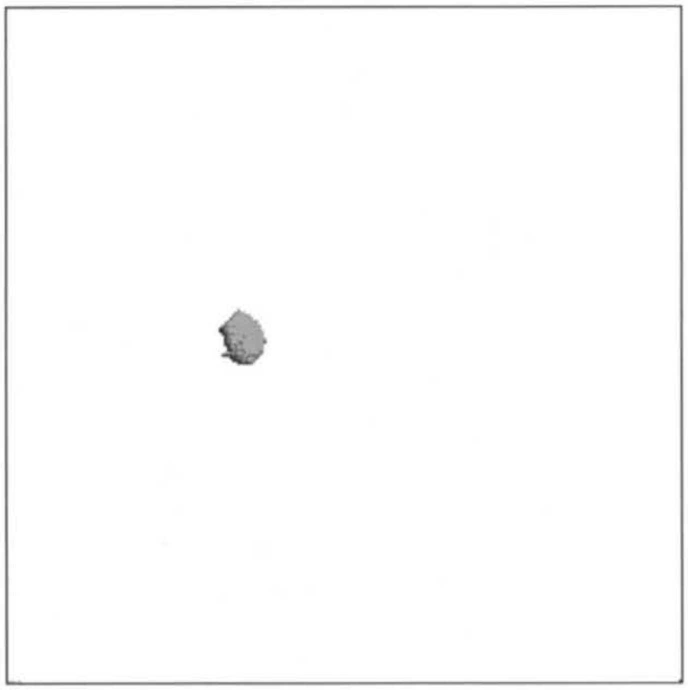


Figure 6.1-7: Eye

6.2 The Density Range and Density Gradient Threshold

As was observed during two-dimensional experimentation, the care with which the upper and lower limits of the density range and the density gradient threshold are chosen, can make the difference between a most informative picture and one in which no meaningful segmentation takes place. This is no less true in three dimensions; in fact, because of the extra dimension involved, and therefore the additional opportunities for connectedness to occur, some structures may not be segmentable at all without some form of manual intervention.

6.3 The Gradient Approximation Method

It can be said in general that the differences apparent in the quality of the images as a result of changes in gradient-approximation method are relatively minor. The six-neighbor method seems to offer the best appearance of all three, although it takes about twice as much time to complete, and requires three to four times the stack space of the other methods.

6.3.1 Six-neighbor Gradient Approximation

All of the canonical images were generated using the six-neighbor method of gradient approximation. We compare the Phong-shaded version of the canonical image of the brain (figure 6.3.1-1) to the images of the brain created using other gradient-approximation methods. Note that the appearance of the surface of the brain is rather smooth. The six-neighbor image was completed in 41.28 minutes.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      151                      IlluminationMethod: Phong
SeedY:      75                       PhongIa:     100.000000
SeedZ:      54                       PhongId:     100.000000
RGImplementation: Iterative          PhongKa:     0.800000
RGAlgorithm: 6-neighbor              PhongKd:     0.800000
LDL:        900                      PhongLx:     -300.000000
UDL:        2000                     PhongLy:     300.000000
Threshold:  900                      PhongLz:     300.000000
ApproximationMethod: 6-neighbor      PhongWTheta: 0.800000
ZBufferView: Sagittal                PhongN:      48.000000
Maxstacktop: 263879                  Voxels accumulated: 408407

ZBufferFile: c:/thesis/mri3d/pix2/phgbrain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain, phong-shaded
```

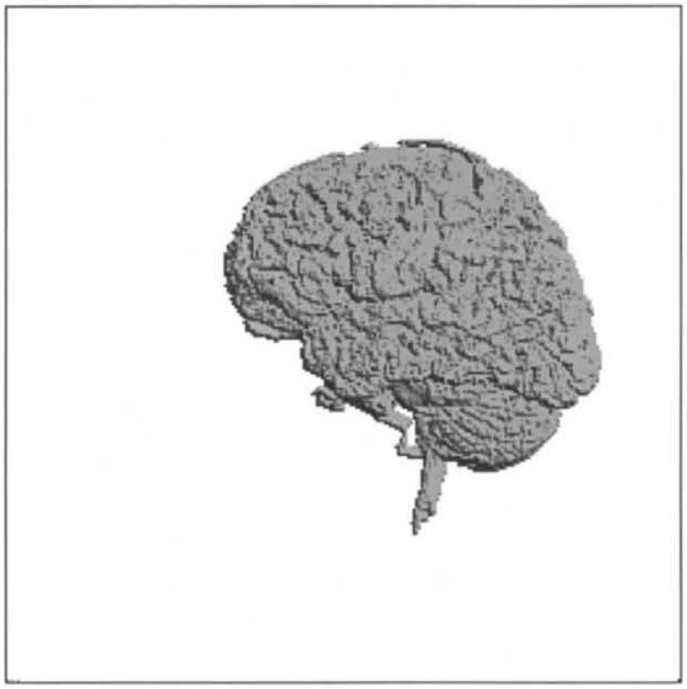


Figure 6.3.1-1: Brain, Six-neighbor Gradient Approximation

6.3.2 Frei-Chen Gradient Approximation

The image created using Frei-Chen gradient approximation (figure 6.3.2-1) displays a more eroded appearance than that of the six-neighbor method. It is also to be noted that the Frei-Chen image took about half the time to generate (20.76 minutes) as the six-neighbor image; and that somewhat less than half the number of voxels were accumulated to the Frei-Chen final data set as were accumulated to the six-neighbor set.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      151                           IlluminationMethod: Phong
SeedY:      75                            PhongIa:     100.000000
SeedZ:      54                            PhongId:     100.000000
RGImplementation: Iterative               PhongKa:     0.800000
RGAlgorithm: 6-neighbor                   PhongKd:     0.800000
LDL:        900                           PhongLx:     -300.000000
UDL:        2000                          PhongLy:     300.000000
Threshold:  1400                          PhongLz:     300.000000
ApproximationMethod: Frei-Chen            PhongWTheta: 0.800000
ZBufferView: Sagittal                     PhongN:      48.000000
Maxstacktop: 82751                        Voxels accumulated: 155270

ZBufferFile: c:/thesis/mri3d/pix2/fcbrain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain, phong-shaded, frei-chen
```

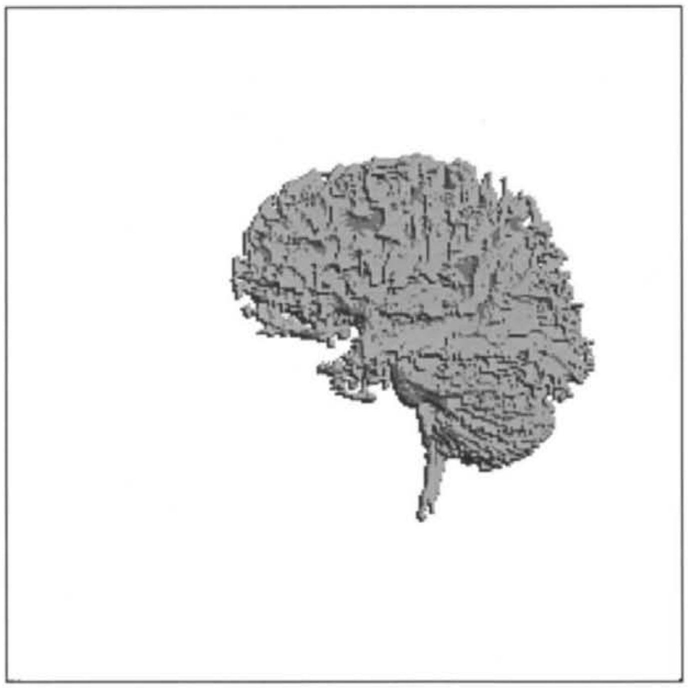


Figure 6.3.2-1: Brain, Frei-Chen Gradient Approximation

6.3.3 Pseudo-Sobel Gradient Approximation

The image created using Pseudo-Sobel approximation (figure 6.3.3-1) is even more eroded than the Frei-Chen image. The Pseudo-Sobel data set contains slightly fewer voxels than the Frei-Chen set, and took slightly less time to complete (18.65 minutes).

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
============================================================
SeedX:      151                          IlluminationMethod: Phong
SeedY:      75                           PhongIa:     100.000000
SeedZ:      54                           PhongId:     100.000000
RGImplementation: Iterative              PhongKa:      0.800000
RGAlgorithm: 6-neighbor                  PhongKd:      0.800000
LDL:        900                          PhongLx:    -300.000000
UDL:        2000                         PhongLy:     300.000000
Threshold:  3300                         PhongLz:     300.000000
ApproximationMethod: Pseudo-Sobel        PhongWTheta: 0.800000
ZBufferView: Sagittal                    PhongN:      48.000000
Maxstacktop: 70528                       Voxels accumulated: 138546

ZBufferFile: c:/thesis/mri3d/pix2/psbrain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain, phong-shaded, pseudo-sobel
```

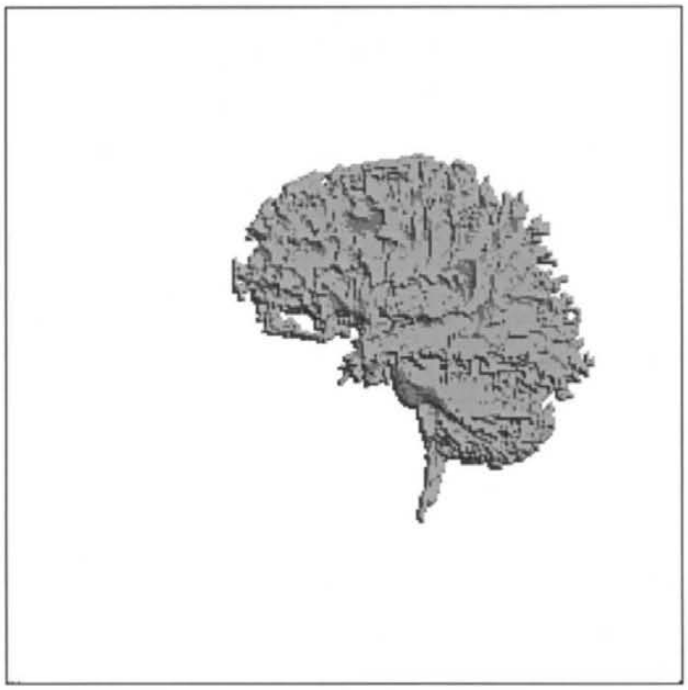


Figure 6.3.3-1: Brain, Pseudo-Sobel Gradient Approximation

6.4 The Region-growing Implementation

If time and stack space were limitless, then the results of different region-growing methods should be essentially the same; observe the results obtained in two dimensions with small regions, where, regardless of whether the algorithm chosen was recursive, simulated-recursive, or spanfilling, the results in terms of voxels accumulated were identical (table 4.2.5.4-1). (The RAW-format files were also shown to be equivalent.) In three dimensions, the degree to which recursion must take place is exponentially greater, since we are dealing with a cubic volume rather than a square area. Therefore, the requirements for stack space and runtime become significantly greater.

6.4.1 Recursion

Figure 6.4.1-1 shows the results of taking the traditional recursive approach to region-growing in three dimensions. The settings used are for the entire brain, as in the canonical image used in figure 6.1-5. Evidently the program ran out of stack space quite early on, terminating in 9.06 minutes, with only 127,105 voxels accumulated (compared to 408,407 voxels for the canonical image using its own stack in extended memory). Whereas the simulated-recursion method was able to grow its stack to 263,879 frames (with room, theoretically, to grow to over 3 million frames in 15 Mb of extended memory), the recursive algorithm, using the system's stack in conventional memory, ran out after only 1429 frames.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
===============================================================
SeedX:      151                       IlluminationMethod: Depth-cue
SeedY:      75                        PhongIa:     100.000000
SeedZ:      54                        PhongId:     100.000000
RGImplementation: Recursive           PhongKa:     0.800000
RGAlgorithm: 6-neighbor               PhongKd:     0.800000
LDL:        900                       PhongLx:     -300.000000
UDL:        2000                      PhongLy:     300.000000
Threshold:  900                       PhongLz:     300.000000
ApproximationMethod: 6-neighbor       PhongWTheta: 0.800000
ZBufferView: Sagittal                 PhongN:      48.000000
Maxstacktop: 1429                     Voxels accumulated: 127105

ZBufferFile: c:/thesis/mri3d/pix2/recbrain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain (true recursion)
```

Figure 6.4.1-1: Brain, Recursive Region-growing

6.4.2 Simulated Recursion

The canonical image of the brain (figure 6.1-5) uses simulated recursion as its region-growing method. It maintains its own stack in extended memory, which, given the data structure used for each "stack frame", permits it to "recurse" over 3 million times in 15 Mb of extended memory. The canonical brain image was generated in 41.23 minutes, accumulating 408,407 voxels and making 263,879 simulated recursive calls.

6.4.3 6-connected and 26-connected Region-growing

All of the three-dimensional images generated previously, recursive or otherwise, used the six-neighbor region-growing algorithm. That is to say, for each voxel under examination for membership in the segmented region, its neighbors to the four compass directions and to the front and back were examined with recursive calls. In the following figure 6.4.3-1, we observe the results of examining not only each voxel's neighbors to the four compass directions and to the front and rear, but also at the corners and along the edges of the cube in which it resides.

It is clear from the picture that segmentation has not been well-performed; the brain is there, but so is a quantity of other material from the face, jaw, and neck. Interestingly, the amount of time required to run the 26-neighbor model, 156.38 minutes, is out of proportion to the number of voxels accumulated (552,501 voxels) and to the number of stack frames used (394,778 frames); compared to the canonical brain's statistics, the 26-

neighbor model took four times as long and twice as much stack space to accumulate 25 percent more voxels. This would suggest that, even if the segmentation had been successful, it still would have been significantly less efficient than the six-neighbor algorithm.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
===============================================================
SeedX:      151                        IlluminationMethod: Depth-cue
SeedY:      75                         PhongIa:    100.000000
SeedZ:      54                         PhongId:    100.000000
RGImplementation: Iterative            PhongKa:    0.800000
RGAlgorithm: 26-neighbor               PhongKd:    0.800000
LDL:        900                        PhongLx:    -300.000000
UDL:        2000                       PhongLy:    300.000000
Threshold:  900                        PhongLz:    300.000000
ApproximationMethod: 6-neighbor        PhongWTheta: 0.800000
ZBufferView: Sagittal                  PhongN:     48.000000
Maxstacktop: 394778                    Voxels accumulated: 552501

ZBufferFile: c:/thesis/mri3d/pix2/26brain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain, 26-connected
```

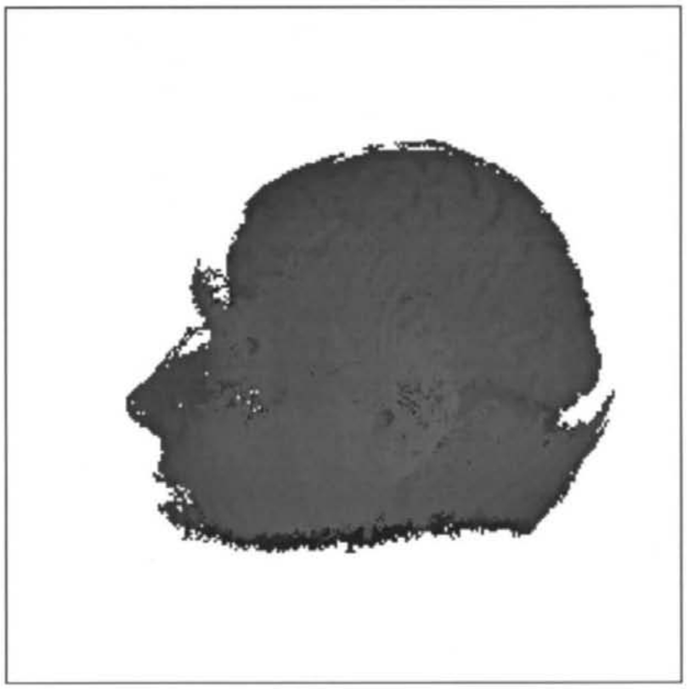


Figure 6.4.3-1: Brain, 26-connected Region-growing

6.4.4 Recursive Spanfilling

The proposed stacked-spanfill algorithm as conceived provides a significant result but also possesses a serious drawback. When used to display the entire head (essentially segmenting the head from the surrounding volume), it is nine times faster than the six-neighbor (spanfilling: 15.48 minutes, and six-neighbor: 138.3 minutes), nearly equally effective at accumulating voxels (spanfilling: 1.51 million voxels, and six-neighbor: 1.55 million voxels), much less hungry for stack frames (spanfilling: 847 frames, and six-neighbor: over 1 million frames); and provides a very pleasing picture. (Figure 6.4.4-1 shows the result of the stacked-spanfill algorithm; figure 6.4.4-2, the six-neighbor equivalent.)

On the other hand, the algorithm considers only connectivity between contiguous voxels in the same slice; therefore, if there exists some sub-structure known to belong to a larger super-structure we wish to image, but which is not intersected by the seedslice, then that sub-structure will not be captured by the algorithm. For example: a transverse seedslice of 3DHEAD (our input VDS), taken at transverse slice 55, intersects the head rather high up, completely missing the ears. If we decide to segment the scalp (taken throughout this paper to mean all of the skin which covers the head, down to the neck) and choose a seedpoint accordingly, we will obtain only that part of the scalp which is intersected by the transverse slice (and, of course, voxels which lie in the same sagittal slices). Since the ears were not intersected by the seedslice <u>and</u> they do not lie in the same sagittal slices, they are completely excluded from the segmentation.

Figures 6.4.4-3 and 6.4.4-4 display the result. Nevertheless, the image of the brain reposing inside the skull *is* rather striking.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
================================================================
SeedX:      127                       IlluminationMethod: Depth-cue
SeedY:      127                       PhongIa:     100.000000
SeedZ:      54                        PhongId:     100.000000
RGImplementation: Iterative           PhongKa:      0.800000
RGAlgorithm: stacked-spanfill         PhongKd:      0.800000
LDL:        400                       PhongLx:     -300.000000
UDL:        2800                      PhongLy:     300.000000
Threshold:  8000                      PhongLz:     300.000000
ApproximationMethod: 6-neighbor       PhongWTheta: 0.800000
ZBufferView: Sagittal                 PhongN:      48.000000
Maxstacktop: 847                      Voxels accumulated: 1515956
Histogram settings: L1=60  U1=189  L2=25  U2=180
ZBufferFile: c:/thesis/mri3d/pix2/spanfil1.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: entire head, stacked-spanfill
```

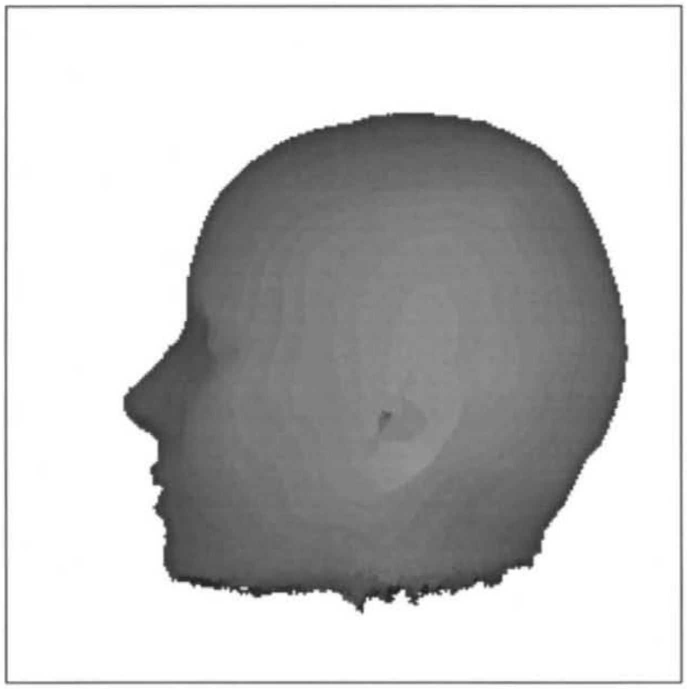


Figure 6.4.4-1: Head, Stacked Spanfill Region-growing

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      127                        IlluminationMethod: Depth-cue
SeedY:      127                        PhongIa:    100.000000
SeedZ:      54                         PhongId:    100.000000
RGImplementation: Iterative            PhongKa:    0.800000
RGAlgorithm: 6-neighbor                PhongKd:    0.800000
LDL:        400                        PhongLx:    -300.000000
UDL:        2800                       PhongLy:    300.000000
Threshold:  8000                       PhongLz:    300.000000
ApproximationMethod: 6-neighbor        PhongWTheta: 0.800000
ZBufferView: Sagittal                  PhongN:     48.000000
Maxstacktop: 1082916                   Voxels accumulated: 1554910
Histogram settings: L1=75  U1=189  L2=25  U2=189
ZBufferFile: c:/thesis/mri3d/pix2/allhead.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: entire head
```

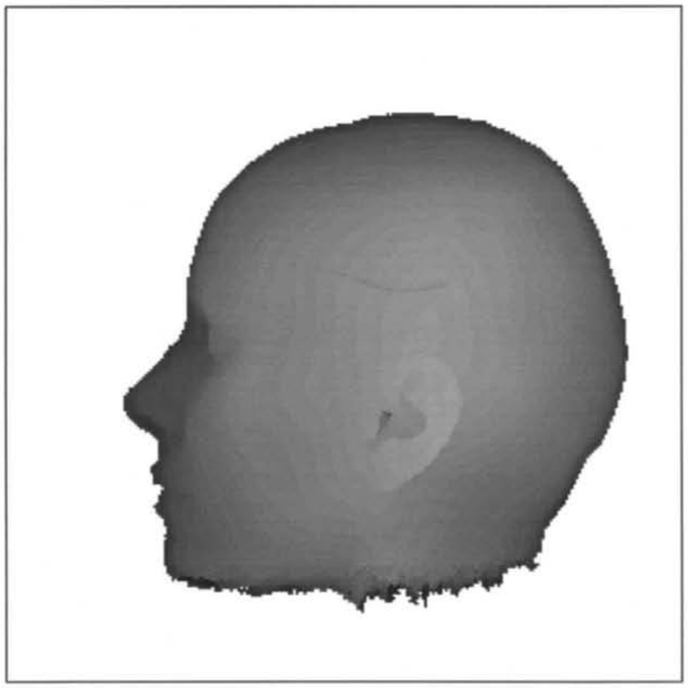


Figure 6.4.4-2: Head, Six-neighbor Region-growing

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
================================================================
SeedX:     199                        IlluminationMethod: Depth-cue
SeedY:     55                         PhongIa:    100.000000
SeedZ:     54                         PhongId:    100.000000
RGImplementation: Iterative           PhongKa:    0.800000
RGAlgorithm: stacked-spanfill         PhongKd:    0.800000
LDL:       400                        PhongLx:    -300.000000
UDL:       2800                       PhongLy:    300.000000
Threshold:  8000                      PhongLz:    300.000000
ApproximationMethod: 6-neighbor       PhongWTheta: 0.800000
ZBufferView: Sagittal                 PhongN:     48.000000
Maxstacktop: 486                      Voxels accumulated: 1123371
Histogram settings: L1=90  U1=140  L2=25  U2=150
ZBufferFile: c:/thesis/mri3d/pix2/spanfil2.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: scalp, stacked-spanfill
```

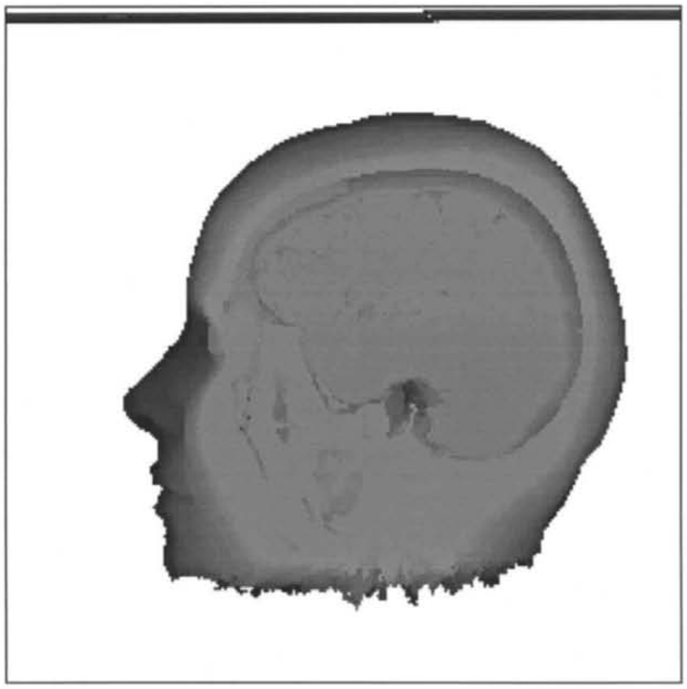


Figure 6.4.4-3: Sagittal Scalp, Stacked Spanfill Region-growing

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
=================================================================
SeedX:      199                          IlluminationMethod: Depth-cue
SeedY:      55                           PhongIa:    100.000000
SeedZ:      54                           PhongId:    100.000000
RGImplementation: Iterative              PhongKa:     0.800000
RGAlgorithm: stacked-spanfill            PhongKd:     0.800000
LDL:        400                          PhongLx:    -300.000000
UDL:        2800                         PhongLy:     300.000000
Threshold:  8000                         PhongLz:     300.000000
ApproximationMethod: 6-neighbor          PhongWTheta: 0.800000
ZBufferView: Coronal                     PhongN:     48.000000
Maxstacktop: 486                         Voxels accumulated: 1123371
Histogram settings: L1=75  U1=160  L2=25  U2=180
ZBufferFile: c:/thesis/mri3d/pix2/spanfil3.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: coronal scalp, stacked-spanfill
```

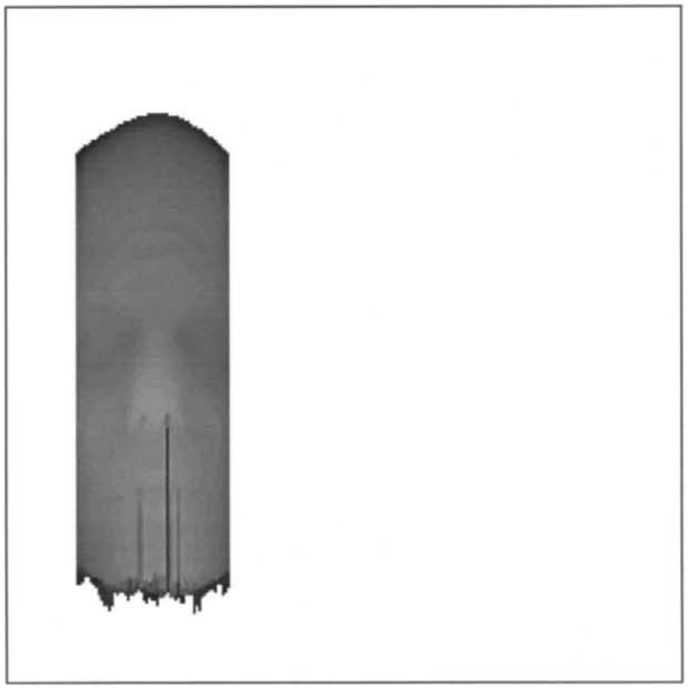


Figure 6.4.4-4: Coronal Scalp, Stacked Spanfill Region-growing

6.5 Image Rendering and Viewpoints

As stated, MRI3D has the capability of rendering images using ray-tracing, and of illuminating them according to the Phong formula. It can also display the rendered image in any of the three axial orientations. Many sagittal views of the brain and head have already been displayed; the remaining images in this chapter show the coronal and transverse views of these structures. Observe that the voxels accumulated and stack frames used are identical to those of their sagittal counterparts; they are, after all, simply different views of the same segmented region.

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      151                          IlluminationMethod: Phong
SeedY:      75                           PhongIa:    100.000000
SeedZ:      54                           PhongId:    100.000000
RGImplementation: Iterative              PhongKa:     0.800000
RGAlgorithm: 6-neighbor                  PhongKd:     0.800000
LDL:        900                          PhongLx:    -300.000000
UDL:        2000                         PhongLy:     300.000000
Threshold:  900                          PhongLz:     300.000000
ApproximationMethod: 6-neighbor          PhongWTheta: 0.800000
ZBufferView: Coronal                     PhongN:      48.000000
Maxstacktop: 263879                      Voxels accumulated: 408407

ZBufferFile: c:/thesis/mri3d/pix2/corbrain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain, phong-shaded, coronal
```

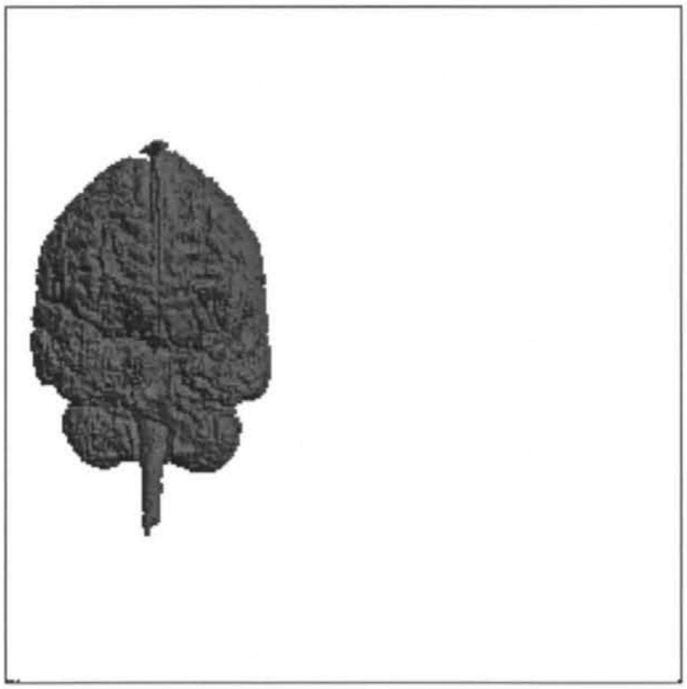


Figure 6.5-1: Brain, Coronal View

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:     199                       IlluminationMethod: Phong
SeedY:     55                        PhongIa:    100.000000
SeedZ:     54                        PhongId:    100.000000
RGImplementation: Iterative          PhongKa:     0.800000
RGAlgorithm: 6-neighbor              PhongKd:     0.800000
LDL:       400                       PhongLx:    -300.000000
UDL:       2800                      PhongLy:    300.000000
Threshold: 8000                      PhongLz:    300.000000
ApproximationMethod: 6-neighbor      PhongWTheta: 0.800000
ZBufferView: Coronal                 PhongN:     48.000000
Maxstacktop: 1092847                 Voxels accumulated: 1555035

ZBufferFile: c:/thesis/mri3d/pix2/corhead.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile: d:/temp.vds
Comments: scalp, coronal view
```

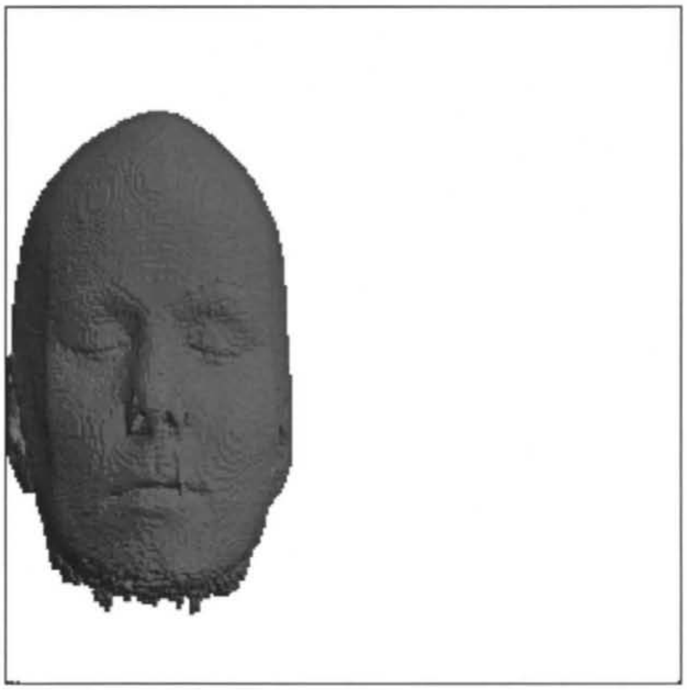


Figure 6.5-2: Head, Coronal View

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      151                          IlluminationMethod: Phong
SeedY:      75                           PhongIa:     100.000000
SeedZ:      54                           PhongId:     100.000000
RGImplementation: Iterative              PhongKa:     0.800000
RGAlgorithm: 6-neighbor                  PhongKd:     0.800000
LDL:        900                          PhongLx:     -300.000000
UDL:        2000                         PhongLy:     300.000000
Threshold:  900                          PhongLz:     300.000000
ApproximationMethod: 6-neighbor          PhongWTheta: 0.800000
ZBufferView: Transverse                  PhongN:      48.000000
Maxstacktop: 263879                      Voxels accumulated: 408407

ZBufferFile: c:/thesis/mri3d/pix2/xvsbrain.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: brain, phong-shaded, transverse
```

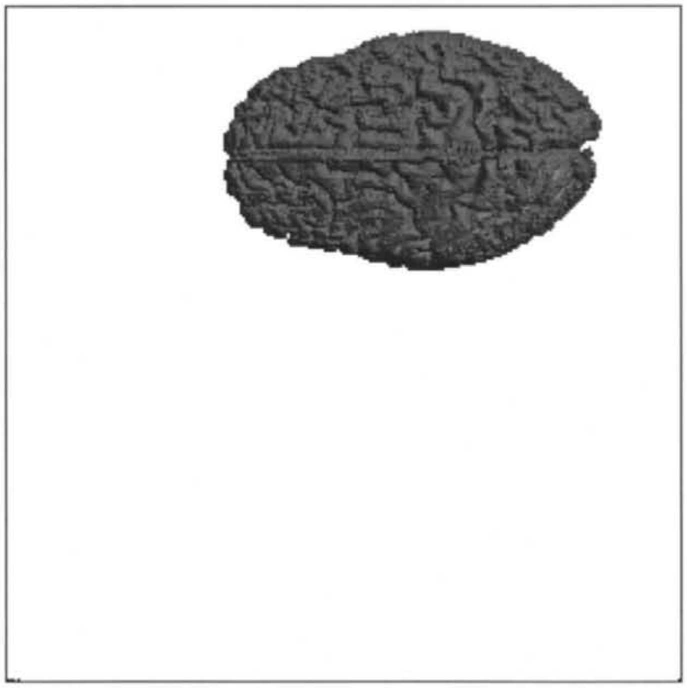


Figure 6.5-3: Brain, Transverse View

```
MRI3D - Bill Bell - Master's Thesis Project - Summer/Fall 1996
==============================================================
SeedX:      199                       IlluminationMethod: Phong
SeedY:      55                        PhongIa:     100.000000
SeedZ:      54                        PhongId:     100.000000
RGImplementation: Iterative           PhongKa:     0.800000
RGAlgorithm: 6-neighbor               PhongKd:     0.800000
LDL:        400                       PhongLx:     -300.000000
UDL:        2800                      PhongLy:     300.000000
Threshold:  8000                      PhongLz:     300.000000
ApproximationMethod: 6-neighbor       PhongWTheta: 0.800000
ZBufferView: Transverse               PhongN:      48.000000
Maxstacktop: 1092847                  Voxels accumulated: 1555035

ZBufferFile: c:/thesis/mri3d/pix2/xvshead.[RAW,PS,EPS]
VDSInfile:  d:/3dhead.vds
VDSOutfile:  d:/temp.vds
Comments: scalp, transverse view
```

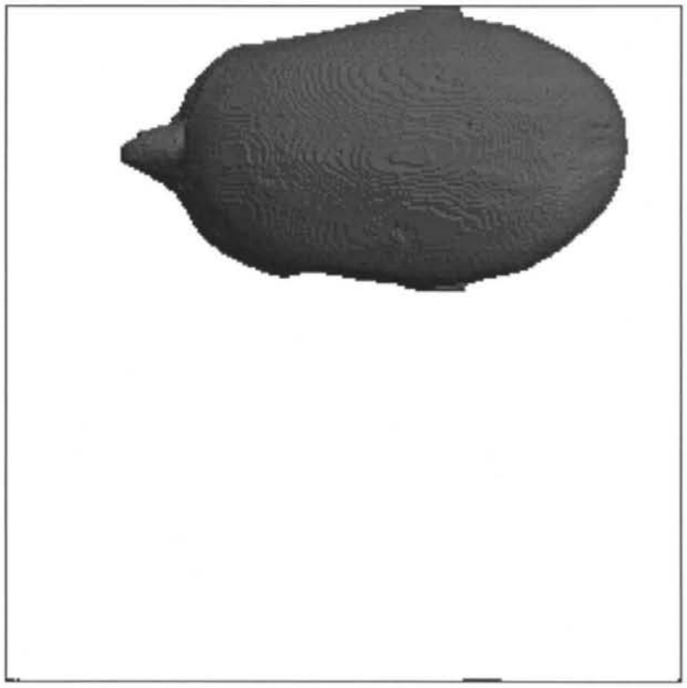


Figure 6.5-4: Head, Transverse View

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

As automated segmentation projects go, the GETCOORD/MRI3D pair acquits itself fairly well. However, there is certainly room for improvement and enhancements. Herewith, then, are some reflections on the strengths and shortcomings of this project, and suggestions for future research.

7.1 Windows User Interface

As it stands, the means of entering configuration data into GETCOORD and MRI3D are rather primitive. The former relies for its input upon a question-and-answer paradigm; the latter, upon a text file of configuration data. The Microsoft Windows GUI provides the developer with ample opportunities to create a user-interface that is easy and intuitive to use. The presence of multiple data entry fields, scroll bars, check boxes, option buttons, and common dialog boxes would provide a familiar environment to the experienced Windows user.

7.2 Changes in the Nature of the Convolution Kernels

Some convolution-based image-processing methods involve kernels larger than 3 x 3. It might be interesting to observe the results of gradient-approximations employing, for example, a 5 x 5 or a 7 x 7 matrix.

The gradient of a volume, as applied here, is the rate of change in density over the volume. A rate of change may be regarded as the first derivative of the underlying data. Just as the convolution kernel used in MRI3D computed the gradient of a volume, a future version might look at the second derivative of the data in a volume (the first derivative of the gradient) as a means of better differentiating adjacent tissue types.

7.3 Image-processing Enhancements

Just as the histogram of grayshades of a two-dimensional image might be stretched in order to improve contrast, one might consider computing a histogram of gradients over a volumetric region, and stretching it to improve tissue differentiation.

The MIDTERM image-processing program [Bell94A] provided many different image-processing techniques for the manipulation of 2-D data: histogram-stretching, adaptive histogram equalization, median filtering, customized convolution kernels, and the like. An extension of these same ideas into three dimensions would no doubt yield fascinating results.

MRI3D was able to segment structures from a single seedpoint. However, some structures in the body come in sets of two or more, for example, eyes, kidneys, lungs, teeth, and bones. The ability to combine the segmented image of multiple structures would be a valuable addition to the project.

7.4 Manual Modification of Segmentation

For all their strengths, GETCOORD and MRI3D demonstrated that automatic segmentation algorithms often leave much to be desired. It is a time-consuming process to narrow down the exact set of density and gradient-threshold limits which will result in a pleasing image. We have often wished we could simply point to an area on the screen and tell the computer, "Focus only on this area; ignore all other areas, even if your criteria for connectedness are met." It would be useful to be able to delineate to the computer (using a mouse or digitizer) those areas of interest. (A three-dimensional implementation of a solution will no doubt prove particularly challenging!)

7.5 Enhancements to Stacked Spanfilling

As observed, the stacked spanfill method of region-growing can, under a few circumstances, yield a pleasing picture in much less time than the recursive methods. However, it often leaves out important structures which were not intersected by the single seedslice. Future research into the efficacy of this method might center upon ways of using multiple seedslices without giving up the significant improvements in runtimes which were realized.

REFERENCES

[Abraham88]
Abraham, R. J. *et al*, Introduction to NMR Spectroscopy, John Wiley and Sons, 1988.

[Baxes94]
Baxes, Gregory A., Digital Image Processing: Principles and Applications, John Wiley and Sons, New York, NY, 1994.

[Bell94A]
Bell, William L., Jr., "MIDTERM Image Processing Methods Program," Image Processing class project, College of Computing Sciences and Engineering, University of North Florida, Jacksonville, FL, 1994.

[Bell94B]
Bell, William L., Jr., "3-Dimensional MRI Segmentation and Rendering," directed independent study project, College of Computing Sciences and Engineering, University of North Florida, Jacksonville, FL, 1994.

[Carlbom92]
Carlbom, Ingrid, William M. Hsu, Gudrun Klinker, Richard Szeliski *et al*, "Modeling and Analysis of Empirical Data in Collaborative Environments," Communications of the ACM 35, 6 (June 1992), pp. 74-84.

[Chellappa85A]
Chellappa, Rama and A. A. Sawchuck, Digital Image Processing and Analysis: Volume 1: Digital Image Processing, IEEE Computer Society Press, 1985.

[Chellappa85B]
Chellappa, Rama and A. A. Sawchuck, Digital Image Processing and Analysis: Volume 2: Digital Image Analysis, IEEE Computer Society Press, 1985.

[Chen88]
Chen, Chin-Tu, Jin-Shin Chou, Wei-Chung Lin, and Charles A. Pelizzari, "Edge and Surface Searching in Medical Images," Proceedings of SPIE - vol. 914, part A: Medical Imaging II, Roger Schneider, ed., Society of Photo-optical Instrumentation Engineers, 1988, pp. 594-599.

[Dalton88]
Dalton, B. L. and G. duBoulay, "Medical Image Matching," Proceedings of SPIE - vol. 914, part A: Medical Imaging II, Roger Schneider, ed., Society of Photo-optical Instrumentation Engineers, 1988.

[Foley90]
Foley, James D., Andries van Dam, Steven K. Feiner, and John F. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley, Reading, MA, 1990.

[Frohse61]
Frohse, Franz, Max Brödel, and Leon Schlossberg, Atlas of Human Anatomy, Barnes and Noble, New York, NY, 1961.

[Gadian95]
Gadian, David G., NMR and Its Applications to Living Systems, Oxford Science Publications, Oxford, U.K., 1995.

[Giger96]
Giger, Maryellen L. and Charles A. Pelizzari, "Advances in Tumor Imaging," Scientific American 275, 3 (September 1996) pp. 110-112.

[Goldman84]
Goldman, Mark R., M.D., and Gerald M. Pohost, M.D., "Nuclear Magnetic Resonance Imaging," Pediatric Cardiac Imaging, William F. Friedman, M.D., and Charles B. Higgins, M.D., eds., W. B. Saunders Company, Philadelphia, PA, 1984.

[Gonzalez92]
Gonzalez, Rafael C. and Richard E. Woods, Digital Image Processing, Addison-Wesley, Reading, MA, 1992.

[Gumas93]
Gumas, Spyro *et al*, Tricks of the Graphics Gurus, Sams Publishing, 1993.

[Hauser91]
Hauser, Peter, M.D., "Magnetic Resonance Imaging in Primary Affective Disorder," Brain Imaging in Affective Disorders, Peter Hauser, ed., American Psychiatric Press, 1991.

[Herman88]
Herman, Gabor T., "From 2-D to 3-D Representation," Mathematics and Computer Science in Medical Imaging, M. A. Viergever, ed., Springer-Verlag, 1988, pp. 197-220.

[HHS86]
U.S. Department of Health and Human Services National Institute of Health, "Magnetic Resonance Imaging", publication #87-1135, November 1986.

[Higgins84]
Higgins, Charles B., M.D., Elias H. Botvinick, M.D., Peter Lanzer, M.D., Robert Herfkens, M.D., *et al*, "Cardiovascular Imaging with Nuclear Magnetic Resonance," Pediatric Cardiac Imaging, William F. Friedman, M.D., and Charles B. Higgins, M.D., eds., W. B. Saunders Company, Philadelphia, PA, 1984.

[Höhne88]
Höhne, Karl-Heinz, Michael Bomans, Ulf Tiede, and Martin Reimer, "Display of Multiple 3-D Objects Using the Generalized Voxel-Model," Proceedings of SPIE - vol. 914, part B: Medical Imaging II, Roger Schneider, ed., Society of Photo-optical Instrumentation Engineers, 1988, pp. 850-854.

[Holzgang92]
Holzgang, David A., Understanding PostScript, Sybex, San Francisco, CA, 1992.

[Kapouleas88]
Kapouleas, Ioannis and Casimir A. Kulikowski, "A Model-Based System for the Interpretation of MR Human Brain Scans," Proceedings of SPIE - vol. 914, part A: Medical Imaging II, Roger Schneider, ed., Society of Photo-optical Instrumentation Engineers, 1988, pp. 429-437.

[Kippenhan94]
Kippenhan, Shane and Matt Schneble, "Design of Enhancements to MEDx Surface-Shade and Ray-Trace Modules", Sensor Systems, Inc., May 24, 1994 (via FTP to alw.nih.gov:/pub/MRIPS).

[Kleppner92]
Kleppner, Daniel, "MRI for the Third World," Physics Today, 45, 3 (March 1992), p. 9.

[Koenig88]
Koenig, H. A. and G. Laub, "Tissue Discrimination in Magnetic Resonance 3D Datasets," Proceedings of SPIE - vol. 914, part A: Medical Imaging II, Roger Schneider, ed., Society of Photo-optical Instrumentation Engineers, 1988, pp. 669-672.

[Korsh88]
Korsh, James F. and Leonard J. Garrett, Data Structures, Algorithms, and Program Style Using C, PWS-Kent, Boston, MA, 1988.

[Levoy88]
Levoy, Marc, "Direct Visualization of Surfaces from Computed Tomography Data," Proceedings of SPIE - vol. 914, part B: Medical Imaging II, Roger Schneider, ed., Society of Photo-optical Instrumentation Engineers, 1988.

[Lindley91]
Lindley, Craig A., Practical Image Processing in C, Wiley and Sons, 1991.

[Lorenson87]
Lorenson, W. E. and Harvey E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," Computer Graphics 21, 4 (July 1987), pp. 163-169.

[Mahoney96]
Mahoney, Diana Phillips, "The Art and Science of Medical Visualization," Computer Graphics World 19, 7 (July 1996), pp. 25-32.

[Patel96]
Patel, Vikas V., Michael W. Vannier, Jefferey L. Marsh, and Lun-Jou Lo, "Assessing Craniofacial Surgical Simulation," IEEE Computer Graphics and Applications 16, 1 (January 1996).

[Pavlidis82]
Pavlidis, Theo, Algorithms for Graphics and Image Processing, Computer Science Press, Rockville, MD, 1982.

[Qu96]
Qu, Xiaoqing and Xiaobo Li, "A 3D Surface Tracking Algorithm," Computer Vision and Image Understanding 64, 1 (July 1996).

[Raichle94]
Raichle, Marcus E., "Visualizing the Mind," Scientific American 270, 4 (April 1994), pp. 58-64.

[Ranjan94]
Ranjan, Vishwa and Alain Fournier, "Volume Models for Volumetric Data," IEEE Computer, July 1994.

[Rosenfeld76]
Rosenfeld, Azriel and Avinash C. Kak, Digital Picture Processing, Academic Press, New York, NY, 1976.

[Russ95]
Russ, John C., The Image Processing Handbook, CRC Press, Boca Raton, FL, 1995.

[Samtaney94]
Samtaney, Ravi, Deborah Silver, Norman Zabusky, and Jim Cao, "Visualizing Features and Tracking their Evolution," IEEE Computer 27, 7 (July 1994), pp. 20-27.

[Schalkoff89]
Schalkoff, Robert J., Digital Image Processing and Computer Vision, John Wiley and Sons, New York City, NY, 1989.

[Schneider95A]
Schneider, David, "MRI Goes Back to the Future," Scientific American 272, 3 (March 1995), p. 42.

[Schneider95B]
Schneider, David, "Changing the Image," Scientific American 272, 4 (April 1995), p. 42.

[Schwarzschild95]
Schwarzschild, Bertram, "Inhaling Hyperpolarized Noble Gas Helps Magnetic Resonance Imaging of Lungs," Physics Today, 48, 6 (June 1995), p. 17.

[Sims96]
Sims, David, "Putting the Visible Human to Work," IEEE Computer Graphics and Applications 16, 1 (January 1996).

[Tenenbaum90]
Tenenbaum, Aaron M., Yedidyah Langsam, and Moshe J. Augenstein, Data Structures Using C, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[Tiede96]
Tiede, Ulf, Thomas Schiemann, and Karl Heinz Höhne, "Visualizing the Visible Human," IEEE Computer Graphics and Applications 16, 1 (January 1996).

[Udupa82]
Udupa, Jayaram K., Sargur N. Srihari, and Gabor T. Herman, "Boundary Detection in Multidimensions," IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-4, 1 (January 1982), pp. 41-50.

[Wang96]
Wang, Stephen T. C., Robert C. Knowlton, Randy A. Hawkins, and Kenneth D. Laxer, "Multimodal Image Fusion for Noninvasive Epilepsy Surgery Planning," IEEE Computer Graphics and Applications 16, 1 (January 1996).

[Yam96]
Yam, Philip, "Magnet on the Brain," Scientific American 275, 2 (August 1996), p. 32.

[Ylä-Jääski88]
Ylä-Jääski, J. and O. Kübler, "Supporting Diagnosis and Surgical Planning by Analysis and 3-D Display of Volume Images," Image Analysis and Processing II, V. Cantoni, ed., Plenum Press, 1988, pp. 511-518.

[Zucker81]
Zucker, Steven W. and Robert A. Hummel, "A Three-Dimensional Edge Operator," IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-3, 3 (May 1981), pp. 324-331.

Acknowledgment

The 3DHEAD volumetric dataset used for this project was created with the Siemens Magnetom and was provided courtesy of Siemens Medical Systems, Inc., Iselin, NJ.

The 3DHEAD volumetric data set was made available through the kind permission of the Department of Computer Science at the University of North Carolina at Chapel Hill (see Appendix A).

APPENDIX A

AVAILABILITY OF 3-D MRI DATA FROM UNC/CHAPEL HILL (ANNOUNCE.3DH)

Announcing the Chapel Hill Volume Rendering Test Dataset, Volume I

SoftLab Software Systems Laboratory
University of North Carolina
Department of Computer Science
Chapel Hill, NC 27599-3175

The Chapel Hill Volume Rendering Test Dataset, Volume I is a collection of seven datasets comprised of the following.

Announcement - Product announcement (This document).

Installation Instructions - Electronic copy of the installation instructions (CH01) included in the distribution packet.

Head data - A 109-slice dataset of a human head. Complete slices are stored consecutively as a 256 x 256 array. Pixels consist of 2 consecutive bytes making one binary integer. Data taken on the Siemens Magnetom and provided courtesy of Siemens Medical Systems, Inc., Iselin, NJ.

Head data information article - An ASCII file containing acknowledgments for the head data files.

Knee data - A 127-slice dataset of a human knee. Complete slices are stored consecutively as a 256 x 256 array. Pixels consist of 2 consecutive bytes making one binary integer. Data taken on the Siemens Magnetom and provided courtesy of Siemens Medical Systems, Inc., Iselin, NJ.

Knee data information article - An ASCII file containing acknowledgments for the knee data files.

HIPIP data - The result of a quantum mechanical calculation of a SOD data of a one-electron orbital of HIPIP, an iron protein. This is an ASCII dataset. Provided courtesy of Louis Noodleman and David Case, Scripps Clinic, La Jolla, CA.

HIPIP information article - An ASCII file containing information about the HIPIP data.

SOD data - An electron density map of the active site of
SOD (superoxide dismutase). This is an ASCII dataset. Provided courtesy of Duncan McRee, Scripps Clinic, La Jolla, CA.

SOD histogram - An ASCII histogram of the SOD dataset. This
is described in the previous file.

SOD information article - An ASCII file containing information
about the SOD dataset.

This dataset can be purchased for a nominal charge of $50.00.
The distribution is available in two different formats. The files on the tape will be written from a DEC VAX computer using the UNIX file copy command "dd" or the UNIX "tar" command. Total block size is 8192 bytes written at 1600 bpi on either a standard 1/2" magnetic tape or a cartridge tape. Please specify your preference when ordering and note that "dd" is not available with the cartridge tape. Installation instructions also accompany the distribution.

For customers interested in Volume I, both Volume I and Volume II can be purchased as a set for $90.00, a saving of $10.00 over ordering these separately. Please remember to be specific as to what you may need.

To obtain these datasets, please contact:

Pamela M. Payne
Mail:SoftLab Coordinator
SoftLab Software Systems Laboratory
University of North Carolina
Department of Computer Science
CB# 3175, 351 Sitterson Hall
Chapel Hill, 27599-3175
Phone:(919) 962-1775
Electronic Mail:softlab@cs.unc.edu

APPENDIX B

THE GRADIENT

The gradient ∇ of a function F at a point (x, y) is the vector

$$\nabla \vec{F} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

which points in the direction of the maximum rate of change of F at (x,y) [Gonzalez92]. The vector itself is

$$\nabla F = mag(\nabla \vec{F}) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

and the magnitude of the vector is

$$\nabla F \cong |G_x| + |G_y|$$

APPENDIX C

WORLD-WIDE WEB SITES

The following list of World-Wide Web (WWW) sites features information relevant to the fields of medical informatics, visualization, NMR and MRI. The Uniform Resource Locator (URL) for each site is followed by a brief description of its contents.

www.nlm.nih.gov
National Library of Medicine, National Institute of Health; the Visible Human Project; 3-D representation of cryosections of the male and female human bodies.

www.scp.caltech.edu/~mep/ivb.html
Interactive Volume Browser of data from the Visible Human Project.

www.voxel.com
Voxel System; holographic 3-D views of CT and MR data.

imacx.wustl.edu
Mallinckrodt Institute of Technology; MIR Image Processing Lab; surgical simulation and planning; 3-D image CT and MR data available via FTP.

www.xray.ufl.edu/~rball/mritutor.html
University of Florida; downloadable computerized MRI tutorial.

www.cs.unc.edu/Research/graphics
University of North Carolina at Chapel Hill; various projects involving computer graphics, visualization, rendering, and ultrasound.

mri.med.yale.edu
Yale University Medical School; NMR Research Group.

www.crd.ge.com/esl/cgsp/projects/medical
General Electric; 3-D medical image reconstruction; MPEG animations.

poseidon.csd.auth.gr:80
Aristotle University of Thessaloniki; Department of Informatics; 3-D object reconstruction from projections; applications in dentistry.

www.nas.nasa.gov/RNR/Visualization/annotatedURLs.html
National Aeronautics and Space Administration; list of URLs for sites relating to scientific visualization.

VITA

William L. (Bill) Bell, Jr. has a Bachelor of Arts degree from the University of the State of New York Regents College in Political Science, 1985 and expects to receive a Master of Science in Computer and Information Sciences from the University of North Florida, December, 1996. Dr. Yap Siong Chua is serving as Bill's thesis adviser. Bill has been employed for the last year at Ploof Truck Lines, Inc., as a computer/network/training specialist. He also serves as an adjunct instructor at the University of North Florida, teaching computer architecture and Visual Basic programming. For the previous seven years, Bill worked at Jacksonville University in Jacksonville, Florida, serving as its Director of Microcomputing Services and as an adjunct instructor.

Bill is interested in all aspects of computer graphics, particularly biomedical applications and fractal imagery. As a student, he has presented several scholarly papers on fractal graphics and mathematics at meetings of the Florida chapter of the Mathematical Association of America. (Inspired by his recent employment in the transportation industry, he has also written a short analysis of certain optimization techniques used to match available semi-trailers with available loads.)

Bill programs mostly in C and Visual Basic these days, and has worked in BASIC, Pascal, and Lisp. He is an avid fisherman and also enjoys studying and speaking human languages, particularly French, German, Russian, Spanish, and a little English. Bill has been married to the former Kathi Lee for 10 years and has one son, Jim, age 20. Those wishing to contact Bill may reach him via electronic mail at:

wbell1@osprey.unf.edu

or can visit his homepage on the World-Wide Web at:

http://www.unf.edu/~wbell1/index.html