

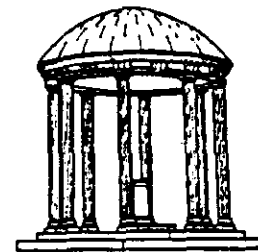
Display of Surfaces From Volume Data

TR89-022

May, 1989

Marc Levoy

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175



UNC is an Equal Opportunity/Affirmative Action Institution.

DISPLAY OF SURFACES FROM VOLUME DATA

by


Marc Levoy

A Dissertation submitted to the faculty of The University of North Carolina at Chapel Hill
in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the
Department of Computer Science.

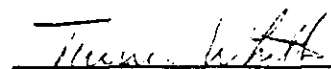
Chapel Hill

1989

Approved by:


Henry Fuchs Advisor


Stephen M. Pizer Reader


Turner Whitted Reader

(c) 1989
Marc Stewart Levoy
ALL RIGHTS RESERVED

Marc Levoy. *Display of Surfaces from Volume Data* (Under the direction of Henry Fuchs.)

ABSTRACT

Volume rendering is a technique for visualizing sampled scalar fields of three spatial dimensions without fitting geometric primitives to the data. A color and a partial transparency are computed for each data sample, and images are formed by blending together contributions made by samples projecting to the same pixel on the picture plane. Quantization and aliasing artifacts are reduced by avoiding thresholding during data classification and by carefully resampling the data during projection. This thesis presents an image-order volume rendering algorithm, demonstrates that it generates images of comparable quality to existing object-order algorithms, and offers several improvements. In particular, methods are presented for displaying iso-value contour surfaces and region boundary surfaces, for rendering mixtures of analytically defined geometry and sampled fields, and for adding shadows and textures. Three techniques for reducing rendering cost are also presented: hierarchical spatial enumeration, adaptive termination of ray tracing, and adaptive image sampling. Case studies from two applications are given: medical imaging and molecular graphics.

to Laurie

ACKNOWLEDGEMENTS

I wish first of all to extend my sincere gratitude to the members of my dissertation committee for their encouragement and support throughout this project. I owe particular thanks to Turner Whitted for guiding my early ramblings in this area, Frederick P. Brooks, Jr. for encouraging wildcat drilling at a key juncture in my research, Henry Fuchs and Stephen M. Pizer for focusing my efforts on problems in medical imaging, and Julian Rosenman for his unbridled enthusiasm regarding applications to radiation treatment planning.

Thanks are due for many enlightening discussions to fellow graduate students Lawrence M. Lifshitz, Andrew S. Glassner, John M. Gauch, Lee Westover, and Andrew Skinner, and to George W. Sherouse and Drs. Edward L. Chaney, Jordan Renner, and Richard Davis of North Carolina Memorial Hospital.

I wish to acknowledge the many anonymous reviewers of the refereed papers drawn from the material in this dissertation. I regret that I cannot thank them by name.

I wish to thank the students and facilities staff of the Department of Computer Science for providing me with logistical and programming support and for tolerating my compute-intensive jobs.

The computerized tomography studies were provided by North Carolina Memorial Hospital, the magnetic resonance study was provided by Siemens AG and edited by Juiqi Tang, and the electron density maps were obtained from Drs. Jane and Dave Richardson of Duke University and from Dr. Chris Hill of the University of York. The texture map is from the feature film *Heidi* by Hanna-Barbera Productions.

Financial support for this work came from ONR grant N00014-86-K-0680, NIH Division of Research Resources grant RR02170-05, NCI grant P01-CA47982, and IBM.

Finally, I want to thank my wife, Laurie Winslow. In appreciation of her sacrifices, patience, and moral support throughout my graduate career, I dedicate this book to her.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	xi
LIST OF ABBREVIATIONS	xii
LIST OF SYMBOLS	xiii
I. INTRODUCTION	1
II. DISPLAYING SURFACES FROM VOLUME DATA	3
2.1. Background	3
2.2. Brute-force rendering algorithm	4
2.2.1. Calculation of voxel colors	5
2.2.2. Calculation of voxel opacities	6
2.2.2.1. Isovalue contour surfaces in electron density maps	6
2.2.2.2. Region boundary surfaces in 3D medical data	7
2.2.3. Volumetric compositing	8
2.3. Implementation details	9
2.4. Simple techniques for reducing computational expense	11
2.5. Simple techniques for improving image quality	11
2.6. Case studies	12
2.7. Summary and discussion	13
III. REDUCING THE COST OF TRACING A RAY	23
3.1. Background	23
3.2. Two optimization techniques	24
3.2.1. Hierarchical enumeration of dataset	24
3.2.2. Adaptive termination of ray tracing	24
3.3. Implementation details	25
3.4. Comparison to ray tracing of geometrically defined scenes	27
3.5. Case studies	27
3.6. Summary and discussion	28
IV. REDUCING THE NUMBER OF RAYS TRACED	36
4.1. Background	36
4.2. Adaptive volume rendering algorithm	37
4.3. Implementation details	37
4.4. Case studies	39

4.5. Summary and discussion	40
V. RENDERING MIXTURES OF GEOMETRIC AND VOLUME DATA	47
5.1. Background	47
5.2. Two rendering algorithms	48
5.2.1. Hybrid ray tracer	48
5.2.2. 3D scan-conversion	49
5.3. Extensions to the rendering algorithms	50
5.3.1. Shadow calculations	50
5.3.2. Texture mapping	51
5.4. Optimization of the rendering algorithms	52
5.5. Implementation details	52
5.6. Case studies	57
5.7. Summary and discussion	58
VI. FINAL SUMMARY AND TOPICS FOR FUTURE RESEARCH	68
6.1. Comparison to Pixar volume rendering algorithm	68
6.2. Real-time volume rendering	69
6.3. User interface design issues	70
6.4. Visualization of multiple fields	71
6.5. How correct is a volume rendering?	71
REFERENCES	73

LIST OF FIGURES

Figure 2.1: Overview of brute-force volume rendering algorithm	15
Figure 2.2: Coordinate systems used in brute-force algorithm	15
Figure 2.3: Ray tracing and resampling steps	16
Figure 2.4: Volumetric compositing calculations	16
Figure 2.5: Calculation of opacities for isovalue contour surface	17
Figure 2.6: Calculation of opacities for region boundary surfaces	17
Figure 2.7: Representative slices from $113 \times 113 \times 113$ voxel electron density map of cytochrome B5	18
Figure 2.8: Volume rendering of isovalue contour surface from dataset shown in figure 2.7	18
Figure 2.9: Volume renderings of region boundary surfaces from $256 \times 256 \times 113$ voxel CT dataset of human head	19
Figure 2.10: Rotated view of same dataset	19
Figure 2.11: Rendering of $512 \times 512 \times 113$ voxel CT dataset	20
Figure 2.12: Rendering of same dataset after interpolation to $512 \times 512 \times 452$ voxels	20
Figure 2.13: Color rendering of CT dataset showing bone, soft tissue, and 3D region of interest formed by scaling down opacity of selected voxels	21
Figure 2.14: View of same dataset following repositioning of region of interest and light source	21
Figure 2.15: Original and edited slices and volume renderings of $256 \times 256 \times 156$ voxel MR dataset of human head	22
Figure 2.16: Rotated view of edited dataset	22
Figure 3.1: Hierarchical enumeration of object space for $N = 5$	30
Figure 3.2: Ray tracing of hierarchical enumeration	30

Figure 3.3: Rendering of $256 \times 128 \times 59$ voxel CT dataset of human jaw with lucite skin after interpolation to $256 \times 128 \times 113$ voxels	31
Figure 3.4: View of same dataset with skin rendered transparently	31
Figure 3.5: Rendering of $24 \times 20 \times 11$ voxel electron density map of ribonuclease after interpolation to $288 \times 244 \times 132$ voxels	32
Figure 3.6: Rendering of $256 \times 256 \times 113$ voxel CT dataset of human head after interpolation to $256 \times 256 \times 226$ voxels	32
Figures 3.7a and 3.7b: Constituent costs of rendering figure 3.6 using brute-force algorithm	33
Figures 3.8a and 3.8b: Constituent costs of rendering figure 3.6 using hierarchical enumeration	33
Figures 3.9a and 3.9b: Constituent costs of rendering figure 3.6 using hierarchical enumeration and adaptive termination of ray tracing	34
Figure 4.1: Overview of adaptive volume rendering algorithm	41
Figure 4.2: Recursive subdivision of image plane	41
Figure 4.3: Adaptive rendering of electron density map, state of image after 5, 12, 17, and 26 seconds of computation	42
Figure 4.4: Visualization of where rays were cast to generate figure 4.3	42
Figure 4.5: Adaptive rendering of CT dataset, state of image after 13 seconds of computation	43
Figure 4.6: Continuation of same rendering, state of image after 104 seconds of computation	43
Figure 4.7: Adaptive rendering of MR dataset, state of image after 18 seconds of computation	44
Figure 4.8: Continuation of same rendering, state of image after 120 seconds of computation	44
Figure 5.1: Overview of hybrid ray tracer for rendering mixtures of geometric and volume data	59
Figure 5.2: Rendering of polygon embedded in volume data	60

Figure 5.3: Overview of 3D scan-conversion method for rendering mixtures of geometric and volume data	61
Figure 5.4: Addition of shadow calculations to 3D scan-conversion method	62
Figure 5.5: Additional coordinate systems used in shadowing and texturing	63
Figure 5.6a: Color rendering of CT dataset and embedded polygons, generated using hybrid ray tracer	64
Figure 5.6b: Color rendering of CT dataset and embedded polygons, generated using 3D scan-conversion method	64
Figures 5.7a and 5.7b: Details from figures 5.6a and 5.6b, comparing image quality of hybrid ray tracer and 3D scan-conversion methods	65
Figures 5.8a and 5.8b: Visualization of where rays were cast to generate figures 5.6a and 5.6b	65
Figure 5.9: Color rendering with shadows of CT dataset and embedded polygons, generated using modified 3D scan-conversion method	66
Figure 5.10: Color rendering of CT dataset and textured polygons, generated using modified hybrid ray tracer	66
Figure 5.11: Color rendering of CT dataset showing bone, soft tissue, tumor (purple), and radiation treatment beam (blue)	67
Figure 5.12: Color rendering with shadows of electron density map and polygons	67

LIST OF TABLES

Table 3.1:	Characteristics of datasets shown in figures 3.3 through 3.6	35
Table 3.2:	Rendering times for datasets characterized in table 3.1	35
Table 4.1:	Performance statistics for adaptive rendering of electron density map	45
Table 4.2:	Performance statistics for adaptive rendering of CT dataset	45
Table 4.3:	Performance statistics for adaptive rendering of MR dataset	46

LIST OF ABBREVIATIONS

ONR	Office of Naval Research
NIH	National Institutes of Health
NCI	National Cancer Institute
IBM	International Business Machines
ACM	Association for Computing Machinery
SIGGRAPH	Special Interest Group for computer GRAPHics and interactive techniques
NCGA	National Computer Graphics Association
C(A)T	Computerized (Axial) Tomography
MR(I)	Magnetic Resonance (Imaging)

LIST OF SYMBOLS

$\mathbf{x} = (x, y, z)$	object space position for reals $1 \leq x, y, z \leq N$
$\mathbf{i} = (i, j, k)$	voxel index for integers $i, j, k = 1, \dots, N$
$\mathbf{u} = (u, v)$	pixel or ray index for integers $u, v = 1, \dots, P$
$\mathbf{U} = (u, v, w)$	image space sample index for ray cast from pixel (u, v) and integer distance $w = 1, \dots, W$ along the ray with $w = 1$ being closest to the eye
$\mathbf{u}_s = (u_s, v_s)$	illumination ray index for integers $u_s, v_s = 1, \dots, P_s$ and light source $s = 1, \dots, S$
$\mathbf{U}_s = (u_s, v_s, w_s)$	light source space sample index for illumination ray (u_s, v_s) and integer distance $w_s = 1, \dots, W_s$ along the ray with $w_s = 1$ being closest to the light source
f	real scalar value of field being visualized
C	real scalar or vector color
α	real opacity where $0 \leq \alpha \leq 1$ and $\alpha = 1$ means complete attenuation
β	real light strength where $0 \leq \beta \leq 1$ and $\beta = 0$ means complete attenuation
V	binary value from pyramid of binary volumes

CHAPTER I

INTRODUCTION

Visualization of scientific and medical data is a rapidly growing field within computer graphics. A large subset of these applications involve sampled scalar fields, also known as volume data. Surfaces are commonly used to visualize volume data because they succinctly present the 3D configuration of complicated objects present in the data. In this thesis, we explore the use of isovalue contour surfaces to visualize molecular electron density maps and the use of region boundary surfaces to visualize computed tomography (CT) and magnetic resonance (MR) data. Although we focus on display of surfaces, the algorithms described in this thesis can be modified to render partially transparent volumes as well.

Previous techniques for displaying surfaces from volume data fall into two broad categories: *surface-based techniques* in which geometric surface primitives are fit to the sample array, and *binary voxel techniques* in which the sample array is converted into a binary voxel representation and projected directly onto the picture plane. Both approaches require making a binary classification of the incoming data. In the presence of small or poorly defined features, error-free binary classification is often impossible. Errors in classification manifest themselves as visual artifacts in the generated image. These artifacts are ubiquitous, distracting, and have hindered acceptance of these visualizations by the user community.

This thesis explores a visualization technique called *volume rendering* which is closely related to the binary voxel techniques, but does not require binary classification of the data. Images are formed by computing a color and partial transparency for all data samples and blending together contributions made by samples projecting to similar points on the picture plane. The omission of a binary classification step does not preclude the display of surfaces as will be demonstrated. The key improvement offered by volume rendering is that it provides a natural mechanism for representing classification uncertainty and thus for displaying small, weak, or fuzzy features.¹

Chapter 2 surveys previous techniques for displaying surfaces from volume data, presents a new volume rendering algorithm based on ray tracing, demonstrates that it generates images of comparable quality to other volume rendering algorithms, and describes techniques for displaying isovalue contour surfaces and region boundary surfaces using the new algorithm. The material in this chapter first appeared in [Levoy87]. In its present form, it represents the union of [Levoy88a] and [Levoy88b].

Chapter 3 presents two techniques for taking advantage of spatial coherence in volume data to reduce the cost of tracing a ray: hierarchical spatial enumeration and adaptive termination of ray tracing. The speedups obtained using these optimizations are highly dependent on the depth complexity of the scene. For the datasets studied, combined savings of up to an order of

¹There is currently some confusion in the literature regarding the terminology used to describe these techniques. Volume rendering has been defined in the image processing field to encompass any display method based on overpainting of voxels. In the computer graphics literature, it has come to denote only techniques based on blending of semi-transparent voxels. This thesis follows computer graphics usage. A more specific term, *volumetric compositing*, is reserved for that portion of a volume rendering algorithm specifically related to the blending calculations. Further discussion of this issue can be found in [Reynolds89, Levoy89b].

magnitude over brute-force algorithms have been observed. These techniques were first reported in [Levoy88c] and are summarized in [Fuchs89a] and [Fuchs89c].

Chapter 4 discusses the use of adaptive image sampling for taking advantage of spatial coherence in images generated from volume data to reduce the number of rays traced. The technique can also be used to progressively refine image quality over an interval of time. Using this approach, speedups of up to an order of magnitude with little degradation in subjective image quality have been observed. The technique first appeared in [Levoy88d], will appear in revised form in [Levoy89d], and is summarized in [Fuchs89a] and [Fuchs89c].

Chapter 5 presents two techniques for extending volume rendering to handle polygonally defined objects. The first method employs a hybrid ray tracer capable of handling both geometric and volume data. The second consists of 3D scan-converting the geometric primitives into the volume dataset and rendering the resulting ensemble. Techniques are also described for casting shadows through mixtures of geometric and volume data and for adding texture to volume renderings. This material was first reported in [Levoy88e] and is summarized in [Fuchs89a] and [Fuchs89c].

Chapter 6 compares the algorithm presented in this thesis with the approach taken by researchers at Pixar, discusses some of the unsolved problems in volume rendering, and suggests topics for future research. Portions of this material first appeared in [Levoy89a] and will appear in revised form in [Levoy89c, Fuchs89a, Fuchs89b, Fuchs89c].

CHAPTER II

DISPLAYING SURFACES FROM VOLUME DATA

2.1. Background

The currently dominant technique for displaying surfaces from volume data consists of applying a surface detector to the sample array, fitting geometric primitives to the detected surfaces, then rendering the resulting geometric representation. These *surface-based techniques* differ from one another mainly in the choice of primitives and the scale at which they are defined. In the medical imaging field, a common approach is to apply thresholding to the volume data. The resulting binary array can be rendered by treating 1's as opaque cubes having six polygonal faces [Herman79]. This approach has been termed the *cube model* [Chen85]. Alternatively, edge tracking can be applied on each slice to yield a set of contours defining features of interest, then a mesh of polygons can be constructed connecting the contours on adjacent slices [Fuchs77, Pizer86]. For displaying isovalue surfaces, polygons can be fit to an approximation of the continuous scalar field within each voxel [Lorensen87, Cline88]. Other techniques based on fitting of geometric primitives are surveyed in [Herman82].

Another broad category of methods for displaying surfaces from volume data are the *binary voxel techniques* in which data samples are mapped directly to image pixels, omitting the intermediate geometric representation. Hidden-surface removal is commonly implemented by thresholding the data and painting voxels in back-to-front [Frieder85] or front-to-back [Gordon85] order. Alternatively, rays can be traced from an observer position through the data, stopping when an opaque object is encountered [Goldwasser85, Goldwasser86a, Schlüsselberg86, Troussel87, Hoehne87, Hoehne88a]. Because volume data samples, unlike geometric primitives, have no defined extent, resampling becomes an important issue. Zeroth or first order interpolation is commonly used. If the binary representation is augmented with the local grayscale gradient at each voxel, substantial improvements in surface shading can be obtained [Hoehne86, Goldwasser86b, Schlüsselberg86, Troussel87].

All of these techniques suffer from the common problem of having to make a binary classification decision at some stage of the rendering process: either a surface passes through the current voxel or it does not. Since classification is performed on a bandlimited representation of the original scene, small or poorly defined features are often incorrectly classified. When the results of the erroneous classification are displayed, they appear as image artifacts, specifically spurious surfaces (false positives) or erroneous holes in surfaces (false negatives).

To avoid these problems, researchers have begun exploring *volume rendering*, a variant of the binary voxel techniques in which a color and a partial opacity is assigned to each voxel. Images are formed from the resulting colored semi-transparent volume by blending together voxels projecting to similar points on the picture plane. The omission of binary classification does not preclude the display of surfaces. The key improvement offered by volume rendering is that it eliminates the necessity of making a binary classification of the data, thus providing a mechanism for displaying poorly defined features.

Early predecessors of this technique include the use of structured systems of particles or points to model smoke [Csuri79], fire [Reeves83], vegetation [Reeves85], and geometrically

defined surfaces [Catmull74, Rubin80, Levoy85]. More closely related to the present technique is the use of spatially ordered volume densities to model clouds [Blinn82] and other atmospheric phenomena [Kajiya84].

Researchers at Pixar, Inc. of San Rafael, California appear to be the first to use volume rendering. Their technique was demonstrated publicly at NCGA '85, described in general terms in [Smith87], and presented in detail in [Drebin88]. It consists of estimating occupancy fractions for each of a set of materials that might be present in a voxel, computing from these fractions a color and a partial opacity for each voxel, geometrically transforming each slice of voxels from object-space to image-space, projecting it onto the image plane, and blending it together with the projection formed by previous slices.

The algorithm presented in this chapter was developed independently of Pixar's. It is similar in general approach, but computes colors and opacities directly from the scalar value of each voxel and renders the resulting volume by tracing viewing rays from an observer position through the dataset. It is not clear that omission of an explicit intermediate material occupancy representation imposes any fundamental limitations. The use of an image-order rather than an object-order rendering algorithm has significant advantages, however, as will be demonstrated in chapters 3 and 4. A more comprehensive comparison of these two approaches is given in chapter 6.

Recent advances in volume rendering include alternative shading models for displaying statistical properties of datasets [Sabella88], more accurate visibility calculations for displaying numerical simulation data [Upson88], a parallelizable object-order volume rendering algorithm [Westover89], and application of volume rendering techniques to diagnostic radiology [Scott87, Fishman87].

2.2. Brute-force rendering algorithm

The remainder of this chapter is devoted to consideration of the brute-force volume rendering algorithm outlined in figure 2.1. We begin with a 3D array of scalar values. Depending on the application, preparation of this array may require a number of pre-processing steps such as correction for non-orthogonal sampling grids in electron density maps, correction for patient motion in computed tomography (CT) data, contrast enhancement, and interpolation of additional samples. For simplicity, let us assume that the array forms a cube measuring N voxels on a side. In this thesis, we treat voxels as point samples of a continuous function rather than as volumes of homogeneous value. Voxels are indexed by a vector $i = (i,j,k)$ where $i,j,k = 1, \dots, N$, and the value of voxel i is denoted $f(i)$. This array is used as input to the shading model described in section 2.2.1, yielding a color $C(i)$ for each voxel. Color is either a scalar (producing a monochrome image) or a three-component vector (red, green, blue); both are used in this thesis. In a separate step, the array is used as input to one of the classification procedures described in section 2.2.2, yielding an opacity $\alpha(i)$ for each voxel.

Parallel rays are then traced into the data from an observer position as shown in figure 2.2. Let us assume that the image is a square measuring P pixels on a side, and that one ray is cast per pixel. Pixels and hence rays are indexed by a vector $u = (u,v)$ where $u,v = 1, \dots, P$. For each ray, a vector of colors and opacities is computed by resampling the data at W evenly spaced locations along the ray and trilinearly interpolating from the colors and opacities in the eight voxels surrounding each sample location as shown in figure 2.3. Samples are indexed by a vector $U = (u,v,w)$ where (u,v) identifies the ray, and $w = 1, \dots, W$ corresponds to distance along the ray with $w = 1$ being closest to the eye. The color and opacity of sample U are denoted $C(U)$ and $\alpha(U)$ respectively. Finally, a fully opaque background is draped behind the dataset, and the resampled colors and opacities are composited with each other and with the background as described in section 2.2.3 to yield a color for the ray. This color is denoted $C(u)$.

2.2.1. Calculation of voxel colors

Using the rendering algorithm presented above, the mapping from scalar value to color provides 3D shape cues, but does not participate in the classification operation. Accordingly, a shading model was selected that provides a satisfactory illusion of smooth surfaces at a reasonable cost. The model chosen is due to [Phong75]:

$$C(i) = \frac{1}{k_1 + k_2 d(i)} \left[C_a k_a + \sum_{s=1}^S C_s \left[k_d (N(i) \cdot L_s) + k_s (N(i) \cdot H_s)^n \right] \right] \quad (2.1)$$

for parallel light sources $s = 1, \dots, S$ where

$C(i)$ = color of voxel i ,

C_a = color of ambient light source,

C_s = color of light source s ,

k_a = ambient reflection coefficient of surface,

k_d = diffuse reflection coefficient of surface,

k_s = specular reflection coefficient of surface,

n = exponent used to approximate specular highlight,

k_1, k_2 = constants used in linear approximation of depth-cueing,

$d(i)$ = perpendicular distance from voxel i to the observer,

$N(i)$ = surface normal at voxel i ,

L_s = normalized vector in direction of light source s ,

H_s = surface normal yielding maximum highlight due to light source s .

Since parallel light sources are used, the L_s 's are constants. Furthermore,

$$H_s = \frac{V + L_s}{|V + L_s|}$$

where

V = normalized vector in direction of observer.

Since an orthographic projection is used, V and hence each H_s is constant. Finally, the surface normal is given by

$$N(i) = \frac{\nabla f(i)}{|\nabla f(i)|}$$

where the gradient vector $\nabla f(i)$ is approximated using the operator

$$\nabla f(i) = \nabla f(i,j,k) = \left[\frac{1}{2} [f(i+1,j,k) - f(i-1,j,k)], \frac{1}{2} [f(i,j+1,k) - f(i,j-1,k)], \frac{1}{2} [f(i,j,k+1) - f(i,j,k-1)] \right]$$

2.2.2. Calculation of voxel opacities

The mapping from acquired data to opacity performs the essential task of surface classification. We will first consider the rendering of isovalue contour surfaces in electron density maps, i.e. surfaces defined by points of equal electron density. Next, we will consider the rendering of region boundary surfaces in computed tomography (CT) and magnetic resonance (MR) data, i.e. surfaces bounding tissues of constant CT or MR number.

2.2.2.1. Isovvalue contour surfaces in electron density maps

Determining the structure of large molecules is a difficult problem. The method most commonly used is *ab initio* interpretation of electron density maps, which represent the averaged density of a molecule's electrons as a function of position in 3-space. These maps are obtained from X-ray diffraction studies of crystallized samples of the molecule [Glusker85]. Current methods for visualizing electron density maps include stacks of isovalue contour lines, ridge lines arranged in 3-space so as to connect local maxima [Williams82], and basket meshes representing isovalue contour surfaces [Purvis86].

One obvious way to display isovalue contour surfaces directly from a sample array is to opaquely render all voxels having values greater than some threshold. This produces 3D regions of opaque voxels the outermost layer of which is the desired isovalue surface. Unfortunately, this solution prevents display of multiple concentric semi-transparent surfaces, a very useful capability. Using a window in place of a threshold does not solve the problem. If the window is too narrow, holes appear. If it too wide, display of multiple surfaces is constrained. In addition, the use of thresholds and windows introduces artifacts into the image that are not present in the data.

The classification procedure employed in this thesis begins by assigning an opacity α_v to voxels having selected value f_v , and assigning an opacity of zero to all other voxels. In order to avoid aliasing artifacts, we would also like voxels having values close to f_v to be assigned opacities close to α_v . The most pleasing image is obtained if the thickness of this transition region stays constant throughout the volume. We approximate this effect by having the opacity fall off as we move away from the selected value at a rate inversely proportional to the magnitude of the local gradient vector.

This mapping is implemented using the expression

$$\alpha(i) = \alpha_v \begin{cases} 1 & \text{if } |\nabla f(i)| = 0 \text{ and } f(i) = f_v \\ 1 - \frac{1}{r} \frac{|f_v - f(i)|}{|\nabla f(i)|} & \text{if } |\nabla f(i)| > 0 \text{ and } f(i) - r|\nabla f(i)| \leq f_v \leq f(i) + r|\nabla f(i)| \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

where r is the desired thickness in voxels of the transition region and the gradient vector is approximated using the operator given in section 2.2.1. A graph of $\alpha(i)$ as a function of $f(i)$ and $|\nabla f(i)|$ for typical values of f_v , α_v , and r is shown in figure 2.5.

If more than one isovalue surface is to be displayed in a single image, they can be classified separately and their opacities combined. Specifically, given selected values f_{v_n} , $n = 1, \dots, N$, $N \geq 1$, opacities α_n , and transition region thicknesses r_n , we can use equation (2.2) to compute $\alpha_n(i)$, then apply the relation

$$\alpha_{tot}(i) = 1 - \prod_{n=1}^N (1 - \alpha_n(i)). \quad (2.3)$$

2.2.2.2. Region boundary surfaces in 3D medical data

From a densitometric point of view, the human body is a complex arrangement of biological tissues each of which is fairly homogeneous and of predictable composition. Clinicians are mostly interested in the boundaries between tissues, from which the sizes and spatial relationships of anatomical features can be inferred.

Although many researchers use isovalue contour surfaces for the display of 3D medical data, it is not clear that they are well suited for that purpose. The reason can be explained briefly as follows. Given an anatomical scene containing two tissue types A and B having values f_{v_A} and f_{v_B} where $f_{v_A} < f_{v_B}$, data acquisition will produce voxels having values $f(i)$ such that $f_{v_A} \leq f(i) \leq f_{v_B}$. Thin features of tissue type B may be represented by regions in which all voxels bear values less than f_{v_B} . Indeed, there is no threshold value greater than f_{v_A} guaranteed to detect arbitrarily thin regions of type B , and thresholds close to f_{v_A} are as likely to detect noise as signal.

The procedure employed in this thesis is based on the following simplified model of anatomical scenes and the CT (or MR) scanning process. We assume that scenes contain an arbitrary number of tissue types bearing CT numbers falling within a small neighborhood of some known value. We further assume that tissues of each type touch tissues of at most two other types in a given scene. Finally, we assume that, if we order the types by CT number, then each type touches only types adjacent to it in the ordering. Formally, given N tissue types bearing CT numbers f_{v_n} , $n = 1, \dots, N$, $N \geq 1$ such that $f_{v_m} < f_{v_{m+1}}$, $m = 1, \dots, N-1$, then no tissue of CT number $f_{v_{n_1}}$ touches any tissue of CT number $f_{v_{n_2}}$, $|n_1 - n_2| > 1$.

If these criteria are met, each tissue type can be assigned an opacity and a piecewise linear mapping can be constructed that converts voxel value f_{v_n} to opacity α_n , voxel value $f_{v_{m+1}}$ to opacity α_{m+1} , and intermediate voxel values to intermediate opacities. Note that all voxels are typically mapped to some non-zero opacity and will thus contribute to the final image. This scheme insures that thin regions of tissue will still appear in the image, even if only as faint wisps. Note also that violation of the adjacency criteria leads to voxels that cannot be unambiguously classified as belonging to one region boundary or another and hence cannot be rendered correctly using this method.

The superimposition of multiple semi-transparent surfaces such as skin and bone can substantially enhance the comprehension of CT or MR data. In order to obtain such effects using volume rendering, we would like to suppress the opacity of tissue interiors while enhancing the opacity of their bounding surfaces. We implement this by scaling the opacities computed above by the magnitude of the local gradient vector.

Combining these two operations, we obtain a set of expressions

$$\alpha(l) = |\nabla f(l)| \begin{cases} \alpha_{v_{n+1}} \left[\frac{f(l) - f_{v_n}}{f_{v_{n+1}} - f_{v_n}} \right] + \alpha_{v_n} \left[\frac{f_{v_{n+1}} - f(l)}{f_{v_{n+1}} - f_{v_n}} \right] & \text{if } f_{v_n} \leq f(l) \leq f_{v_{n+1}} \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

for $n = 1, \dots, N-1, N \geq 1$. The gradient vector is approximated using the operator given in section 2.2.1. A graph of $\alpha(l)$ as a function of $f(l)$ and $|\nabla f(l)|$ for three tissue types A, B, and C, having typical values of $f_{v_A}, f_{v_B}, f_{v_C}, \alpha_{v_A}, \alpha_{v_B}$, and α_{v_C} is shown in figure 2.6.

2.2.3. Volumetric compositing

The blending of colors and opacities along a viewing ray is performed using *volumetric compositing*, an approximation to the visibility calculations required to render a semi-transparent gel. The following development is adopted loosely from [Blinn82]. The visibility method derived in [Sabella88] for a varying density emitter follows similar lines. Let us define a gel as a transparent medium in which a large number of opaque spherical particles of fixed radius, non-uniform distribution, and varying reflectance are suspended. Our approximation considers the effect of inter-particle shadowing along the line of sight, but ignores inter-particle shadowing along lines of illumination and ignores inter-particle scattering.

Figure 2.4 shows the rectangular beam defined by projecting a pixel through image space. Let us decompose this beam into slabs numbered 1 through W , front-to-back, each having unit volume. Let us further assume that the density and brightness of particles in a single slab is fixed, i.e. slab w in the figure contains exactly n_w randomly distributed particles of radius p and brightness B_w . Let us now consider the brightness due to a cylindrical sub-beam of radius p as shown in the figure. The intersection of each slab with the sub-beam defines a sub-slab having volume V_w . If the density of particles in each slab is low, and we consider a particle to lie in a sub-slab only if the particle center lies within the sub-slab boundaries, then the probability that one or more particles occupies sub-slab w is given by the Poisson density

$$P(>0; V_w) = 1 - P(0; V_w) = 1 - e^{-n_w V_w}. \quad (2.5)$$

If there are one or more particles in sub-slab w and no particles in sub-slabs 1 through $w-1$, then the brightness seen at the top of the cylinder will be brightness B_w . Since each slab is independent, the joint probability of this event is given by

$$P(>0; V_w; 0; V_1, \dots, 0; V_{w-1}) = P(>0; V_w) P(0; V_1) \cdots P(0; V_{w-1}) = (1 - e^{-n_w V_w}) \prod_{\alpha=1}^{w-1} e^{-n_\alpha V_\alpha}. \quad (2.6)$$

The expected brightness due to the entire rectangular beam is then given by

$$B = \sum_{w=1}^W \left[B_w (1 - e^{-n_w V_w}) \prod_{\alpha=1}^{w-1} e^{-n_\alpha V_\alpha} \right]. \quad (2.7)$$

Volume terms drop out because they sum to unity.

We can simplify this expression slightly by defining the opacity α_w of unit volume slab w using the exponential relation [Johns83]

$$\alpha_w = 1 - e^{-n_w V_w}. \quad (2.8)$$

Substituting, the expected brightness is now given by

$$B = \sum_{w=1}^W \left[B_w \alpha_w \prod_{\alpha=1}^{w-1} (1 - \alpha_\alpha) \right]. \quad (2.9)$$

Solving for the brightness $B_{1, \dots, w}$ due to slabs 1 through w in terms of the brightness $B_{1, \dots, w-1}$ due to slabs 1 through $w-1$ and the brightness B_w and opacity α_w of slab w gives us the relation

$$B_{1, \dots, w} \alpha_{1, \dots, w} = B_{1, \dots, w-1} \alpha_{1, \dots, w-1} + B_w \alpha_w (1 - \alpha_{1, \dots, w-1}) \quad (2.10a)$$

where

$$\alpha_{1, \dots, w} = \alpha_{1, \dots, w-1} + \alpha_w (1 - \alpha_{1, \dots, w-1}) \quad (2.10b)$$

This formula appears frequently in the image compositing literature [Levoy78, Wallace81, Porter84] as a method for combining colored partially transparent 2D images. The latter two papers derive alternative formulations for compositing from back to front or from front to back (as above) with equivalent results. The volume rendering algorithms in [Levoy88b] and [Drebin88] process data from back to front, while the algorithms in [Sabella88] and [Upson88] operate from front to back. In the present algorithm, we work from front to back, compositing the color and opacity at each sample location *under* the ray in the sense of [Porter84]. Specifically, the color $C_{out}(u;U)$ and opacity $\alpha_{out}(u;U)$ of ray u after processing sample U is related to the color $C_{in}(u;U)$ and opacity $\alpha_{in}(u;U)$ of the ray before processing the sample and the color $C(U)$ and opacity $\alpha(U)$ of the sample by the relation

$$\hat{C}_{out}(u;U) = \hat{C}_{in}(u;U) + \hat{C}(U)(1 - \alpha_{in}(u;U)) \quad (2.11a)$$

and

$$\alpha_{out}(u;U) = \alpha_{in}(u;U) + \alpha(U)(1 - \alpha_{in}(u;U)). \quad (2.11b)$$

where $\hat{C}_{in}(u;U) = C_{in}(u;U)\alpha_{in}(u;U)$, $\hat{C}_{out}(u;U) = C_{out}(u;U)\alpha_{out}(u;U)$, and $\hat{C}(U) = C(U)\alpha(U)$.

After all samples along a ray have been processed, the color $C(u)$ of the ray is obtained from the expression $C(u) = \hat{C}_{out}(u;W) / \alpha_{out}(u;W)$ where $W = (u, v, W)$. If a fully opaque background is draped behind the dataset at $w' = W + 1$ and composited under the ray after it has passed through the data, then $\alpha_{out}(u;W') = 1$ where $W' = (u, v, w')$, and this normalization step can be omitted.

2.3. Implementation details

The complete brute-force rendering algorithm is summarized in pseudo-code as follows:

```

procedure RenderVolume1( ) begin
    {Compute color and opacity for each voxel in dataset}
    for all  $i$  in Dataset do begin
        ComputeOpacity( $i$ );
        if  $\alpha(i) > 0$  then
            ComputeColor( $i$ );
    end
    {Trace ray from each pixel in image}
    for all  $u$  in Image do
        TraceRay1( $u$ );
    DisplayImage1( );
end RenderVolume1.

```

```

procedure TraceRay1(u) begin
    Ĉ(u) := 0;
    α(u) := 0;
    x1 := First(u);
    x2 := Last(u);
    U1 := [Image(x1)];
    U2 := [Image(x2)];

    (Loop through all samples falling within data)
    for U := U1 to U2 do begin
        x := Object(U);

        (If sample opacity > 0,)
        (then resample color and composite into ray)
        α(U) := Sample(α,x);
        if α(U) > 0 then begin
            Ĉ(U) := Sample(Ĉ,x);
            Ĉ(u) := Ĉ(u) + Ĉ(U)(1 - α(u));
            α(u) := α(u) + α(U)(1 - α(u));
        end
    end
end TraceRay1.

```

The *ComputeOpacity* procedure calculates the opacity of a voxel using one of equations (2.2) or (2.4) and loads it into an array. The *ComputeColor* procedure calculates the color of a voxel using equation (2.1) and loads it into another array. The *TraceRay₁* procedure traces a ray into the arrays of colors and opacities loads the resulting color into an image array. The *Display-Image₁* procedure displays the image array.

The *First* and *Last* procedures accept a ray index and return the object-space coordinates of the points where the ray enters and leaves the data respectively. These coordinates are denoted by real vectors of the form $x = (x,y,z)$ where $1 \leq x,y,z \leq N$. The *Object* and *Image* procedures convert between object-space coordinates and image-space coordinates. Although these calculations normally require matrix multiplications, they can be simplified for the restricted case of an orthographic viewing projection by retaining the coordinates computed in the previous invocation and using differencing. The *Sample* procedure accepts a 3D array of colors or opacities and the object-space coordinates of a point, and returns an approximation to the color or opacity at that point by trilinearly interpolating from the eight surrounding voxels.

The minimum memory required for the algorithm is $2N^3$ bytes to hold a monochrome color and opacity for each voxel and P^2 bytes to hold a monochrome output image. The time required to calculate voxel opacities is proportional to the number of voxels in the dataset. Given scalar value $f(i)$ and gradient magnitude $|\nabla f(i)|$, the computation of each opacity $\alpha(i)$ can be implemented with one lookup table reference. The time required to calculate voxel colors is proportional to the number of non-empty voxels (voxels whose opacity is non-zero). Given scalar value $f(i)$, surface normal vector $N(i)$, and a pre-computed table of depth cueing attenuation fractions, the computation of each color $C(i)$ requires ten multiplications, six additions, and one exponentiation per light source.

The cost of finding all non-empty samples along a viewing ray is proportional to the length of the ray clipped to the boundaries of the dataset. If we assume an orthographic viewing projection, in which case sample coordinates can be efficiently calculated using differencing, the testing of each sample opacity $\alpha(U)$ requires three additions, one trilinear interpolation, and a comparison. The cost of compositing non-empty samples is proportional to the number found. Since sample opacity $\alpha(U)$ has already been computed, the computation of sample color $C(U)$ and new ray color $C(u)$ requires only one additional trilinear interpolation and two linear interpolations.

2.4. Simple techniques for reducing computational expense

This algorithm consists of several steps: shading, classification, ray tracing, resampling, and compositing. Each step is controlled by user-selectable parameters and produces as output a sampled scalar or vector-valued volume. For animation sequences in which only a subset of the controlling parameters change from frame to frame, these intermediate results can be stored in arrays, and only those calculations whose parameters change need be repeated on each frame.

A common type of sequence is one in which the object and light sources are fixed and the observer moves. If specular reflection is removed from the shading model, voxel color becomes invariant and need be computed only once. This optimization substantially reduces image generation time, but the consequent lack of continually changing surface reflections makes it difficult to reliably distinguish surface orientation from surface albedo.

If the light sources move relative to the object, but the observer stays motionless, the depth along each viewing ray at which the first non-empty voxel is encountered does not change. This depth can be recorded in an array during generation of the first frame in a sequence and used to speed generation of subsequent frames. Hoehne reports success using a similar depth buffer in his own work [Hoehne88b]. If the shading model includes multiple light sources only one of which is moving, the contribution made by the stationary sources can be pre-computed and added on each frame to the contribution computed for the moving source (assuming that multiple scattering effects are ignored).

Another common type of sequence is one in which the object, light source, and the observer are all fixed, and only voxel opacities are changed. For example, users frequently ask for some means of highlighting and interactively moving a 3D region of interest. The notion of treating the voxels inside a defined region differently from the rest of a dataset has been explored extensively by Hoehne [Hoehne87, Hoehne88a]. In the context of volume rendering, one way to highlight such a region is to increase the opacity of voxels inside the region and to decrease the opacity of voxels outside the region. In some cases (such as figures 2.13 and 2.14), it is preferable to perform the inverse transformation, decreasing the opacities of voxels inside the region of interest.

As a final note, the local gradient vector at each voxel is a function only of the input data and does not depend on any of the controlling parameters. If this vector is pre-computed for all voxels, calculation of new opacities following a change in classification parameters entails only generation of a new lookup table followed by one table reference per voxel.

2.5. Simple techniques for improving image quality

Although the notation used in equation (2.11) has been borrowed from the literature of image compositing, the analogy is not exact, and the differences are fundamental. Volume data consists of samples taken from a bandlimited 3D scene, whereas the data acquired from an image digitizer consists of samples taken from a bandlimited 2D projection of a 3D scene. Unless we reconstruct the original scene that gave rise to our volume data, we cannot compute an accurate projection of it. Volume rendering reconstructs only the bandlimited scene, not the original. The

surfaces that appear in volume rendered images are therefore renditions of fuzzy surfaces present in the bandlimited scene, not anti-aliased renditions of surfaces present in the original scene.

Within the context of the present algorithm, blurring and supersampling are useful tools for improving surface renditions. If the array of acquired values are blurred slightly during data preparation, the oversharp surface silhouettes occasionally exhibited by volume renderings are softened. Alternatively, we can apply blurring to the opacities generated by the classification procedure, but leave the shading untouched. This has the effect of softening silhouettes without adversely affecting the crispness of surface detail.

The decision to reduce aliasing at the expense of resolution arises from two conflicting goals: generating artifact-free images and keeping rendering costs low. In practice, the slight loss in image sharpness might not be disadvantageous. Indeed, it is not clear that the accuracy afforded by more expensive visibility calculations is useful, at least for the types of data considered in this study. Blurry silhouettes have less visual impact, but they reflect the true imprecision in our knowledge of surface locations.

An alternative means for improving image quality is supersampling. The basic idea is to interpolate additional samples between the acquired ones prior to compositing. If the interpolation method is a good one, the accuracy of the visibility calculations is improved, reducing some kinds of aliasing. Another option is to apply this interpolation during data preparation. Although this alternative substantially increases computational expense in the remainder of the pipeline, it improves the accuracy of shading and classification as well as visibility calculations.

2.6. Case studies

To illustrate how this algorithm behaves on typical datasets, let us consider several examples. The first is a $113 \times 113 \times 113$ voxel portion of an electron density map for the protein cytochrome B5. Figure 2.7 shows four slices spaced 10 voxels apart in this dataset. Each whitish cloud represents a single atom. Using the shading and classification calculations described in sections 2.2.1 and 2.2.2.1, colors and opacities were computed for each voxel in the expanded dataset. These calculations required 30 seconds on a Sun 4/280. Ray tracing, resampling, and compositing were performed as described in the introduction to section 2.2 and in section 2.2.3 and took 30 seconds, yielding the image in figure 2.8.

Figures 2.9 and 2.10 were generated from a computed tomography (CT) study of a cadaver acquired as 113 slices of 256×256 samples each. Using the shading and classification calculations described in sections 2.2.1 and 2.2.2.2, two sets of colors and opacities were computed, one showing the air-skin interface and a second showing the tissue-bone interface. The computation of each set required 2 minutes. Two views were then computed from each set of colors and opacities, producing four images in all as shown in figure 2.9. The computation of each view required an additional 2 minutes. The horizontal bands through the patient's teeth in these images are artifacts due to scattering of X-rays from dental fillings and are present in the acquired data. The bands across her forehead and under her chin in the air-skin images are gauze bandages used to immobilize her head during scanning. Her skin and nose cartilage are rendered semi-transparently over the bone surface in the tissue-bone images.

Figure 2.10 was generated by combining halves from each of the two sets of colors and opacities already computed for figure 2.9. Heightened transparency of the temporal bone and the bones surrounding the maxillary sinuses - more evident in moving sequences than in a static view - is due to generalized osteoporosis. It is worth noting that rendering techniques employing binary classification decisions would likely display holes here instead of thin, wispy surfaces.

The dataset used in figures 2.11 and 2.12 is of the same cadaver, but was acquired as 113 slices of 512×512 samples each. Figure 2.11 was generated using the same procedure as for figure 2.9, but casting four rays per slice in the vertical direction in order to correct for the aspect ratio of the dataset. Figure 2.12 was generated by expanding the dataset to 452 slices using a

cubic B-spline in the vertical direction, then generating an image from the larger dataset by casting one ray per slice. As expected, more detail is apparent in figure 2.12 than figure 2.11.

Figures 2.13 and 2.14 exemplify some of the types of animation sequences discussed in section 2.4. In these two figures, the dataset used in figure 2.9 has been rendered in color to show both bone and soft tissue, and the opacities of all voxels inside a cube-shaped region above the right eye have been scaled down to nearly zero. To avoid aliasing artifacts, the transition from scaled to unscaled opacities has been spread over a distance of several voxels. To further improve the visualization, voxels in the transition zone have been shaded as if the region of interest contained air rather than tissue. The effect of this extra step is to cap off anatomical structures where they enter the region of interest. Figure 2.14 is identical to figure 2.13 except that the region of interest and light source have been moved. More evident in a moving sequence than in still images, moving the light source helps resolve ambiguities in 3D shapes and object relationships.

Features not meeting the adjacency criteria described in section 2.2.2.2 include internal soft tissue organs in CT studies and most structures in MR studies. Figures 2.15 and 2.16 illustrate one possible strategy for rendering these features. The left pair of images in figure 2.15 show a slice and a volume rendering from a $256 \times 256 \times 156$ voxel magnetic resonance (MR) study of a human head. The apparent mottling of the facial surface in the volume rendering is due to noise in the acquired data. In order to display the cortical surface, the overlying tissues were removed by manually erasing selected voxels on each slice. The right pair of images and figure 2.16 show a slice and two volume renderings from the edited dataset. Since the boundary between erased and unerased voxels falls within tissues that are rendered transparently, the boundary is not seen in the volume rendering and need not be specified precisely. In other words, the user is not called upon to define surface geometry, but merely to isolate a region of interest.

2.7. Summary and discussion

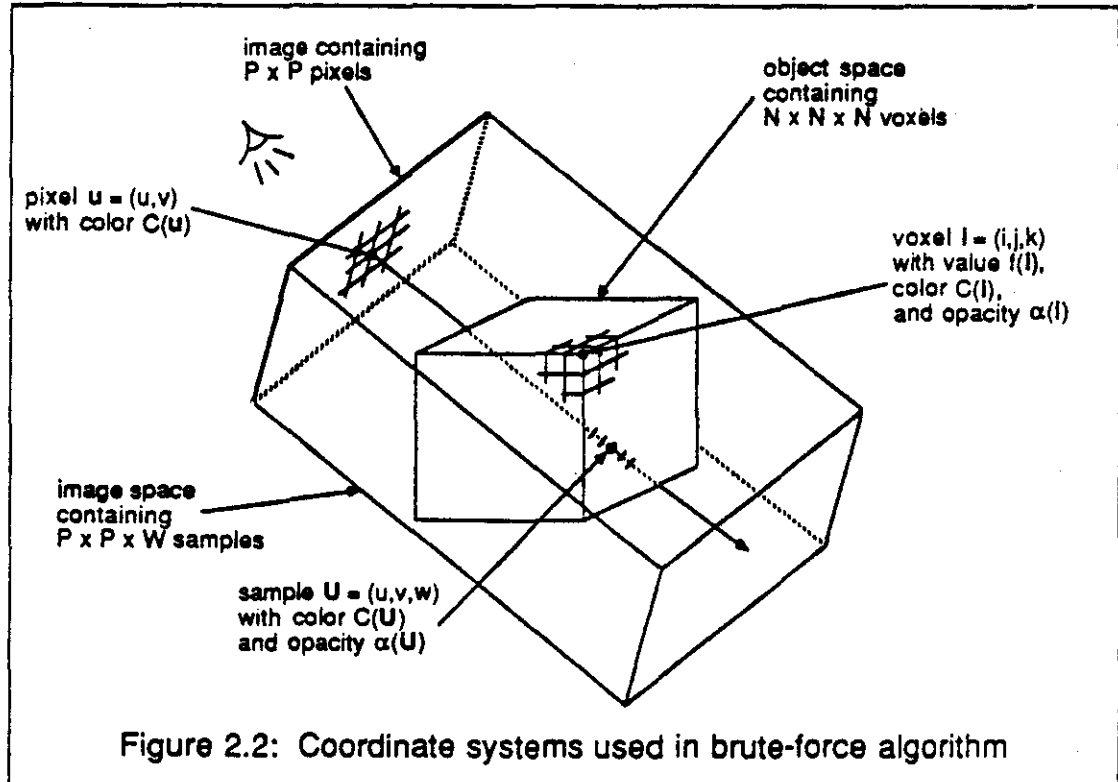
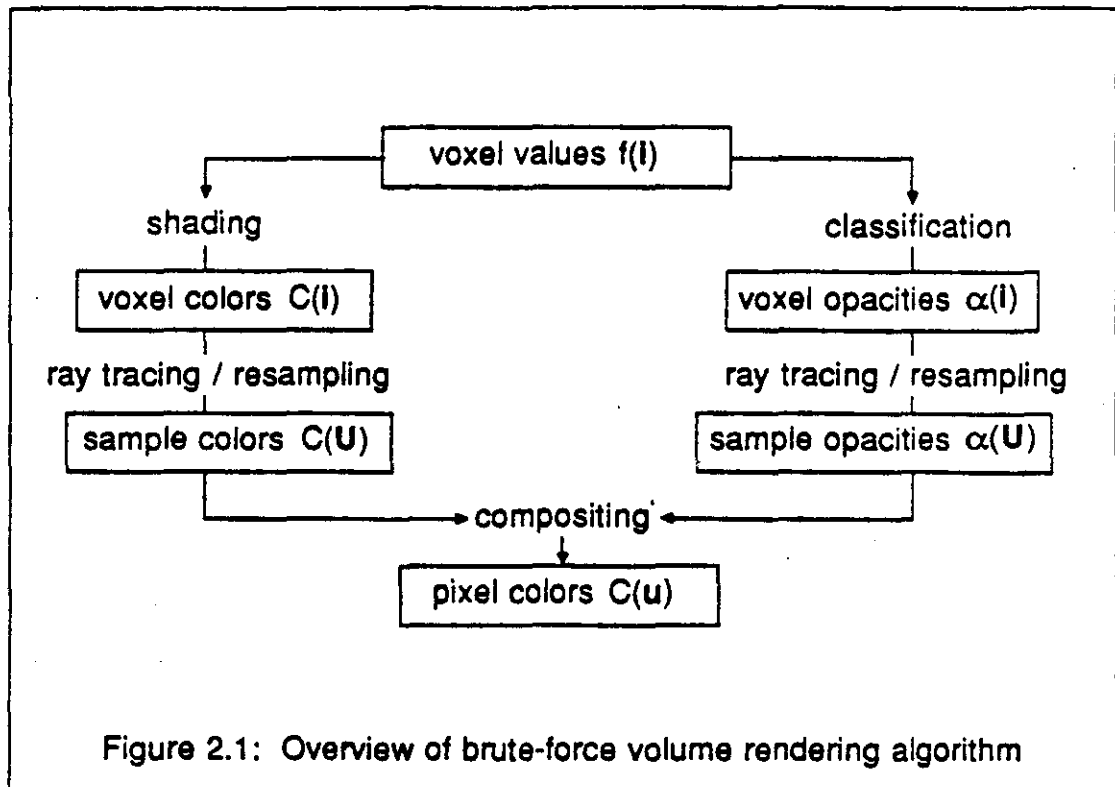
Volume rendering has been shown to be an effective modality for the display of surfaces from sampled scalar fields of three spatial dimensions. As demonstrated by the figures, it can generate images exhibiting approximately equivalent resolution, yet fewer interpretation errors, than techniques relying on geometric primitives or binary voxel representations.

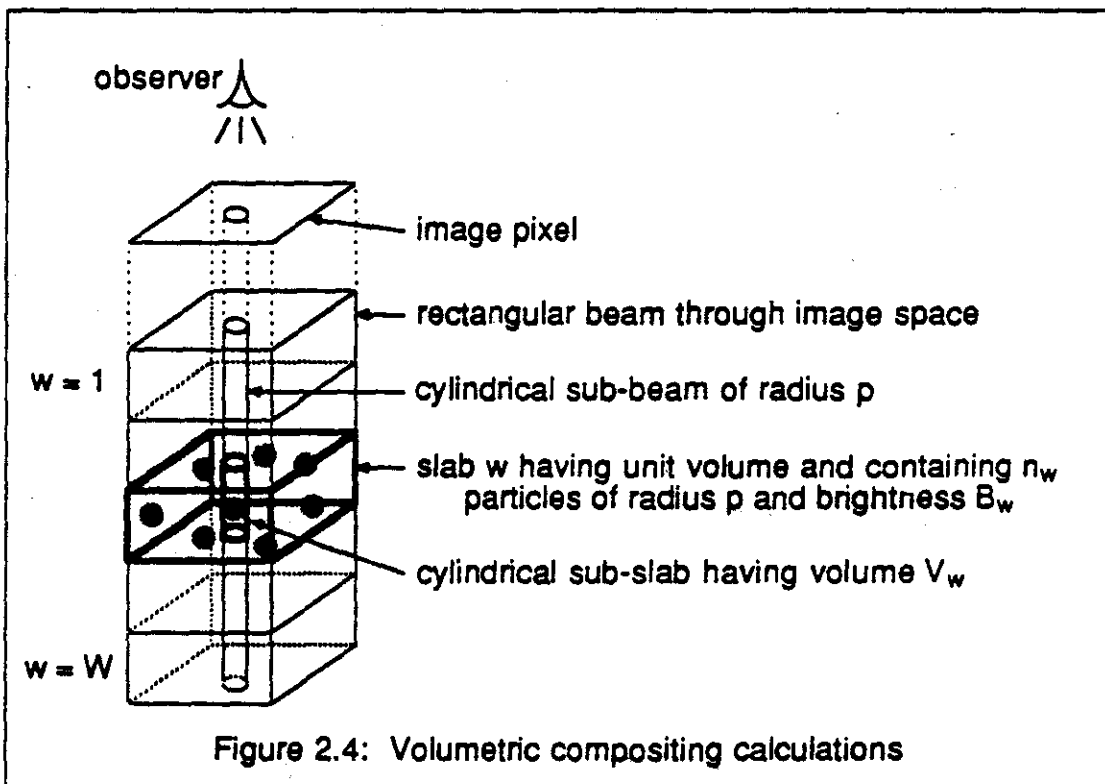
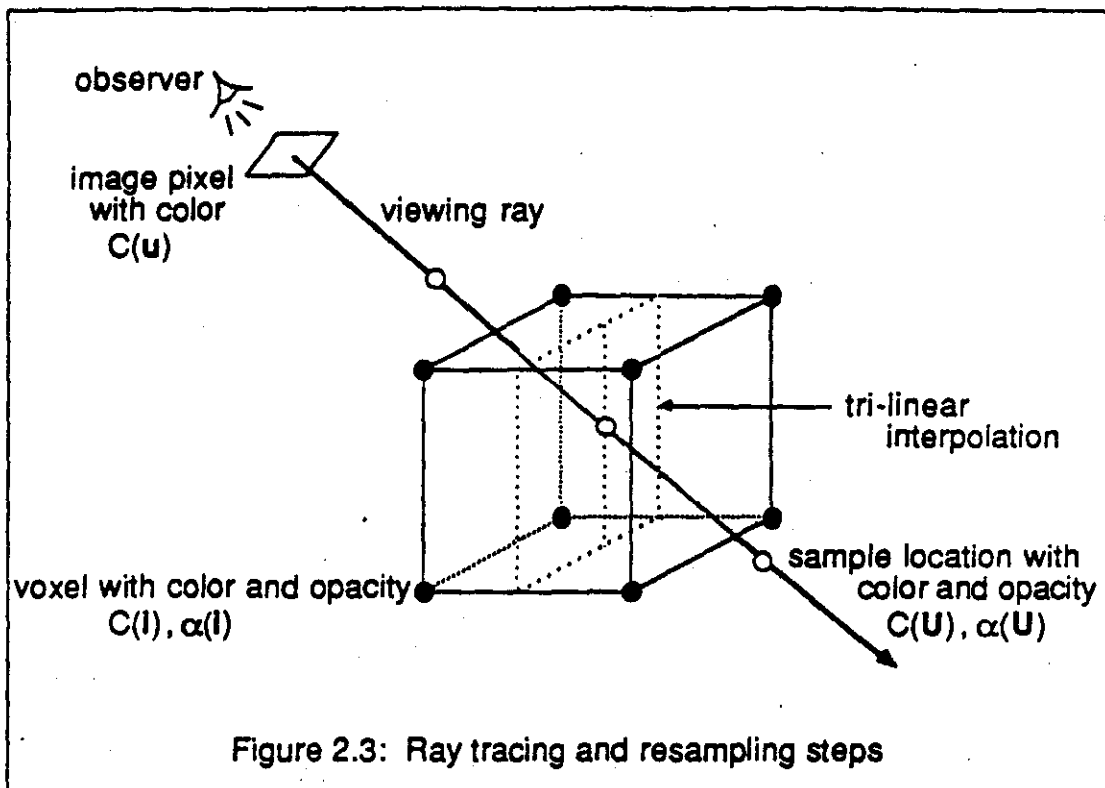
Despite its advantages, volume rendering has several problems. The omission of an intermediate geometric representation makes selection of appropriate shading parameters critical to the effectiveness of the visualization. Slight changes in opacity ramps or interpolation methods radically alter the features that are seen as well as the overall quality of the image. For example, the thickness of the transition region surrounding the isovalue contour surfaces described in section 2.2.2.1 stays constant only if the local gradient magnitude stays constant within a radius of r voxels around each point on the surface. The time and ensemble averaging inherent in X-ray crystallography usually yields suitable data, but there are considerable variations among datasets. Algorithms are needed that automatically select an optimum value for r based on the characteristics of a particular dataset.

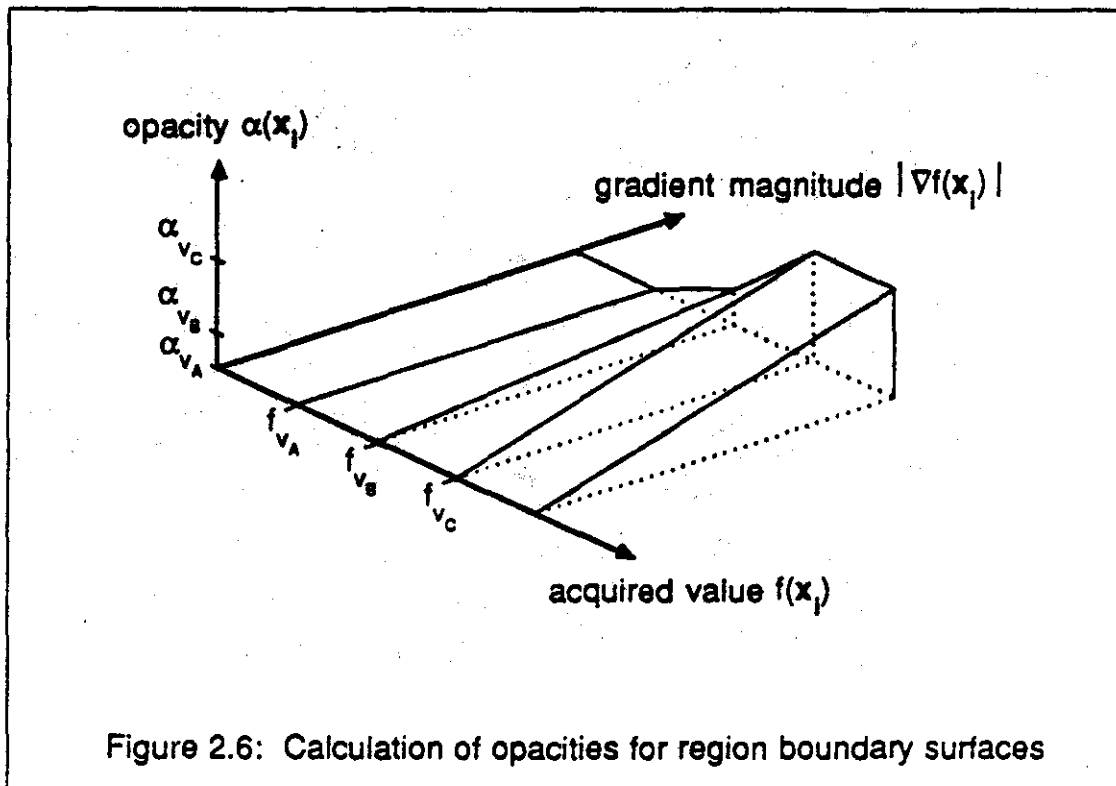
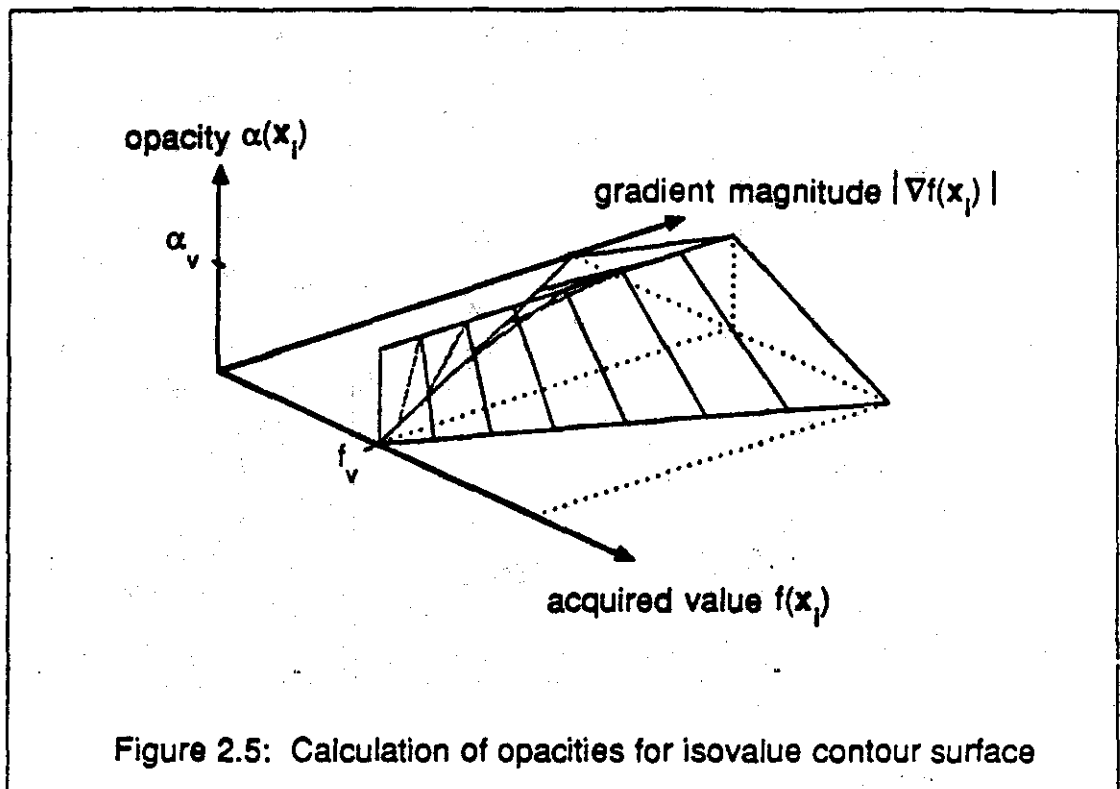
Volume rendering is also very sensitive to artifacts in the acquisition process. For example, CT scanners generally have anisotropic spatial sensitivity. This problem manifests itself as striping in images. With live subjects, patient motion is also a serious problem. Since shading calculations are strongly dependent on the orientation of the local gradient, slight misalignments between adjacent slices produce strong striping.

An alternative solution for features not meeting the adjacency criteria described in section 2.2.2.2 would be to combine volume rendering with high-level object definition methods such as [Gauch88] in an interactive setting. Initial visualizations, made without the benefit of object definition, would be used to guide scene analysis and segmentation algorithms, which would in turn be used to isolate regions of interest, producing a better visualization. If the output of such

segmentation algorithms included confidence levels or probabilities, they could be mapped to opacity and thus modulate the appearance of the image.







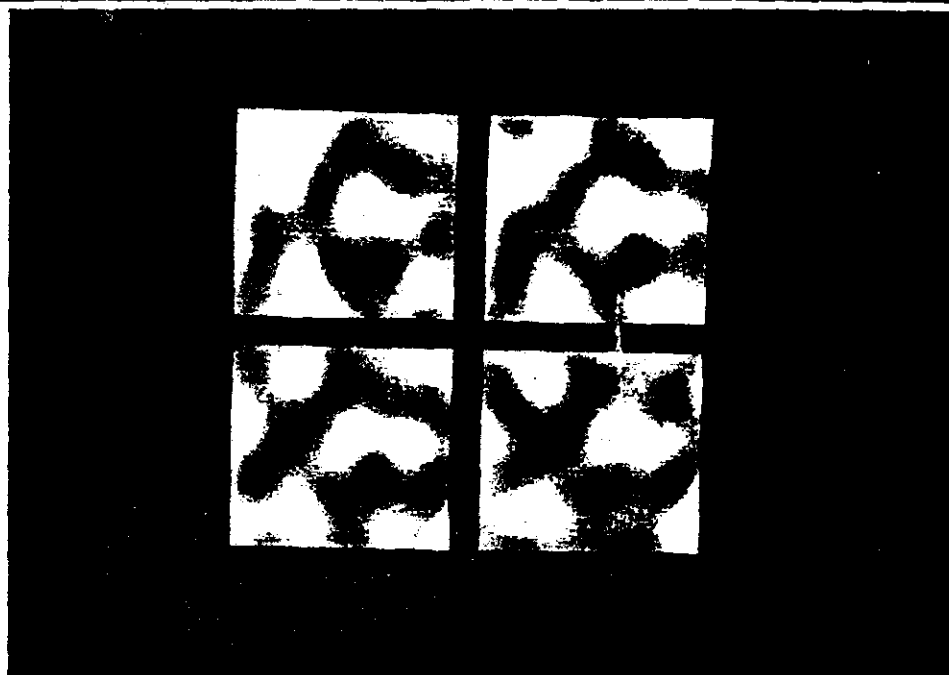


Figure 2.7: Representative slices from 113 x 113 x 113 voxel electron density map of cytochrome B5



Figure 2.8: Volume rendering of isovalue contour surface from dataset shown in figure 2.7

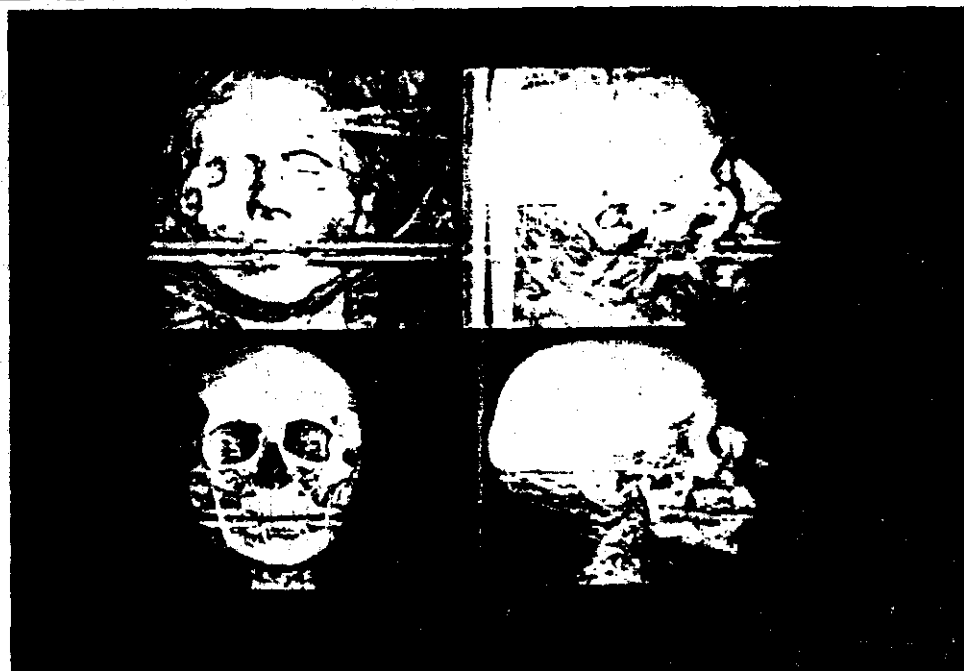


Figure 2.9: Volume renderings of region boundary surfaces from 256 x 256 x 113 voxel CT dataset of human head



Figure 2.10: Rotated view of same dataset



Figure 2.11: Rendering of 512 x 512 x 113 voxel CT dataset



Figure 2.12: Rendering of same dataset after interpolation to 512 x 512 x 452 voxels

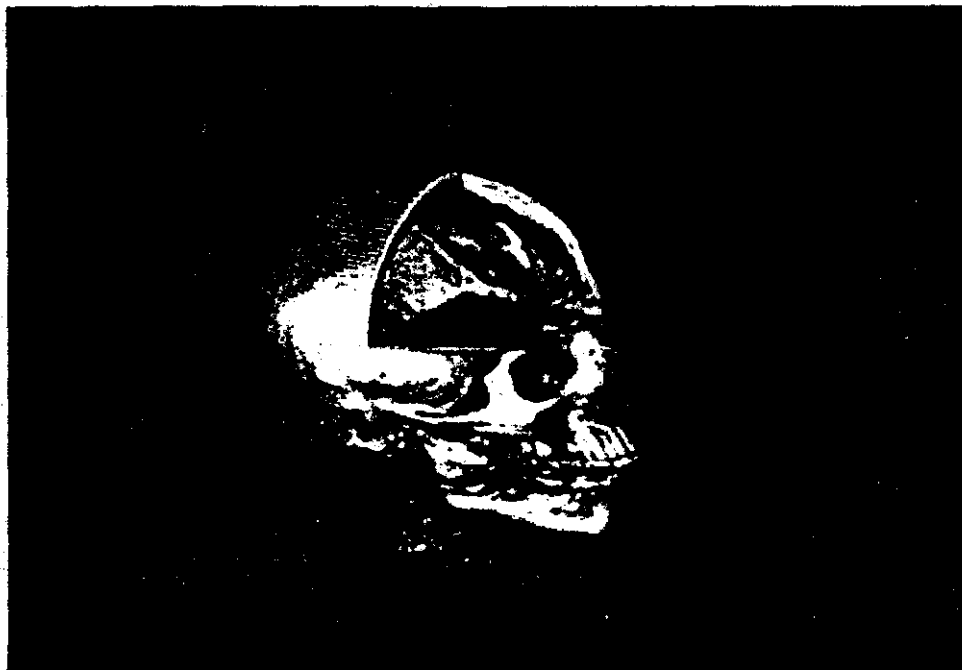


Figure 2.13: Color rendering of CT dataset showing bone, soft tissue, and 3D region of interest formed by scaling down opacity of selected voxels



Figure 2.14: View of same dataset following repositioning of region of interest and light source

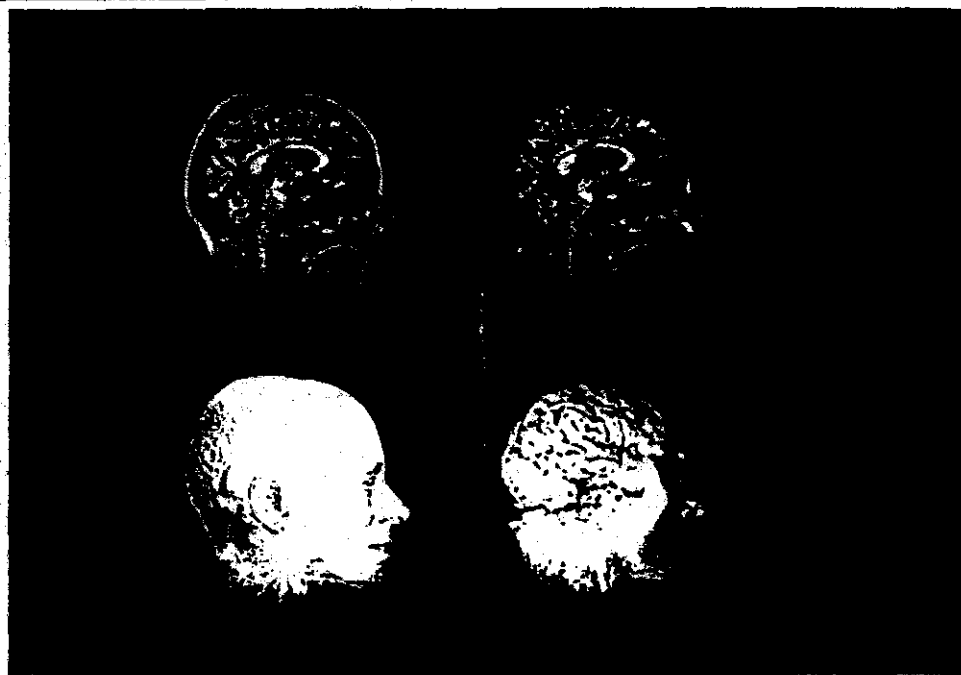


Figure 2.15: Original and edited slices and volume renderings of 256 x 256 x 156 voxel MR dataset of human head



Figure 2.16: Rotated view of edited dataset

CHAPTER III

REDUCING THE COST OF TRACING A RAY

3.1. Background

One of the principal drawbacks of the volume rendering algorithm presented in the previous chapter is its cost. Since all voxels participate in the generation of each image, rendering time grows linearly with the size of the dataset. This chapter presents two techniques for reducing the expense of tracing a ray through volume data.

The first optimization is based on the observation that many datasets contain coherent regions of empty voxels. In the context of volume rendering, a voxel is defined as empty if its opacity is zero. Techniques for encoding coherence in volume data include octree hierarchical spatial enumerations [Meagher82], polygonal representations of bounding surfaces [Fuchs77, Pizer86], and octree representations of bounding surfaces [Gargantini86]. The algorithm presented in this chapter employs an octree enumeration similar to that of Meagher, but represents the enumeration by a pyramid of binary volumes or *complete octree* [Yau83] rather than by a condensed representation. The present algorithm also differs from the work of Meagher in that it renders data in image order, i.e. by tracing viewing rays from an observer position through the octree, while Meagher renders in object order, i.e. by traversing the octree in depth-first manner while following a consistent direction through space.

The second optimization is based on the observation that once a ray has struck an opaque object or has progressed a sufficient distance through a semi-transparent object, opacity accumulates to a level where the color of the ray stabilizes and ray tracing can be terminated. Many algorithms for displaying medical data stop after encountering the first surface or the first opaque voxel. In this guise, the idea has been reported in [Goldwasser86b, Schlusberg86, Trouset87] and perhaps elsewhere. In volume rendering, surfaces are not explicitly detected. Instead, they appear in the image as a natural byproduct of the stepwise accumulation of color and opacity along each ray. Adaptive termination of ray tracing can be added to the present algorithm by stopping each ray when its opacity reaches a user-selected threshold level.

The speedup obtained using these optimizations is highly dependent on the depth complexity of the scene. In this thesis, we focus on visualizations consisting of opaque or semi-transparent surfaces. A plot of opacity along a line perpendicular to one of these surfaces typically exhibits a bump shape several voxels wide, and voxels not in the vicinity of surfaces have an opacity of zero. For these scenes, savings of up to an order of magnitude over the brute-force algorithm described in the previous chapter has been observed. For scenes consisting solely of opaque surfaces, the cost of generating images has been observed to grow nearly linearly with the size of the image rather than linearly with the size of the dataset.

3.2. Two optimization techniques

3.2.1. Hierarchical enumeration of dataset

The first optimization technique we consider is hierarchical spatial enumeration. For a dataset measuring N voxels on a side where $N = 2^M + 1$ for some integer M , we represent this enumeration by a pyramid of $M+1$ binary volumes as shown in figure 3.1 for the case of $N = 5$. Volumes in this pyramid are indexed by a level number m where $m = 0, \dots, M$, and the volume at level m is denoted V_m . Volume V_0 measures $N-1$ cells on a side, volume V_1 measures $(N-1)/2$ cells on a side, and so on up to volume V_M , which is a single cell. Cells are indexed by a level number m and a vector $i = (i,j,k)$ where $i,j,k = 1, \dots, N-1$, and the value contained in cell i on level m is denoted $V_m(i)$. We define the size of cells on level m to be 2^m times the spacing between voxels. Since voxels are treated as points, whereas cells fill the space between voxels, each volume is one cell larger in each direction than the underlying dataset as shown in the figure. We also place voxel $(1,1,1)$ at the front-lower-right corner of cell $(1,1,1)$. Thus, for example, cell $(1,1,1)$ on level zero encloses the space between voxels $(1,1,1)$ and $(2,2,2)$.

We construct the pyramid as follows. Cell i in the base volume V_0 contains a zero if all eight voxels lying at its vertices have opacity equal to zero. Cell i in any volume V_m , $m > 0$, contains a zero if all eight cells on level $m-1$ that form its octants contain zeros. In other words, let $\{1,2,\dots,k\}^n$ be the set of all n -vectors with entries $\{1,2,\dots,k\}$. In particular, $\{1,2,\dots,k\}^3$ is the set of all vectors in 3-space with integer entries between 1 and k . We then define

$$V_0(i) = \begin{cases} 1 & \text{if } \alpha(i+\Delta i) = 1 \text{ for } i \in \{1,2,\dots,N-1\}^3 \text{ and any } \Delta i \in \{0,1\}^3 \\ 0 & \text{otherwise} \end{cases} \quad (3.1a)$$

and

$$V_m(i) = \begin{cases} 1 & \text{if } V_{m-1}(2i-\Delta i) = 1 \text{ for } i \in \{1,2,\dots,(N-1)/(m+1)\}^3 \text{ and any } \Delta i \in \{0,1\}^3 \\ 0 & \text{otherwise} \end{cases} \quad (3.1b)$$

for $m = 1, \dots, M$.

We now reformulate the ray tracing, resampling, and compositing steps of our rendering algorithm to use this pyramidal data structure. For each ray, we first compute the point where the ray enters the single cell at the top level. We then traverse the pyramid in the following manner. When we enter a cell, we test its value. If it contains a zero, we advance along the ray to the next cell on the same level. If the parent of the new cell differs from the parent of the old cell, we move up to the parent of the new cell. We do this because if the parent of the new cell is unoccupied, we can advance the ray further on our next iteration than if we had remained on a lower level. This ability to advance quickly across empty regions of space is where the algorithm saves its time. If, however, the cell being tested contains a one, we move down one level, entering whichever cell encloses our current location. If we are already at the lowest level, we know that one or more of the eight voxels lying at the vertices of the cell have opacity greater than zero. We then draw samples at evenly spaced locations along that portion of the ray falling within the cell, resample the data at these sample locations, and composite the resulting color and opacity into the color and opacity of the ray.

3.2.2. Adaptive termination of ray tracing

The second optimization technique we consider is adaptive termination of ray tracing. Our goal is to quickly identify the last sample location along a ray that significantly changes the color of the ray. Returning to equation (2.11a), we define a significant color change as one in which $C_{\alpha_m}(u;U) - C_{\alpha_n}(u;U) > \epsilon$ for some small $\epsilon > 0$. Since $\alpha_m(u;U)$ increases monotonically along the

3.4. Comparison to ray tracing of geometrically defined scenes

The tracing of rays through coherent regions of empty voxels in volume data is analogous to the tracing of rays through expanses of empty space in geometrically defined scenes. This problem has received much attention in the computer graphics literature (see [Arvo88] for an excellent survey), and it is useful to compare the present algorithm to strategies for speeding up ray tracing of geometric scenes.

One such strategy is to place bounding volumes around primitives or groups of primitives. Rays are tested first against these volumes, and if a volume is hit, then against its contents. Bounding volume schemes that have been tried include spheres [Whitted80], parallelepipeds [Rubin80], extruded extents [Kajiya83], and convex hulls [Kay86]. This technique can be applied to volume data by fitting geometric primitives to the sample array. Primitives that have been used for this purpose include opaque cubes [Herman79], polygonal meshes constructed from 2D contours [Fuchs77, Pizer86], and voxel-sized polygons generated directly from 3D data samples [Lorensen87, Cline88]. The principal drawback of this approach is that fitting of primitives requires making a binary classification of the data, leading to artifacts in the generated images.

An alternative strategy is to subdivide space into disjoint cells and to associate with each cell a list of primitives that fall wholly or partially inside it. Rays are advanced incrementally through the scene, moving from cell to cell. When a ray enters a cell that contains primitives, the ray is tested against those primitives; when a ray enters a cell marked as empty, the ray is simply advanced to the next cell. Variants of this technique include uniform subdivision of space into a regular 3D grid of cubic cells [Fujimoto86], adaptive subdivision into parallelepipeds, generalized cubes, or tetrahedra [Dippe84], and adaptive hierarchical subdivision into cubic cells of varying size using octrees [Glassner84]. These techniques can be applied to a geometric description of the volume data by fitting primitives as described above, or they may be applied directly to the sample array. Specifically, if we treat each data sample as a cell, the resulting regular 3D grid of cells is analogous to uniform subdivision of a geometric scene. Similarly, octree representations of volume data are analogous to adaptive hierarchical spatial subdivisions of geometric scenes.

The analogy between spatial enumeration of volume data and spatial subdivision of a geometric scene is not exact, however, and comparisons made in the literature between competing schemes for subdividing geometric scenes do not scale well when applied to volume data. In particular, spatial subdivisions of geometric scenes typically consist of hundreds of cells each containing many primitives [Cleary88], whereas volume datasets consist of tens of millions of spatially ordered cells each containing a single data sample. Several researchers [Fujimoto86, Amantides87, Cleary88] have reported that, for the geometric scenes they have tested, uniform subdivision outperforms hierarchical subdivision. For the volume datasets considered in this thesis, a hierarchical data structure seems to work better.

3.5. Case studies

To understand how the algorithm behaves on typical scenes, let us consider some examples. The characteristics of three datasets are given in table 3.1. The first is a computed tomography (CT) study of a human skull mounted in a lucite head cast. To demonstrate the effect of semi-transparent surfaces on the performance of the algorithm, this dataset was rendered twice, once with a semi-transparent air-lucite boundary surface (figure 3.3), and once with a completely transparent boundary surface (figure 3.4). The second dataset is a portion of an electron density map of *Staphylococcus Aureus* ribonuclease. A volume rendering of an isovalue contour surface from this map is shown in figure 3.5. The polymer backbone crosses the image from bottom to top, and two Tyrosine residues with their characteristic six-atom benzene rings can be seen extending to the left and right sides of the backbone. To study the growth of rendering cost with respect to dataset size, this dataset was rendered at three different spatial resolutions, the largest of which is shown in the figure. The last dataset is a CT study of a complete human head, a

volume rendering of which is shown in figure 3.6.

For the $256 \times 128 \times 113$ voxel jaw with semi-transparent skin (figure 3.3), calculation of voxel colors and opacities took 1 minute on a Sun 4/280, and calculation of the pyramid of binary volumes took another 30 seconds. The combined costs of ray tracing, resampling, and compositing for all three datasets are summarized in table 3.2. Separate entries are provided for the brute-force algorithm, the optimized algorithm with adaptive termination of ray tracing disabled by setting $\epsilon = 0$, and the fully optimized algorithm with $\epsilon = .05$. As the table shows, hierarchical enumeration reduced rendering time by a factor of between 2.0 and 5.0 for this data, and adaptive termination of ray tracing added another factor of between 1.3 and 2.2. We also observe that adding a semi-transparent surface to the rendering of the skull fragment decreased the amount of time saved, but did not eliminate the savings completely. We finally note that doubling the width of the electron density map increased rendering time by roughly a factor of eight for the brute-force algorithm and five for the optimized algorithm.

To help us interpret these results, the cost of generating figure 3.6 has been broken down into its constituent parts. Using the brute-force rendering algorithm described in chapter 2, the cost of finding all non-empty samples along a ray is proportional to the length of the ray clipped to the boundaries of the dataset. For the observer position used in figure 3.6, a visualization of this cost is shown in figure 3.7a. Brighter pixels represent more work. The image is essentially an X-ray of a cube of uniform density. The cost of resampling and compositing the non-empty samples along a ray is proportional to the number found along the ray. For the dataset under consideration, a visualization of this cost is shown in figure 3.7b. This image is essentially an X-ray of a binary representation of the data. As expected, it is brightest along silhouettes where rays pass through large amounts of bony material. The total cost of rendering figure 3.6 using the brute-force algorithm is a weighted sum of figures 3.7a and 3.7b.

Using hierarchical enumeration, the cost of finding all non-empty samples along a ray is proportional to the number of iterations through the outer loop in the *TraceRay₂* procedure plus the number of tests of level zero cells performed in the *RenderCell₁* procedure. A visualization of this cost is shown in figure 3.8a. This image is essentially an X-ray of an octree. The cost of resampling and compositing the non-empty samples is shown in figure 3.8b. Since hierarchical enumeration does not reduce the number of non-empty samples, figure 3.8b is identical to figure 3.7b. The total cost of rendering figure 3.6 using hierarchical enumeration is a weighted sum of figures 3.8a and 3.8b.

Adaptive termination of ray tracing reduces the number of non-empty samples which must be found. For $\epsilon = .05$, a visualization of the reduced cost is shown in figure 3.9a. In regions where fewer samples are processed, resampling and compositing costs drop as well, as shown in figure 3.9b. The total cost of rendering figure 3.6 using both of the optimization techniques is a weighted sum of figures 3.9a and 3.9b.

3.6. Summary and discussion

Two techniques for reducing the expense of tracing rays through volume data have been described, hierarchical spatial enumeration of the dataset and adaptive termination of ray tracing. Any opacity assignment operator that partitions a volume dataset into coherent regions of opaque and transparent voxels is a candidate for this algorithm. The amount of time saved depends on the depth complexity of the partitioned scene.

A strategy used to speed up ray tracing of geometrically defined scenes that has not been addressed here is to group together rays emanating from similar locations and traveling in similar directions. Specific techniques include the light buffer of [Haines86] and the ray classification algorithm of [Arvo87]. In the present algorithm, an orthographic viewing projection is used, and shadowing, reflection, and refraction are not supported. All rays consequently travel in the same direction. Many volume rendering systems offer a perspective viewing projection, however, and

chapter 5 describes algorithms for casting shadows through volume data. Directional data structures might be useful in these cases. Other ray tracing techniques that might be applicable to volume rendering include generalized rays such as beams [Heckbert84], cones [Amantides84], and pencils [Shinya87], statistical optimizations such as distributed ray tracing [Cook86], and frame-to-frame coherence [Badi88].

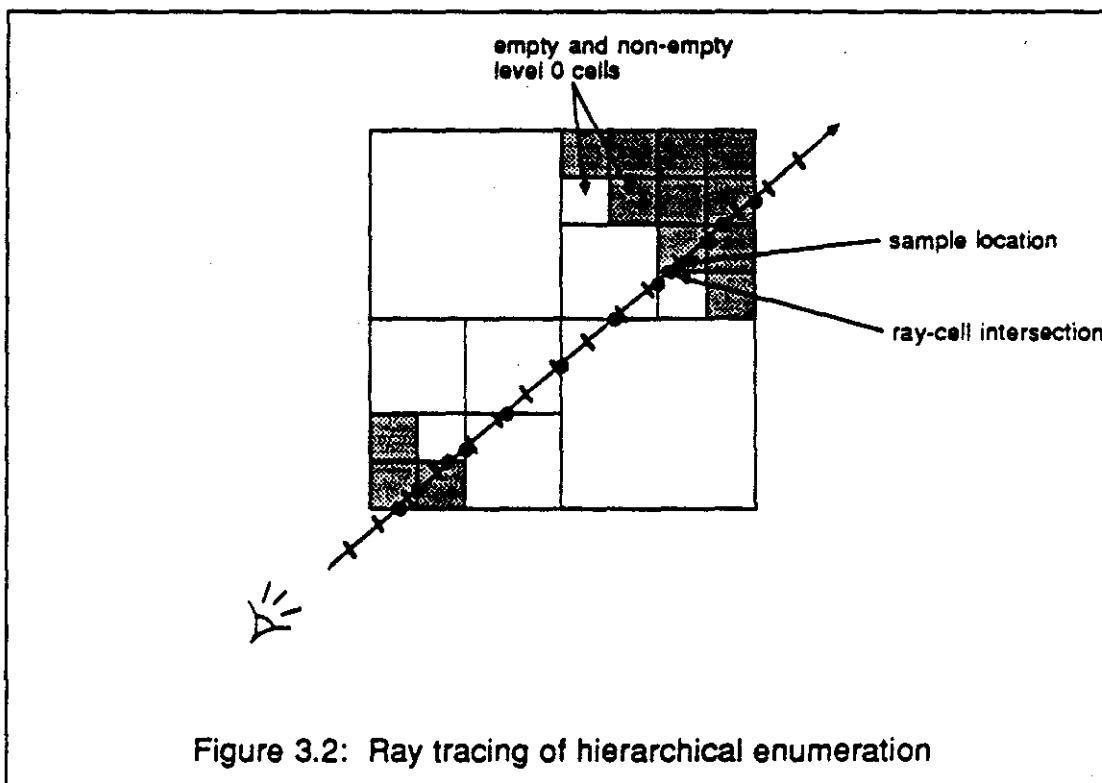
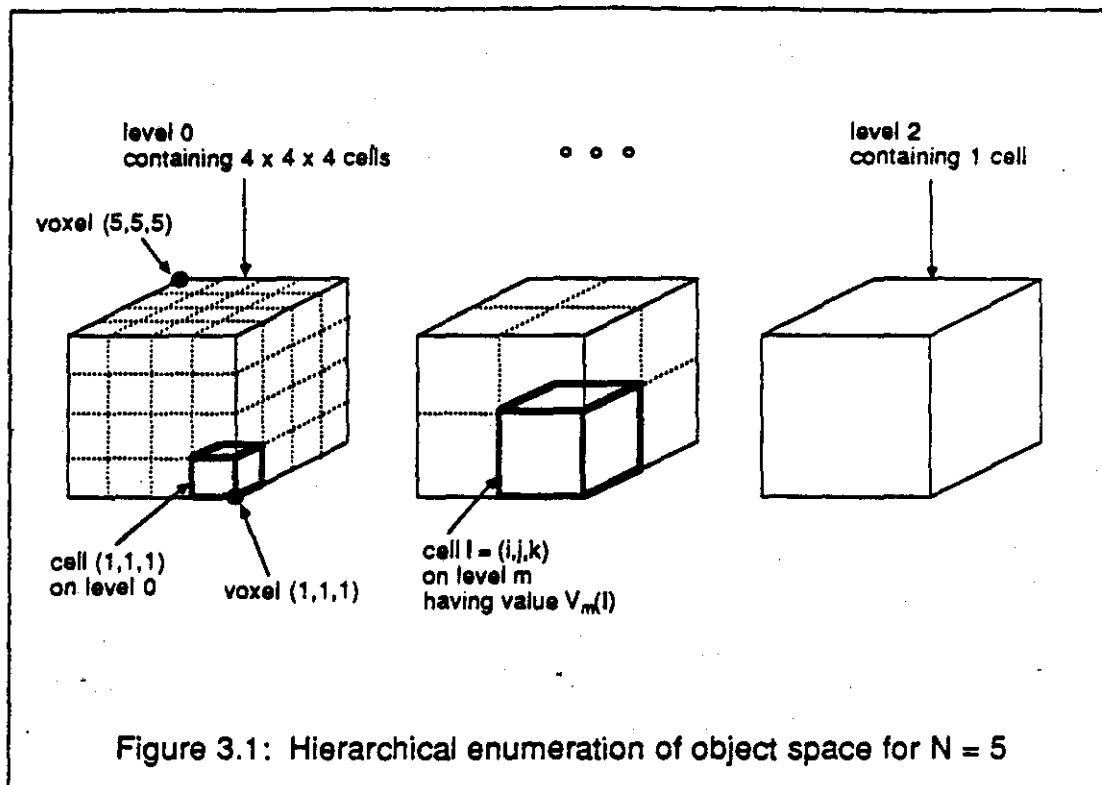




Figure 3.3: Rendering of 256 x 128 x 59 voxel CT dataset of human jaw with lucite skin after interpolation to 256 x 128 x 113 voxels



Figure 3.4: View of same dataset with skin rendered transparently



Figure 3.5: Rendering of 24 x 20 x 11 voxel electron density map of ribonuclease after interpolation to 288 x 244 x 132 voxels

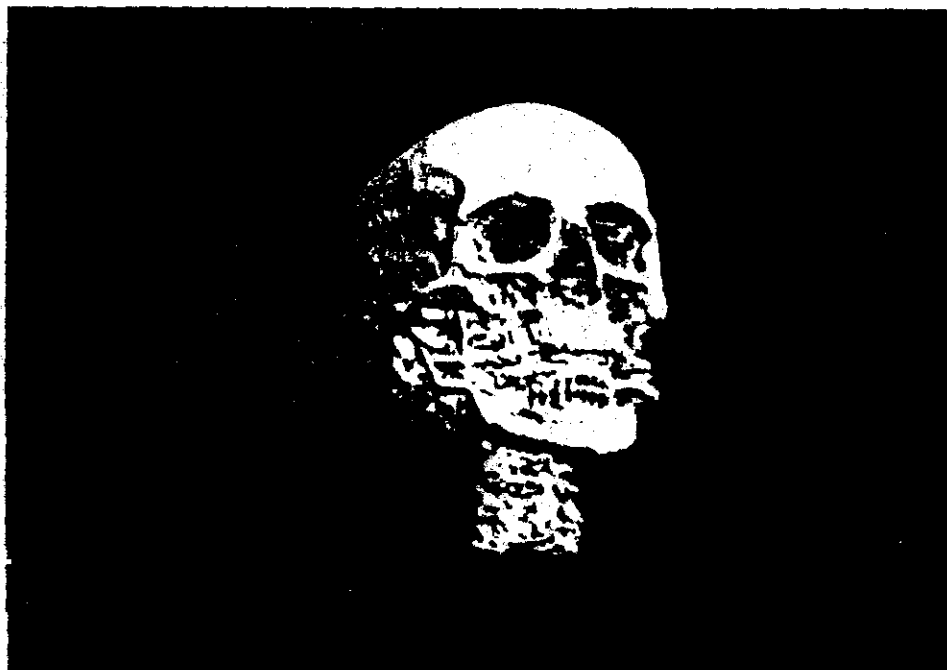
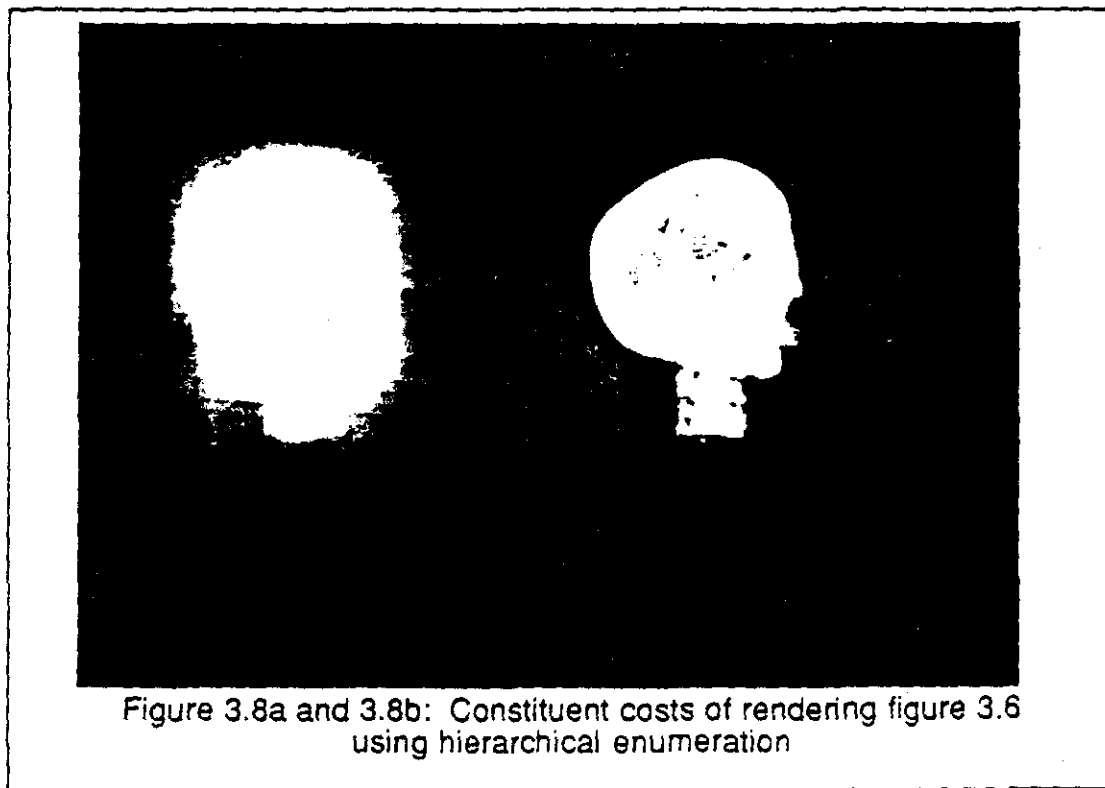
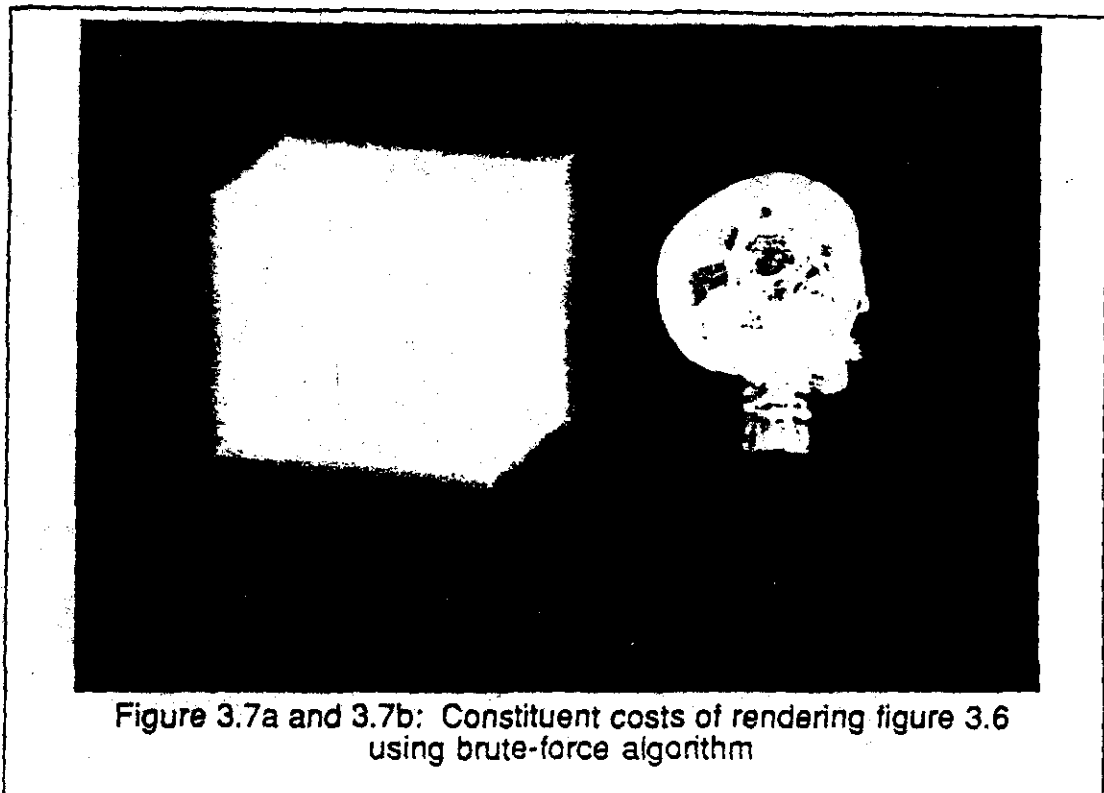


Figure 3.6: Rendering of 256 x 256 x 113 voxel CT dataset of human head after interpolation to 256 x 256 x 226 voxels



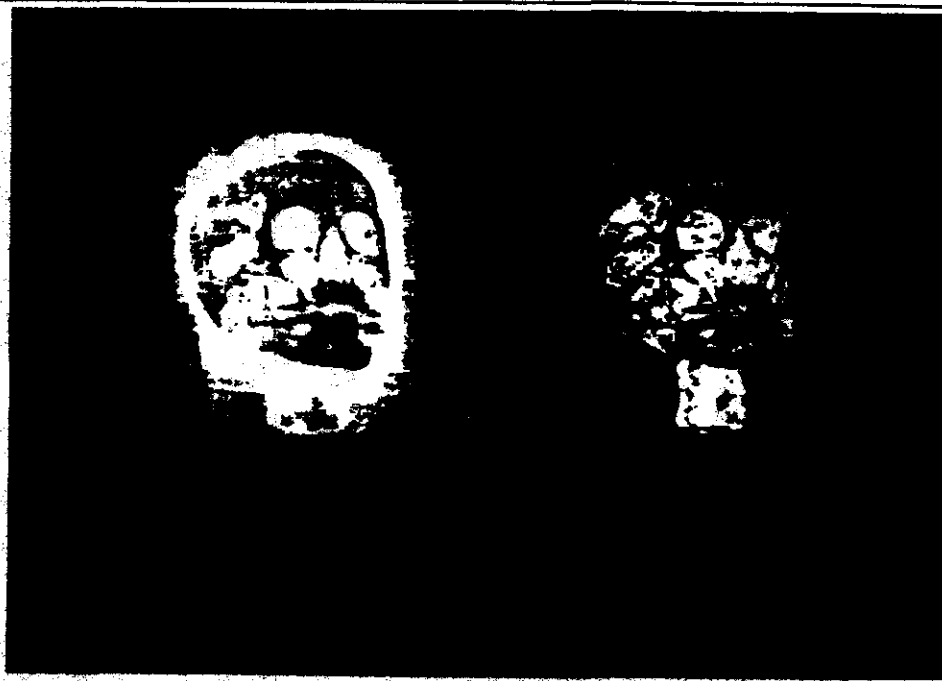


Figure 3.9a and 3.9b: Constituent costs of rendering figure 3.6 using hierarchical enumeration and adaptive termination of ray tracing

name	fig	acquired size	scaling factor	size after scaling	samples drawn	samples with $\alpha > 0$	%
jaw with skin	3.3	256 x 128 x 59	1 x 1 x 2	256 x 128 x 113	3,541,851	594,472	17
jaw w/o skin	3.4	256 x 128 x 59	1 x 1 x 2	256 x 128 x 113	3,541,851	335,751	9
ribonuclease	3.5	24 x 20 x 11	12 x 12 x 12	288 x 244 x 132	7,067,842	810,542	11
ribonuclease	-	24 x 20 x 11	6 x 6 x 6	144 x 120 x 66	825,485	160,747	19
ribonuclease	-	24 x 20 x 11	3 x 3 x 3	72 x 60 x 33	92,724	25,531	27
head	3.6	256 x 256 x 113	1 x 1 x 2	256 x 256 x 226	14,081,917	1,249,458	9

Table 3.1: Characteristics of datasets shown in figures 3.3 through 3.6

name	fig	brute-force	hierarchical enumeration	enumeration and adaptive termination	col. 1 / col. 2	col. 2 / col. 3	col. 1 / col. 3
jaw with skin	3.3	293 secs	94 secs	57 secs	3.1	1.6	5.1
jaw w/o skin	3.4	288	61	39	4.7	1.6	7.4
ribonuclease	3.5	571	146	75	3.9	1.9	7.6
ribonuclease	-	68	27	15	2.5	1.8	4.5
ribonuclease	-	8	4	3	2.0	1.9	2.7
head	3.6	1183	238	105	5.0	2.2	11.3

Table 3.2: Rendering times for datasets characterized in table 3.1

CHAPTER IV

REDUCING THE NUMBER OF RAYS TRACED

4.1. Background

This chapter presents a technique for reducing the number of rays that must be traced to render a volume dataset. The technique can also be used to progressively refine image quality over time. A survey of progressive refinement techniques for polygonal environments is given in [Bergman86]. These techniques have three principal characteristics: they distribute work according to where it makes the most difference, they form intermediate images from partial information, and they minimize the amount of work discarded after formation of each image.

Conventional ray tracing algorithms are not well suited for use in a progressive refinement system. Although numerous techniques exist for distributing rays according to local image complexity [Whitted80, Lee85, Dippe85, Cook86, Kajiya86], these techniques cast large numbers of rays per pixel, resulting in long image generation times. They also complete the rendering of one pixel before moving on to the next, precluding the display of images from partial information. These limitations are necessary when rendering analytically defined objects; undersampling of such scenes produces objectionable aliasing artifacts.

Volume data, unlike analytically defined objects, is assumed to be bandlimited to the Nyquist frequency prior to sampling. The transfer function of many acquisition processes fall off well below the Nyquist frequency as in the case of computed tomography (CT) and electron density maps obtained from X-ray diffraction data [Herman80, Glusker85]. Certain amorphous phenomena occurring in astronomy and physics are even more bandlimited [Upson86]. This property allows subjectively acceptable images to be generated from partial information and therefore images to be displayed as they are being refined.

In the present algorithm, an initial image is generated by casting a uniform but sparse grid of rays into the volume data, interpolating between the resulting colors, and resampling at the display resolution. A sampling rate of one ray per four pixels, corresponding to a data sampling rate of one ray per four voxels, is typical. Subsequent images are generated by discarding interpolated colors, casting more rays, and repeating the interpolation and resampling steps. Recursive subdivision based on color differences is used to concentrate these additional rays in regions of high image complexity, and recursive bi-linear interpolation is used to form images from the resulting non-uniform array of colors. The approach is similar to that described by Whitted [Whitted80], but it is extended to allow sampling rates of less than one ray per pixel and modified to provide a mechanism for progressive refinement.

The cost of computing each image in a refinement sequence is equal to the sum of the costs of recursive subdivision, ray tracing, and recursive interpolation. Of these, only ray tracing would be required if intermediate images were not displayed. In the current implementation, this cost dominates the others by two orders of magnitude. Nearly all the work expended generating intermediate images therefore also contributes toward generating the final image.

Using this algorithm, crude images of many datasets can be obtained in few seconds. Gradually better images are obtained at intervals of a few seconds each, culminating in a high

quality image in less than a minute.

4.2. Adaptive volume rendering algorithm

Figure 4.1 outlines the adaptive rendering algorithm. It begins as in previous chapters with a 3D array of scalar values which is shaded and classified to yield a color and an opacity for each voxel. Parallel viewing rays are traced into the array from an observer position as before, but this time the image plane is divided into square sample regions measuring ω_{\max} pixels on a side, and rays are cast only from the four corner pixels of each region as shown in figure 4.2.

The colors returned by these four rays are then used to estimate local image complexity. Methods that have been used to measure this complexity include color differences [Whitted80] and statistical variance [Lee85, Kajiya86]. Since a sample size of four is too low to justify statistical methods, color differences are used in the present algorithm. If the range of colors returned by the four rays in a sample region is less than some ϵ , no further processing is performed on the region. Otherwise, the region is divided into four subregions and more rays are cast. Subdivision continues until the range of colors falls below ϵ or the size of the region reaches some ω_{\min} where $\omega_{\min} \leq \omega_{\max}$. If $\omega_{\min} < 1$, the image is effectively supersampled, but due to the bandlimited nature of the incoming data and the limited accuracy of operators employed in the present rendering algorithm, such supersampling has not been found to be useful.

When all sample regions have been processed, an image is formed by interpolating between the available colors and resampling. To insure continuity despite the non-uniform distribution of colors, a recursive technique similar to the algorithm employed during ray tracing is used. The image plane is again divided into square regions measuring ω_{\max} pixels on a side. Pixels are interpolated at the midpoints of the four sides and at the center of each region. The region is then divided into four subregions and the process is repeated. Subdivision continues until the region contains a single pixel.

When all pixels have been filled in, the resulting image is displayed. To continue the refinement process, the image is cleared of all interpolated colors, the level of detail is raised by decreasing ω_{\min} , ω_{\max} , or ϵ , the image plane is again divided into sample regions, and ray tracing begins anew. The refinement process alternates between casting rays and forming images, terminating when $\omega_{\max} = 1$, when the user changes a rendering parameter, or when the observer moves.

4.3. Implementation details

Pseudo-code replacing selected procedures from sections 2.3 and 3.3 and incorporating the adaptive rendering algorithm follows:

```

procedure RenderVolume2( ) begin
    {Compute color and opacity for each voxel in dataset}
    for all i in Dataset do begin
        ComputeOpacity(i);
        if  $\alpha(i) > 0$  then
            ComputeColor(i);
    end
end

```

```

{Initialize level-of-detail parameters and flag arrays}
 $\omega_{\max} := First\omega_{\max}()$ ;  $\omega_{\min} := First\omega_{\min}()$ ;  $\epsilon := First\epsilon()$ ;
for all  $u$  in Image do begin
     $F(u) := 0$ ;  $G(u) := 0$ ;
end

{Loop until image is fully refined}
while  $\omega_{\max} > 1$  do begin
    {Divide image into sample regions and cast some (more) rays}
    for  $u := \{1, \omega_{\max}, \dots, P\}^2$  do
         $RayTraceRegion_1(u, \omega_{\max}, \omega_{\min}, \epsilon)$ ;
    {Redivide image into regions and interpolate any missing pixels}
    for  $u := \{1, \omega_{\max}, \dots, P\}^2$  do
         $InterpolateRegion_1(u, \omega_{\max})$ ;
    {Display image, then clear all interpolated colors}
     $DisplayImage_1()$ ;
    for all  $u$  in Image do
        if not  $F(u)$  then  $G(u) := 0$ ;
    {Increment level-of-detail parameters}
     $\omega_{\max} := Next\omega_{\max}()$ ;  $\omega_{\min} := Next\omega_{\min}()$ ;  $\epsilon := Next\epsilon()$ ;
end
end RenderVolume_2

procedure  $RayTraceRegion_1(u, \omega, \omega_{\min}, \epsilon)$  begin
    {Cast rays from four corners of region}
    for  $v := \{0, \omega\}^2$  do
        if  $u+v$  in Image and not  $F(u+v)$  then begin
             $TraceRay_2(u+v)$ ;  $F(u+v) := 1$ ;
        end
    {If region is larger than  $\omega_{\min}$  by  $\omega_{\min}$  pixels and color difference > threshold,}
    {divide into four subregions and continue ray tracing}
    if  $\omega > \omega_{\min}$  and  $Difference(u, \omega, \epsilon)$  then
        for  $v := \{0, \omega/2\}^2$  do
             $RayTraceRegion_1(u+v, \omega/2, \omega_{\min}, \epsilon)$ ;
end  $RayTraceRegion_1$ 

```

```

procedure InterpolateRegion1(u, $\omega$ ) begin
  (Interpolate colors at midpoints of sides and at center of region)
  for v := ((0, $\omega/2$ ),( $\omega$ , $\omega/2$ ),( $\omega/2$ ,0),( $\omega/2$ , $\omega$ ),( $\omega/2$ , $\omega/2$ )) do
    if u+v in Image and not F(u+v) and not G(u+v) then begin
      Interpolate(u, $\omega$ ,u+v); G(u+v) := 1;
    end
  (If region is larger than 2 × 2 pixels.)
  (divide into four subregions and continue interpolation)
  if  $\omega/2 > 1$  then
    for v := (0, $\omega/2$ )2 do
      InterpolateRegion1(u+v, $\omega/2$ );
end InterpolateRegion1.

```

The *First* and *Next* procedures respectively initialize and increment the level-of-detail parameters according to some user-selected sequence. The syntax for $u := (n_1, n_2, \dots, n_k)^2$ do *statement* is adopted from the notation in section 3.2.1 and means perform *statement* exactly k^2 times with the 2-vector $u = (u, v)$ equal to successive members of the set of 2-space vectors with integer entries in the specified list. The syntax for $v := ((u_1, v_1), (u_2, v_2), \dots, (u_k, v_k))$ do *statement* means perform *statement* exactly k times with the 2-vector $v = (u, v)$ equal to successive vectors from the specified list.

The *Difference* procedure decides if, for the pixels at the four corners of a square sample region specified by its lower-left corner and width, the range of intensities for any color component (red, green, or blue) exceeds some threshold. The *Interpolate* procedure computes from the pixels at the four corners of a sample region a color for the specified pixel using linear interpolation (in the case of region boundary midpoints) or bi-linear interpolation (in the case of region centers) and loads it into the image array.

Since the colors associated with traced rays are retained throughout the refinement process, some means is needed to avoid tracing the same rays repeatedly. In the current algorithm, this is implemented by maintaining a flag *F* for each pixel. Rays are traced only from pixels whose flags are clear. Once a ray has been traced from a given pixel, its flag is set. It is worth noting that the retention of colors has the useful side effect of allowing rays to be shared by adjacent sample regions. This reduces the number of rays that must be traced, thereby improving the efficiency of the algorithm.

Some means is also needed to distinguish pixels whose colors are interpolated from pixels whose colors are computed by ray tracing. This is implemented in the current algorithm by maintaining another flag *G* for each pixel. Colors are interpolated only at pixels whose *F* and *G* flags are both clear. Once a pixel has been interpolated, its *G* flag is set. After display of each intermediate image, all *G* flags are cleared before additional rays are cast.

4.4. Case studies

To demonstrate the performance of this algorithm, two case studies are presented. The first is a $123 \times 123 \times 123$ voxel portion of an electron density map of cytochrome B5. Using methods described in section 2.2.2.1, an isovalue contour surface was selected for display. Using the algorithm described above, a four-frame progressive refinement sequence was then generated. The resulting images are shown in figure 4.3 with the sequence running from top-left to bottom-right. Each image measures 256×256 pixels.

Performance statistics for this sequence are given in table 4.1. For each image, the table gives values for the three level-of-detail parameters, incremental and total ray counts, and incremental and total elapsed times. Timings are for a Sun 4/280.

A visualization of the F array showing where rays were cast is given in figure 4.4. Each white pixel in this figure corresponds to a single ray. Thus, any pixel in figure 4.3 whose corresponding pixel in figure 4.4 is white was computed by ray tracing, whereas any pixel in figure 4.3 whose figure 4.4 pixel is black was computed by interpolation. As expected, ray densities are highest along surface silhouettes where color differences are highest, and the overall number of rays increases from the first to the last frame of the sequence. Since $\omega_{\max} = 1$ in the last frame, rays are cast from every pixel, and the F array is completely white.

The second dataset is a computed tomography (CT) study of a human head and was acquired as 113 slices of 256×256 samples each. Using methods described in section 2.2.2.2, the bone surface was selected for display. A four-frame progressive refinement sequence was then generated, the first frame of which is shown in figure 4.5 and the last frame of which is shown in figure 4.6. Each image measures 512×512 pixels. Performance statistics for this sequence are summarized in table 4.2.

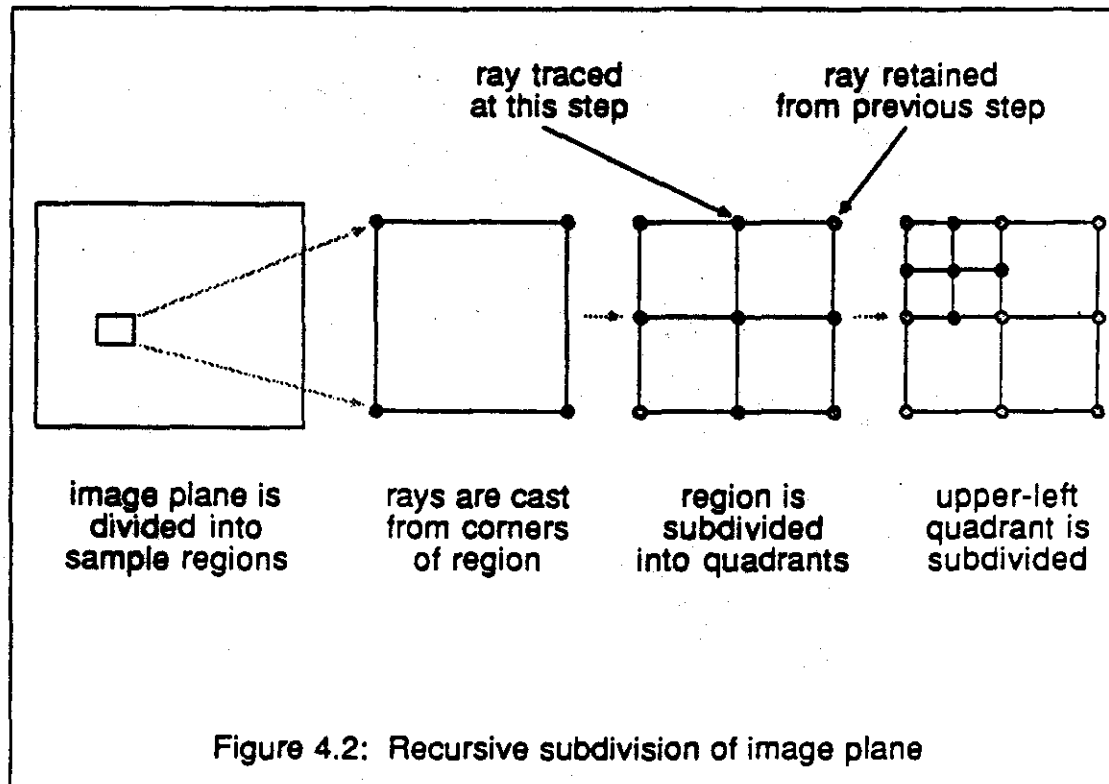
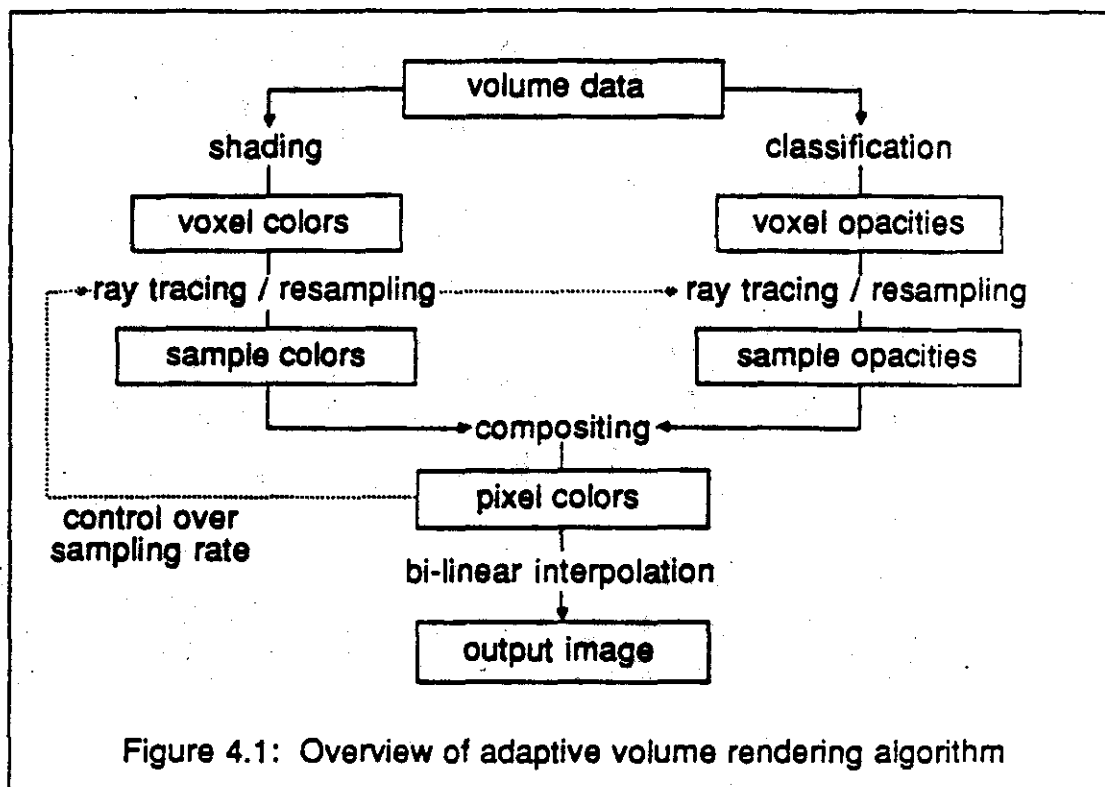
The last dataset is a $256 \times 256 \times 156$ voxel magnetic resonance (MR) study of a human head, edited to remove tissues overlying the cortical surface as described in section 2.6. The first 512×512 pixel image in a four-frame sequence is shown in figure 4.7 and the last frame is shown in figure 4.8. Performance statistics are summarized in table 4.3.

As the tables show, elapsed time is nearly proportional to the number of rays cast. This holds for all datasets and all levels of detail. On the other hand, the effect of changing a particular level-of-detail parameter on the number of rays cast varies considerably between datasets.

4.5. Summary and discussion

A technique for reducing the number of rays required to render a volume dataset has been described. The algorithm casts fewer rays than conventional ray tracers, reflecting the bandlimited nature of volume data, and casts them in an order that allows display of intermediate images.

In this study, values for the three level-of-detail parameters ω_{\max} , ω_{\min} , and ϵ were selected manually. Generally speaking, high values of ω_{\max} cause features to be missed, high values of ϵ cause features to be ignored even if they are not missed, and high values of ω_{\min} causes features to be poorly resolved even if they are neither missed nor ignored. Inappropriate values for these parameters cause suboptimal presentation of the data as well as unequal intervals between successive frames in refinement sequences. Algorithms are needed that automatically select an optimum sequence of values based on the characteristics of a particular dataset.



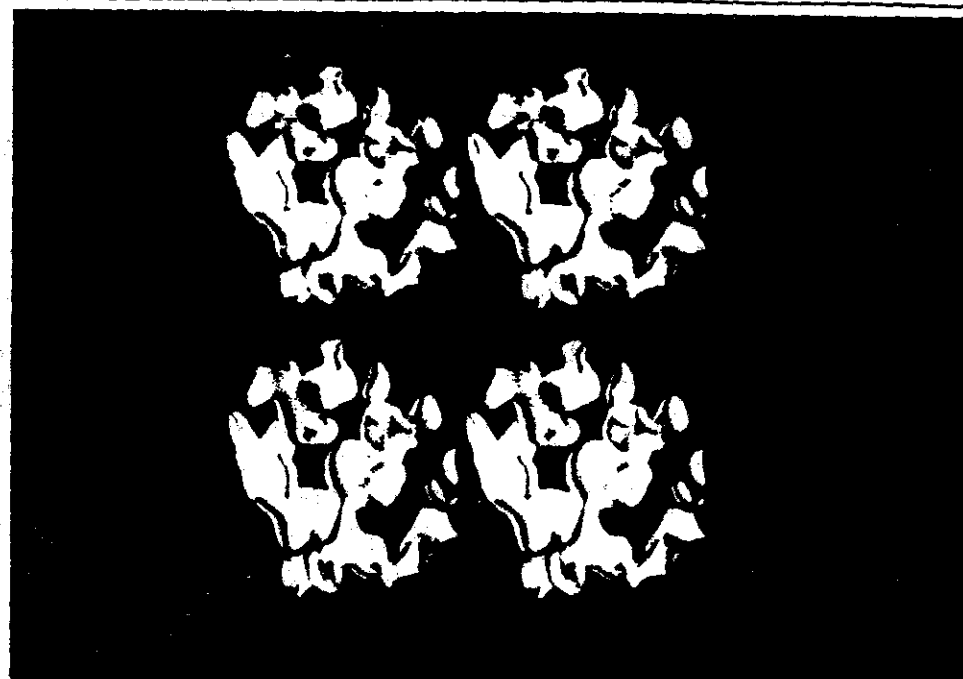


Figure 4.3: Adaptive rendering of electron density map, state of image after 5, 12, 17, and 26 seconds of computation

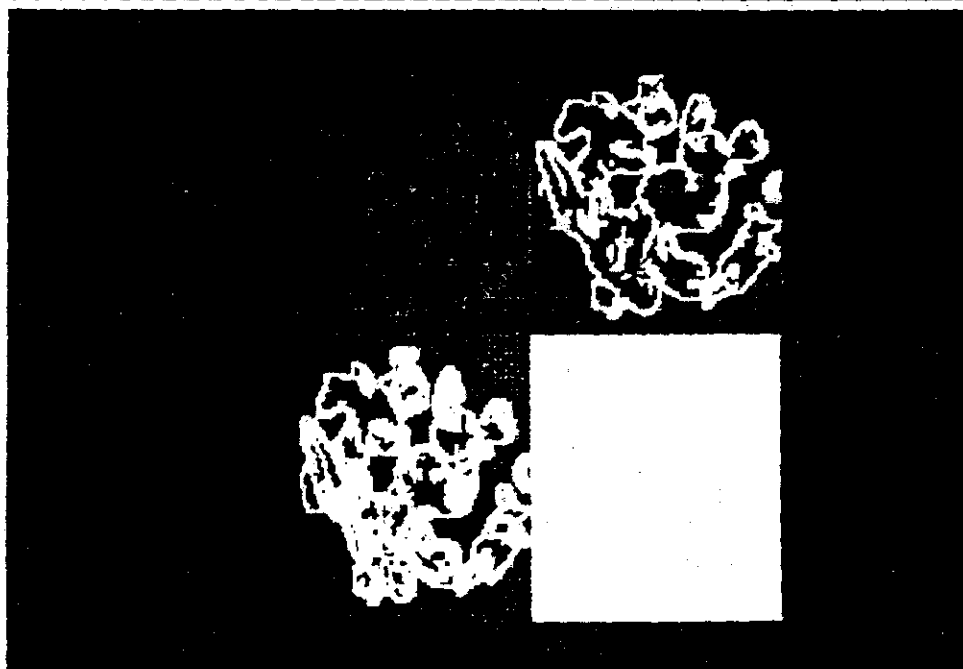


Figure 4.4: Visualization of where rays were cast to generate figure 4.3

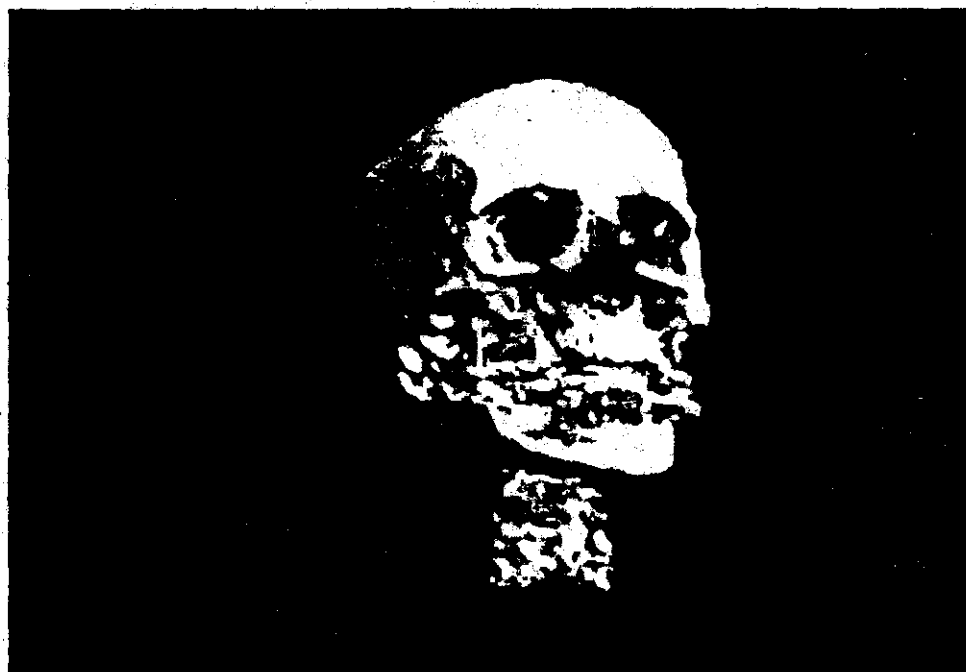


Figure 4.5: Adaptive rendering of CT dataset,
state of image after 13 seconds of computation



Figure 4.6: Continuation of same rendering,
state of image after 104 seconds of computation

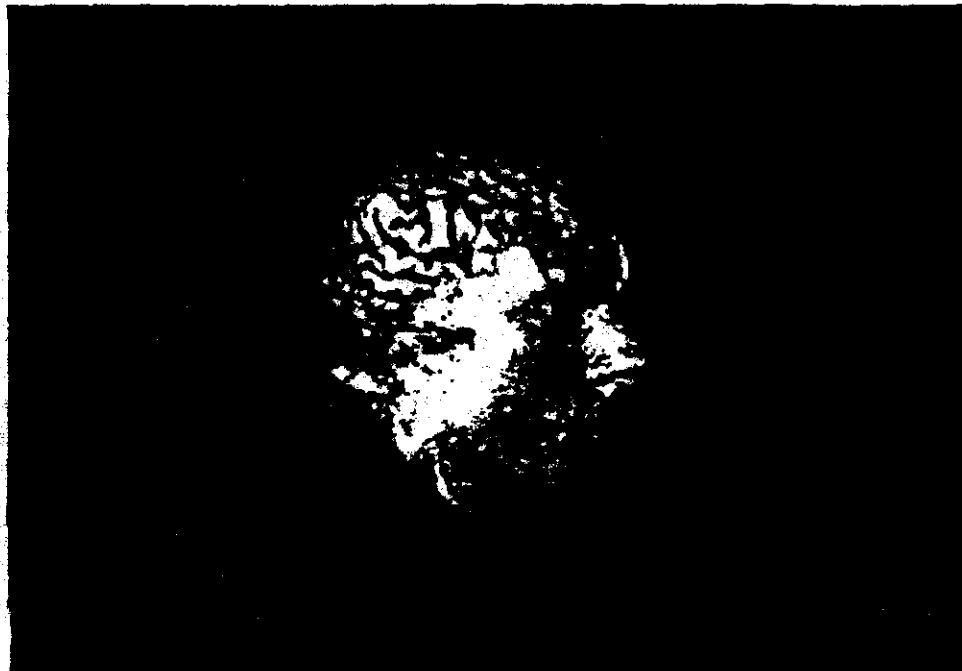


Figure 4.7: Adaptive rendering of MR dataset,
state of image after 18 seconds of computation



Figure 4.8: Continuation of same rendering,
state of image after 120 seconds of computation

frame	image fig	rays fig	total rays	additional rays	%	total time	time interval	ω_{max}	ω_{min}	ϵ
1	4.3a	4.4a	3,870	3,870	20	4.6 secs	4.6 secs	16	2	32
2	4.3b	4.4b	9,947	6,077	51	12.1	7.5	8	1	32
3	4.3c	4.4c	14,459	4,512	74	17.1	5.0	4	1	16
4	4.3d	4.4d	19,289	4,830	100	25.7	8.6	1	1	-

Table 4.1: Performance statistics for adaptive rendering of electron density map

frame	image fig	rays fig	total rays	additional rays	%	total time	time interval	ω_{max}	ω_{min}	ϵ
1	4.5	-	5,179	5,179	16	13 secs	13 secs	16	2	16
2	-	-	16,136	10,957	52	41	28	8	1	16
3	-	-	21,625	5,489	70	63	22	2	1	12
4	4.6	-	30,603	8,978	100	104	41	1	1	-

Table 4.2: Performance statistics for adaptive rendering of CT dataset

frame	image fig	rays fig	total rays	additional rays	%	total time	time interval	ω_{max}	ω_{min}	ϵ
1	4.7	-	5,850	5,850	19	18 secs	18 secs	16	2	16
2	-	-	18,538	12,688	59	57	39	16	1	16
3	-	-	26,841	8,303	85	83	26	4	1	8
4	4.8	-	31,535	4,694	100	120	17	1	1	-

Table 4.3: Performance statistics for adaptive rendering of MR dataset

CHAPTER V

RENDERING MIXTURES OF GEOMETRIC AND VOLUME DATA

5.1. Background

Many scientific problems require sampled functions and analytically defined objects to appear together in a single visualization. This chapter addresses the problem of extending volume rendering to handle such objects.

Most previous efforts in this area have used polygonal meshes or binary voxel representations to display both the geometric and volume data. Fuchs et al. [Fuchs77] and Pizer et al. [Pizer86] track edges on each slice of a computed tomography (CT) dataset to yield a set of contours, tile between contours on adjacent slices, supplement the resulting polygonal mesh with analytically defined objects, and render the ensemble using conventional hidden-surface algorithms. Lørsensen and Cline [Lørsensen87, Cline88] apply surface detectors at each sample location to produce a large collection of voxel-sized polygons which can be supplemented with analytically defined objects and rendered using conventional algorithms. Kaufman (with others) thresholds volume data to produce a binary voxel representation, use 3D scan-conversion to add polygons [Kaufman87b], polyhedra [Kaufman86b], and cubic parametric curves, surfaces, and volumes [Kaufman87a] to the array, and render the resulting ensemble using custom-designed hardware [Kaufman88a]. All of these approaches require a binary classification of the volume data, leading to artifacts as described in chapter 2. Kaufman's use of a binary voxel representation for analytically defined objects gives rise to additional artifacts, although a solution to this problem has apparently been worked out [Kaufman88b].

Although many researchers have suggested methods for extending volume rendering to handle analytically defined objects, no implementations have yet appeared in the literature. Sun Microsystems has extended a conventional ray tracer to handle volume data [Mosher88]. In a paper currently in review, Goodsell et al. [Goodsell88] describe a multi-pass approach that combines a Z-buffer algorithm for rendering polygonally defined atomic structure, a ray tracer for rendering volume data, and a depth-buffer enhanced image compositor such as described in [Duff85]. Although their method produces satisfactory visualizations in many cases, it cannot efficiently handle semi-transparent polygons. Furthermore, the division of labor into two passes necessitates rendering all polygons even though they might be obscured by volume data.

This chapter presents two methods for rendering mixtures of volume data and polygonally defined objects. The first method employs a hybrid ray tracer. Rays are simultaneously cast through the volume data and the polygonal objects, samples of each are drawn at equally spaced intervals along the rays, and the resulting colors and opacities are composited together in depth-sorted order. To avoid errors in visibility, volume samples lying immediately in front of and behind polygons are given special treatment. To avoid aliasing of polygonal edges, adaptive supersampling is used. The second method employs 3D scan-conversion with analytic anti-aliasing. Polygons are shaded, filtered, sampled, and combined with the volume data. The resulting composite dataset can then be rendered using the algorithm described in chapter 2. If the polygonal data is sufficiently bandlimited prior to sampling, this method also produces images free from aliasing artifacts.

To compare the relative versatility of these two methods, techniques for adding shadows and textures will also be presented. Nelson Max has written a brief but excellent survey of algorithms for casting shadows [Max86]. The present study employs a two-pass approach [Williams78, Reeves87], but stores shadow information in a 3D light strength buffer instead of a 2D shadow depth buffer. The amount of memory required for a 3D buffer is obviously much greater, but the representation has several advantages. By computing a fractional light strength at every point in space, penumbras and shadows cast by semi-transparent objects are correctly rendered. Moreover, the shadow aliasing problem encountered by Williams is reduced, but without resorting to expensive resampling methods such as are proposed by Reeves et al. Finally, the present algorithm also correctly handles shadows cast by volumetrically defined objects on themselves as well as shadows cast by polygons on volumetric objects and vice versa.

There are several kinds of texture mapping that might be useful when rendering mixtures of geometric and volume data. Wrapping textures around volumetrically defined objects requires knowing where their defining surfaces lie - a hard problem. Projecting textures through space and onto these surfaces is much easier and can be handled by a straightforward extension of the shading calculations described in chapter 2. Mapping textures onto polygons embedded in volume datasets is also relatively simple. The two latter techniques will be described in this chapter.

5.2. Two rendering algorithms

5.2.1. Hybrid ray tracer

The first rendering method we will consider is the hybrid ray tracer shown in figure 5.1. It begins as before with a 3D array of scalar values which is shaded and classified to yield a color $C_v(l)$ and an opacity $\alpha_v(l)$ for each voxel. Parallel viewing rays are then traced into the data from an observer position. For each ray, a vector of colors $C_v(U)$ and opacities $\alpha_v(U)$ is computed by resampling the volume data at equally spaced positions along the ray and tri-linearly interpolating from the colors and opacities in the eight voxels surrounding each sample location. Independently, all intersections between the ray and polygons in the environment are computed and shaded, yielding a color $C_p(x)$ and opacity $\alpha_p(x)$ for each point of intersection, which is denoted by a real vector of the form $x = (x,y,z)$ where $1 \leq x,y,z \leq N$. For simplicity, we assume that all polygons lie strictly within the boundaries of the volume dataset. Finally, the resampled volume colors and opacities are composited with each other and with the polygonal colors and opacities in depth-sorted order to yield a color $C(u)$ for the ray.

As discussed in section 2.2.3, volumetric compositing correctly renders the appearance of a gel composed of many small slabs each of identical size and homogeneous color and opacity. The thickness of each slab is equal to the spacing between samples along a viewing ray, and the width of the slab is equal to the spacing between adjacent rays as shown in figure 5.2a. If both spacings are set roughly equal to the spacing between voxels in the volume data, and tri-linear interpolation is used to compute the color and opacity of each slab as described in the previous paragraph, the resulting image will generally be free of aliasing artifacts.

When a polygon is embedded in the volume data as shown in figure 5.2b, it passes through some of these slabs, obscuring some portion of the gel in each slab and being obscured by the remainder. An exact solution of the hidden-volume problem inside every slab would be very expensive. Supersampling is an alternative, but also expensive. A solution of less accuracy and expense that has proved satisfactory in practice is to treat a polygon locally as a plane perpendicular to the ray and placed at the point of intersection as shown in figure 5.2c. An exact solution of the hidden-volume problem for this restricted case is simple. Rephrasing equation (2.8) for the opacity $\alpha_v(U)$ of a slab of homogeneous material in terms of its density $d_v(U)$ and thickness $t_v(U)$, we obtain the relation

$$\alpha_V(U) = 1 - e^{-\alpha_V(U)t_V(U)} \quad (5.1)$$

where $0 < \alpha_V(U) < 1$ and $\alpha_V(U) = 1$ signifies complete attenuation. Let the thicknesses of those portions of the slab lying immediately in front of and behind an embedded plane perpendicular to the viewing ray be denoted $t_F(U)$ and $t_B(U)$ respectively. Solving equation (5.1) for opacities $\alpha_F(U)$ and $\alpha_B(U)$ in terms of opacity $\alpha_V(U)$ and thicknesses $t_F(U)$ and $t_B(U)$ gives

$$\alpha_F(U) = 1 - (1 - \alpha_V(U))^{t_F(U)/t_V(U)} \quad (5.2a)$$

and

$$\alpha_B(U) = 1 - (1 - \alpha_V(U))^{t_B(U)/t_V(U)}. \quad (5.2b)$$

Working from front to back, we first composite $C_V(U)$ and $\alpha_F(U)$ into the ray, followed by $C_P(x)$ and $\alpha_P(x)$, and finally by $C_V(U)$ and $\alpha_B(U)$. If the ray intersects more than one polygon within a slab, the contribution made by each polygon and each sliver of volumetric gel must be computed and composited separately.

At polygonal edges, polygon-polygon intersections, and polygon shading highlights, such an approximation does not suffice to prevent aliasing. Therefore, the number of rays cast per pixel is increased in these regions using adaptive supersampling [Whitted80]. One difficulty with this approach is distinguishing whether the color difference observed in a sample region is due to volume data or geometric data, since supersampling is only appropriate in the latter instance for reasons given in section 4.2. The problem is solved by computing two additional colors during ray tracing as shown in figure 5.1. The first, denoted $C_G(u)$, contains contributions only from polygons (obtained by setting $\alpha_V(U) = 0$ for all volume samples). The second, denoted $C_A(u)$, contains contributions only from polygons, but attenuated by passage through the volume data (obtained by setting $C_V(U) = 0$ for all volume samples). If the range of $C_A(u)$ within a sample region exceeds some ϵ but the range of $C_G(u)$ does not, the observed color difference is due to spatially varying volume data rather than a geometric event (such as a polygonal edge). Supersampling is not required in this case. If the range of $C_G(u)$ exceeds ϵ but the range of $C_A(u)$ does not, the region contains a geometric event, but that event is hidden from view by overlying opaque volume data. No supersampling is needed in this case either. Only if the ranges of both colors exceed ϵ , signifying that the region contains a visible geometric event, should more rays be cast.

5.2.2. 3D scan-conversion

The second rendering method we will consider is 3D scan-conversion of the geometric data as shown in figure 5.3. We begin by shading and classifying the volume data to yield a color $C_V(i)$ and opacity $\alpha_V(i)$ for each voxel. Independently, each polygon is shaded, filtered, and sampled at the resolution of the volume data to yield a color $C_G(i)$ and opacity $\alpha_G(i)$ for each voxel. The polygons used in the present studies are of homogeneous color and are filtered by convolution with a 3D Bartlett window 4 voxels in diameter. The cost of scan-converting a polygon using this algorithm is proportional to its surface area.

These two sets of colors and opacities are combined using volume matting [Drebin88], specifically the *over* operator described in [Porter84], to yield a composite color $C_C(i)$ and opacity $\alpha_C(i)$ for each voxel. Since this operator is not commutative, the order in which polygons are scan-converted and whether polygons are matted over or under volume data must be considered. The composite dataset is then ray traced, resampled, and composited without giving further consideration to the geometric data it contains. Since the polygons have already been filtered, an image sampling rate of one ray per pixel usually suffices to prevent aliasing artifacts, even along polygonal edges.

5.3. Extensions to the rendering algorithms

5.3.1. Shadow calculations

Both of the rendering methods described above can be modified to cast shadow rays toward a light source from each non-empty sample position on a viewing ray. Opacity accumulated along such a shadow ray can be used to compute the strength of the light reaching the sample, which can in turn be used to shade the sample. Since the number of non-empty samples on a viewing ray may be high, this solution is expensive. It also requires that at least part of the shading calculations (given in equation 2.1) be delayed until image generation time.

A less computationally expensive solution would be to cast illumination rays into the data as a pre-processing step prior to image generation. The opacity accumulated at each sample position on such a ray represents the light strength at that point in space. This information can be stored in a 3D buffer. To generate an image, the buffer is resampled at each non-empty viewing ray sample position and used to shade the sample as in the previous solution. For S light sources, the cost of generating an image with shadows using this algorithm is $S+1$ times the cost of generating an image without shadows. Alternatively, the buffer can be resampled at each non-empty voxel position and used to shade the voxel prior to image generation, thus moving the computational burden from the image generation phase to the pre-processing phase of the rendering pipeline.

In view of its relatively low computational expense, the last solution has been chosen for implementation here. If used in conjunction with the hybrid ray tracer, a difficulty arises in that the passage of light through analytically defined polygons may give rise to sudden drops in light strength between successive sample positions on an illumination ray. These discontinuities must be filtered before they can be represented in the 3D light strength buffer. For this reason, the pre-filtered representation of polygons employed by the scan-conversion method makes it the more convenient starting point.

Figure 5.4 summarizes the modified algorithm. The first steps are shading and classification of the volume data, shading and scan-conversion of the polygons, and matting to yield for each voxel a set of composite colors $C_1(i), \dots, C_S(i)$ for light sources $s = 1, \dots, S$ and a composite opacity $\alpha_C(i)$. The contribution $C_V(i)$ by the volume data in voxel i to the composite color $C_s(i)$ for light source s is computed by separating equation (2.1) into equations of the form

$$C_V(i) = C_s \left[k_d(N(i) \cdot L_s) + k_r(N(i) \cdot H_s)^n \right] \quad (5.3)$$

where C_s , k_d , k_r , n , $N(i)$, L_s , and H_s are as defined in section 2.2.1.

Parallel illumination rays are then cast into the data from each light source as shown in figure 5.5. Let us assume that light source s is a square measuring P_s , illumination rays on a side. Rays are indexed by a vector $u_s = (u_s, v_s)$ where $u_s, v_s = 1, \dots, P_s$. For each ray, we assume some initial light strength $\beta_s(u_s)$, draw samples of the volume data at equally spaced positions along the ray, compute an opacity at each location by tri-linearly interpolating from the nearest eight voxels, and attenuate the strength of the ray in proportion to the computed opacities. Samples are indexed by a vector $U_s = (u_s, v_s, w_s)$ where (u_s, v_s) identifies the illumination ray, and $w_s = 1, \dots, W_s$ corresponds to distance along the ray with $w_s = 1$ being closest to the light source. The opacity of sample U_s is denoted $\alpha(U_s)$, and the set of strengths at sample U_s are denoted $\beta_1(U_s), \dots, \beta_S(U_s)$.

Attenuation of light strength along a ray is inversely proportional to accumulation of opacity along the ray and can be approximated using volumetric compositing. Working from the light source toward the data and adopting the notation of equation (2.11), the strength $\beta_{om}(u_s, U_s)$ of illumination ray u_s after processing sample location U_s can be computed from the strength $\beta_{in}(u_s, U_s)$ before processing the sample and the opacity $\alpha(U_s)$ of the sample by the relation

$$\beta_{om}(u_i; U_s) = \beta_{in}(u_i; U_s)(1 - \alpha(U_s)) \quad (5.4)$$

After this computation, the opacity is discarded, but the light strength is stored in a 3D light strength buffer. When all color and light strength volumes have been computed, a total color $C_T(i)$ for each voxel is computed as the weighted sum

$$C_T(i) = \frac{1}{k_1 + k_2 d(i)} \left[C_o k_o + \sum_{s=1}^S C_s(i) \beta_s(i) \right] \quad (5.5)$$

where the $\beta_s(i)$'s are obtained from the $\beta_s(U_s)$'s by mapping from voxel indices i to 3-space indices U_s in the coordinate system of light source s and tri-linearly interpolating from the nearest eight light strength values. The arrays of total color $C_T(i)$ and composite opacity $\alpha_C(i)$ are then ray traced, resampled, and composited as usual.

One problem that arises when using a sampled shadow representation is the tendency of surfaces to shadow themselves. The use of a 3D light strength buffer rather than a 2D shadow depth buffer reduces the severity of these artifacts, but does not eliminate them entirely. In the above algorithm, this problem applies to both naturally occurring surfaces (such as tissue boundaries) and those introduced using 3D scan-conversion. The solution adopted is to translate the 3D light strength volume a few voxels away from the light source just before computing the $C_T(i)$'s [Williams78, Reeves87]. While biasing of shadows in this manner necessarily reduces the accuracy of shadowing within small objects, it avoids introducing distracting aliasing artifacts.

5.3.2. Texture mapping

Projecting textures through space and onto volumetrically defined surfaces can be added to the shading and classification calculations described in chapter 2 simply by mapping from voxel coordinates to a 2D texture array and resampling the information at that location. Suitable resampling methods are described in [Feibush80] and [Heckbert86]. Such a texture can be used to modify surface reflection coefficients, voxel opacity, or any other rendering parameter [Cook84]. A 3D texture array, also known as a solid texture [Peachey85], can be used in place of a 2D texture, thus simultaneously visualizing two 3D datasets.

Both of the rendering methods described in section 5.2 can be modified to support textured polygons. The pre-filtered representation of polygons that made the scan-conversion method of section 5.2.2 well suited to shadow casting would severely blur any texture applied to the polygons prior to scan-conversion. For this reason, texture mapping is better handled by the hybrid ray tracer described in section 5.2.1. As part of the shading calculations performed at each point of intersection between a ray and a polygon, a mapping is performed from object space to a 2D or 3D texture array as shown in figure 5.5. The texture array is then resampled and used to modify the shading calculations as described above.

Assuming that the texture array is properly filtered during resampling, image supersampling in the interior of textured polygons is not necessary. To prevent the addition of textures from triggering the casting of excessive number of rays, the adaptive supersampling procedure described in the last paragraph of section 5.2.1 must be modified. Specifically, those aspects of the polygon shading model that are analytic and might give rise to sudden changes in polygon color $C_p(x)$ or opacity $\alpha_p(x)$ should be included in the supersampling decision, whereas contributions by sampled functions, which are assumed bandlimited, should not. In the current implementation, this rule is satisfied by including directional shading and depth cueing but not texture mapping in the computation of colors $C_G(u)$ and $C_A(u)$.

5.4. Optimization of the rendering algorithms

The hierarchical spatial enumeration described in section 3.2.1 is readily integrated with the 3D scan-conversion method described in section 5.2.2 by constructing a pyramid of binary volumes from the array of composite opacities $\alpha_C(i)$. This solution speeds the tracing of rays through both volume and geometric data. The resulting pyramid depends on both the volume and geometric data, however, and must re-computed if either changes. For the hybrid ray tracer described in section 5.2.1, a pyramid can be used to speed the tracing of rays through segments of volume data lying between successive polygon intersections in the hybrid ray tracer. The pyramid is independent of the polygon geometry and need not be recomputed if the latter changes. This method does not, however, reduce the cost of computing intersections between rays and polygons. The addition of a separate data structure to handle polygonal data (e.g. bounding volumes or spatial subdivision - see section 3.4), or the use of a single data structure to represent both polygons and volume data, are among the possible solutions to this problem.

Adaptive termination of ray tracing as described in section 3.2.2 is readily integrated with both of the rendering methods described in this chapter. In the case of the hybrid ray tracer, adaptive termination reduces both the number of voxels that must be resampled and the number of ray-polygon intersections that must be shaded. Note that this optimization must be applied independently in the computation of each of $C_G(u)$ and $C_A(u)$ (see section 5.2.1) so as not to adversely affect the adaptive supersampling process.

The progressive refinement algorithm described in chapter 4 is well suited to the scan-conversion method in which polygons are filtered prior to ray tracing. It may also be integrated with the adaptive supersampling employed in the hybrid ray tracer, but one runs the risk of missing small polygons if the initial sampling rate is too low.

5.5. Implementation details

Of the two rendering algorithms, two extensions, and three optimizations described in the foregoing sections, one has been selected for description in detail: the optimized hybrid ray tracer without extensions. Pseudo-code for this algorithm, which replaces all pseudo-code in sections 2.3, 3.3, and 4.3, follows:

```

procedure RenderVolume3( ) begin
    {Compute color and opacity for each voxel in dataset}
    for all i in Dataset do begin
        ComputeOpacity(i);
        if  $\alpha(i) > 0$  then
            ComputeColor(i);
    end

    {Initialize level-of-detail parameters and flag arrays}
     $\omega_{\max} := First\omega_{\max}()$ ;  $\omega_V := First\omega_V()$ ;  $\omega_G := First\omega_G()$ ;  $\epsilon := First\epsilon()$ ;
    for all u in ImagePlane do begin
         $F(u) := 0$ ;  $G(u) := 0$ ;
    end

```



```

{Loop until image is fully refined}
while  $\omega_{max} > \omega_G$  do begin
    {Divide image plane into sample regions and cast some (more) rays}
    for  $u := \{1, \omega_{max}, \dots, S\}^2$  do
        RayTraceRegion2( $u, \omega_{max}, \omega_V, \omega_G, \epsilon$ );
    {Redivide image plane into regions and interpolate any missing pixels}
    for  $u := \{1, \omega_{max}, \dots, S\}^2$  do
        InterpolateRegion2( $u, \omega_{max}, \omega_V$ );
    {Gather contributions from all samples lying in  $2\omega_D$  by  $2\omega_D$ }
    {open region centered on each pixel}
    for  $u := \{1, \omega_D, \dots, S\}^2$  do begin
        for  $v := \{0, \omega_D\}^2, w := \{\omega_D, 0\}^2$  do
             $\tilde{C}(v) := \text{AverageRegion}(u - (\omega, \omega) + v, \omega_D, w)$ ;
         $C(u) := \sum \tilde{C}(v)/4$ ;
    end
    {Display image, then clear all interpolated colors}
    DisplayImage2( );
    for all  $u$  in ImagePlane do
        if not  $F(u)$  then  $G(u) := 0$ ;
    {Increment level-of-detail parameters}
     $\omega_{max} := \text{Next}\omega_{max}()$ ;  $\omega_V := \text{Next}\omega_V()$ ;  $\omega_G := \text{Next}\omega_G()$ ;  $\epsilon := \text{Next}\epsilon()$ ;
end
end RenderVolume3

procedure RayTraceRegion2( $u, \omega, \omega_V, \omega_G, \epsilon$ ) begin
    {Cast rays from four corners of region}
    for  $v := \{0, \omega\}^2$  do
        if  $u+v$  in ImagePlane and not  $F(u+v)$  then begin
            TraceRay3( $u+v$ );  $F(u+v) := 1$ ;
        end
    {If region is larger than  $\omega_V$  by  $\omega_V$  pixels and color difference > threshold,}
    {or if region is larger than  $\omega_G$  by  $\omega_G$ }
    {and  $C_G$  and  $C_A$  color differences > threshold,}
    {divide into four subregions and continue ray tracing}
    if  $\left[ \omega > \omega_V \text{ and } \text{Difference}_V(u, \omega, \epsilon) \right]$  or  $\left[ \omega > \omega_G \text{ and } \text{Difference}_G(u, \omega, \epsilon) \right]$  then
        for  $v := \{0, \omega/2\}^2$  do
            RayTraceRegion2( $u+v, \omega/2, \omega_V, \omega_G, \epsilon$ );
    end RayTraceRegion2

```

```

procedure InterpolateRegion2(u,ω,ωD) begin
  (Interpolate colors at midpoints of sides and at center of region)
  for v := [(0,ω/2),(ω,ω/2),(ω/2,0),(ω/2,ω),(ω/2,ω/2)] do
    if u+v in ImagePlane and not F(u+v) and not G(u+v) then begin
      Interpolate(u,ω,u+v); G(u+v) := 1;
    end
  (If region is larger than ωD by ωD pixels.)
  (divide into four subregions and continue interpolation)
  if ω/2 > ωD then
    for v := (0,ω/2)2 do
      InterpolateRegion2(u+v,ω/2,ωD);
end InterpolateRegion2

```

```

procedure AverageRegion(u,ω,w) begin
  (If region is larger than ωG by ωG)
  (and was subdivided during ray tracing or interpolation.)
  (divide into four subregions and continue gathering contributions)
  if ω > ωG and [ F(u+(ω/2,ω/2)) or G(u+(ω/2,ω/2)) ] then begin
    for v := (0,ω/2)2 do
      C̃(v) := AverageRegion(u+v,ω/2,w/2);
    return ∑ C̃(v)/4;
  end

```

```

  (Else return corner closest to display pixel as color of region)
  else return C(u+w);
end AverageRegion.

```

```

procedure TraceRay2(u) begin

```

```

  xp1 := NextPoly(x,u);

```

```

  (Loop through volume dataset,

```

```

  (terminating early if polygon-only opacity αG > threshold)

```

```

  while x in Dataset and αG(u) ≤ 1 - ε do begin

```

```

    (Loop through volume data lying between)

```

```

    (current position and next ray-polygon intersection.)

```

```

    (terminating early if volumetric+polygon opacity α > threshold)

```

```

    xp2 := NextPoly(x,up1);

```

```

    while x < xp2 and α(u) ≤ 1 - ε do begin

```

{If level zero cell contains a one, render it,
 (but only portion lying between ray-polygon intersections)
 if $V_m(I)$ then $RenderCell_2(u,x,Next(m,x,u),x_{p_1},x_{p_2})$;

{Advance to next cell and maybe jump to higher level,
 (but advance no farther than next ray-polygon intersection)

$x := \min(Next(m,x,u),x_{p_2})$;

end
 $RenderPoly(u,x_{p_2})$;

$x_{p_1} := x_{p_2}$;

end

end *TraceRay*₃

procedure $RenderCell_2(u,x_1,x_2,x_{p_1},x_{p_2})$ begin

$U_1 := \left[\max(Image(x_1), Image(x_{p_1}) - (0,0,1/2)) \right]$;

$U_2 := \left[\min(Image(x_2), Image(x_{p_2}) + (0,0,1/2)) \right]$;

$U_{p_1} := Image(x_{p_1})$;

$U_{p_2} := Image(x_{p_2})$;

{Loop through all samples falling within cell
 {and lying between ray-polygon intersections}

for $U := U_1$ to U_2 do begin

$x := Object(U)$;

{If any of eight surrounding voxels have opacity > 0,
 (then resample color and opacity and composite into ray.)
 {including only portion lying between ray-polygon intersections.)
 if $V_o(Index(0,x))$ then begin

$\hat{C}(U) := Sample(\hat{C},x)$;

$\alpha(U) := Sample(\alpha,x)$;

$t := 1$;

if $U_{p_1} > U - (0,0,1/2)$ then

$t := t - U_{p_1} - U + (0,0,1/2)$;

if $U_{p_2} < U + (0,0,1/2)$ then

$t := t - U + (0,0,1/2) - U_{p_2}$;

```

         $\alpha(U) := 1 - (1 - \alpha(U))^4;$ 
         $\hat{C}(u) := \hat{C}(u) + \hat{C}(U)(1 - \alpha(u));$ 
         $\alpha(u) := \alpha(u) + \alpha(U)(1 - \alpha(u));$ 
    end
end
end RenderCell2

procedure RenderPoly(u,xp) begin
     $\hat{C}(u) := \hat{C}(u) + \hat{C}_p(x_p)(1 - \alpha(u));$ 
     $\alpha(u) := \alpha(u) + \alpha_p(x_p)(1 - \alpha(u));$ 
     $\hat{C}_A(u) := \hat{C}_A(u) + \hat{C}_p(x_p)(1 - \alpha(u));$ 
     $\hat{C}_G(u) := \hat{C}_G(u) + \hat{C}_p(x_p)(1 - \alpha_G(u));$ 
     $\alpha_G(u) := \alpha_G(u) + \alpha_p(x_p)(1 - \alpha_G(u));$ 
end RenderPoly.
```

ω_V in this implementation is equivalent to ω_{min} in section 4.3, ω_D is the size of a display pixel (assumed to be 1 in section 4.3) where $\omega_D \leq \omega_V$, and ω_G is the minimum region size during adaptive supersampling where $\omega_G \leq \omega_D$. To allow rays to be shared by adjacent sample regions, thus minimizing the number of rays traced, the image array in section 4.3 has been replaced with an image plane array measuring S samples on a side where $S = \frac{\omega_D}{\omega_G} P$. The additional storage

allows rays all traced during supersampling to be retained throughout the refinement process. An open region of size $2\omega_D$ by $2\omega_D$ centered on each pixel is used during averaging down because it encloses the largest number of image samples (in the vicinity of polygonal edges) without everywhere blurring the rendition of volume data (samples of which are spaced ω_D apart in the absence of a polygonal edge). The *DisplayImage₂* procedure displays the image formed by entries spaced ω_D apart in the image plane array. The *Difference_v* procedure is equivalent to the *Difference* procedure in section 4.3. The *Difference_G* procedure decides if the range of both $C_G(u)$ and $C_A(u)$ as described in section 5.2.1.

The *NextPoly* procedure computes the object-space coordinates of the next polygon intersected by a ray starting at the specified object-space location and moving in the specified direction. The *RenderCell₂* procedure composites the contribution made to a ray by the specified interval of volume data, but only that portion lying between the specified ray-polygon intersections. The *RenderPoly* procedure composites the contribution made to a ray by the specified polygon, and also updates computes $C_G(u)$ and $C_A(u)$ as described in section 5.2.1.

The memory required for the hybrid ray tracer is equal to that of the algorithm given in chapter 3 plus a relatively small amount of storage to hold the polygonal database. The time required to perform shading, classification, and pyramid construction are also similar to those given in chapter 3. The time required to generate an image using the hybrid ray tracer is approximately equal to the sum of the times required to separately render those portions of the geometric and volume data that are visible in the composite image. Since the present implementation includes no means for reducing the cost of computing ray-polygon intersections, the cost of computing each intersection is proportional to the number of polygons in the database.

5.6. Case studies

In order to compare the two rendering methods described in this chapter, let us consider a simple test environment consisting of three mutually perpendicular polygons embedded in a $256 \times 256 \times 113$ voxel computed tomography (CT) study of a human head. Figures 5.6a and 5.6b are 380×380 pixel views of this data rendered with the hybrid ray tracer described in section 5.2.1 and the scan-conversion method described in section 5.2.2 respectively.

Shading and classification of the volume data took about 2 minutes on a Sun 4/280. Scan-conversion of the large polygons used here took about 1 second apiece. Construction of a pyramid of binary volumes from the array of voxel opacities (for the hybrid ray tracer) or composite opacities (for the scan-conversion method) took about 1 minute. Ray tracing, interpolation, and averaging down (for the hybrid ray tracer) to generate figures 5.6a and 5.6b took 80 seconds and 70 seconds respectively. Slightly less than one ray per pixel was traced on average. By contrast, the volume data without the polygons can be rendered with the same level of refinement in about 50 seconds.

Figures 5.7a and 5.7b show details from figures 5.6a and 5.6b. In general, polygon edges produced by the hybrid ray tracer are sharper than those produced by 3D scan-conversion, although a higher image generation cost is paid for the better rendition. The slight aliasing noticeable in the bony tissue in these and other figures is due to insufficient bandlimiting during CT scanning and is not a result of the rendering process.

A visualization of where rays were cast during generation of figures 5.6a and 5.6b is given in figures 5.8a and 5.8b. Each pixel in these visualizations corresponds to one position in the image plane array, and each 4×4 block of pixels corresponds to a single pixel in figure 5.6. White pixels in the visualizations correspond to cast rays, and black pixels to samples filled in by interpolation. As expected, the number of rays per unit area is lowest in the interiors of homogeneous regions, higher along the silhouettes of volumetrically defined objects, and in the case of the hybrid ray tracer, higher still along polygonal edges.

Figure 5.9 shows the effect of casting shadows using the algorithm described in section 5.3.1. The scene contains two light sources: a low-intensity light shining over the observer's right shoulder and a high-intensity light shining up from below and to the left. To insure that the shapes of shadowed objects are not completely obscured, shadows were only computed for the high-intensity light. While this is not strictly correct, the goal is enhanced insight, not photo-realism. Initial light strengths for the high-intensity source were assigned from a texture containing a filtered rectangular grid. The effect is to project this texture through the dataset and onto all illuminated surfaces, including the five scan-converted polygons. The addition of shadows roughly doubles the time required to compute the array of total voxel colors $C_T(i)$, but does not affect the time required to generate each frame in a rotation sequence (assuming fixed object and light sources and moving observer as discussed in section 2.4).

Figure 5.10 shows the effect of mapping a texture onto embedded polygons using the algorithm described in section 5.3.2. The effect on image generation time of adding textures depends on the number and size of textured polygons. For this example, image generation time increased from 80 seconds to 120 seconds. Although a whimsical texture has been used here, the technique may be used to display measurement grids or secondary datasets.

Figure 5.11 suggests one possible way in which these techniques might be applied to the problem of radiation treatment planning. A polygonally defined treatment volume (enclosing the tumor - in purple) and treatment beam (in blue) have been added to the color visualization on a CT dataset shown in figures 2.13 and 2.14. Scaling the opacities of all voxels inside a region of interest down to nearly zero helps clarify the 3D relationships between the various objects.

Figure 5.12 illustrates how scan-converted backdrop planes and cast shadows can be used to enhance comprehension of an isovalue contour surface from the Staphylococcus Aureus ribonuclease electron density map used in chapter 3. In this visualization, a color-coded stick representation of the molecular structure has been superimposed on the image to aid in its

interpretation.

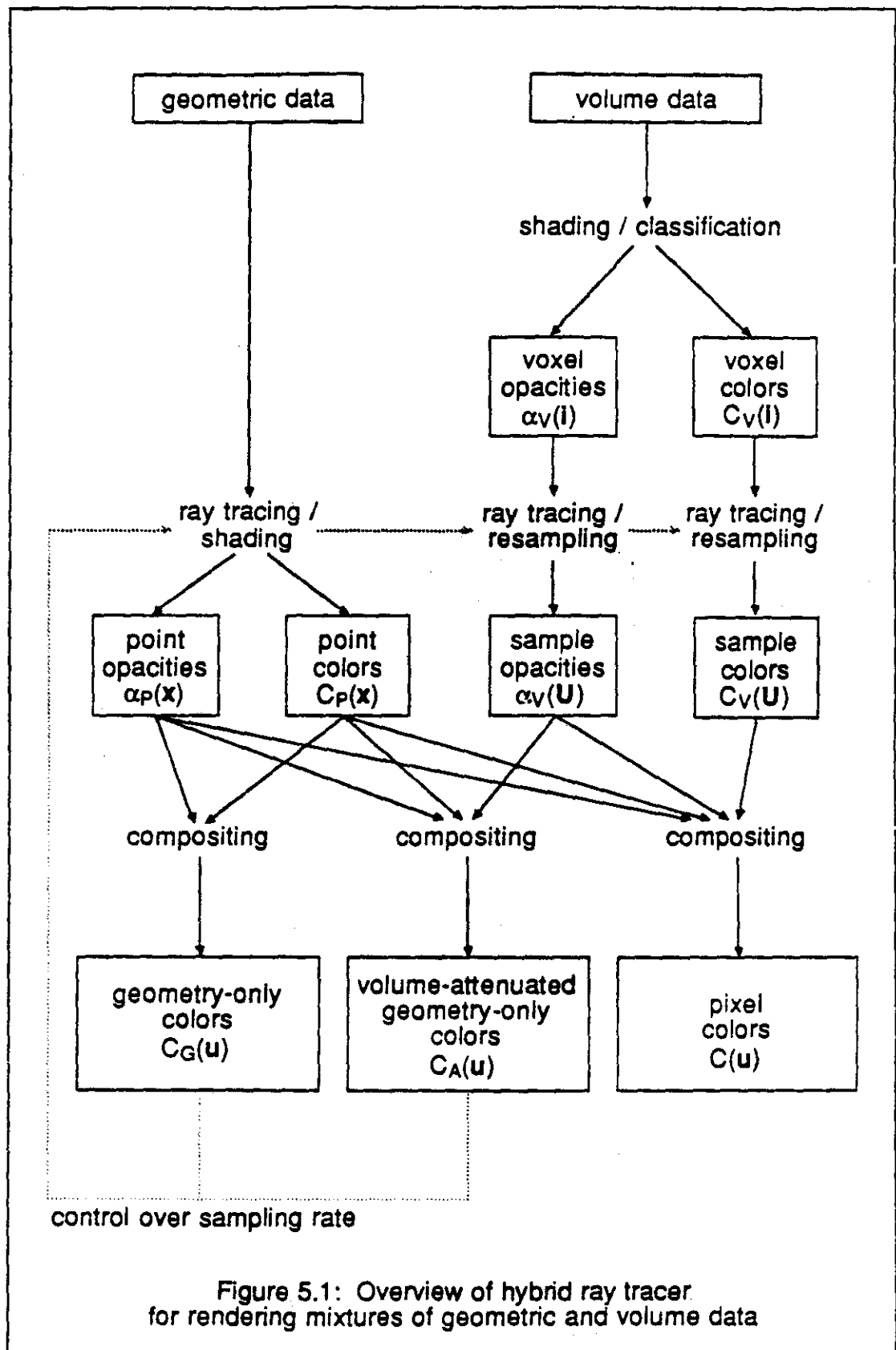
5.7. Summary and discussion

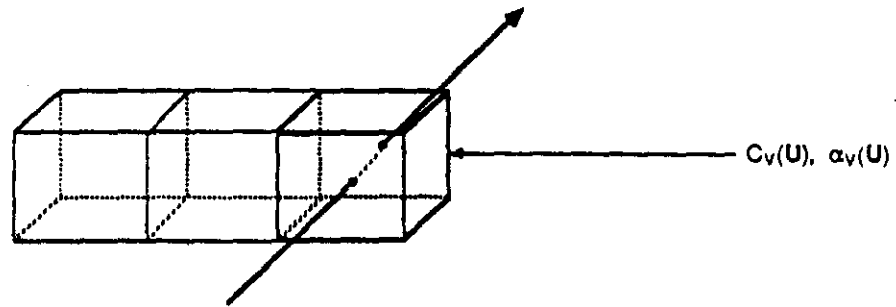
Two methods for rendering mixtures of geometric and volume data have been described. Although both are reasonably efficient and produce images free from aliasing artifacts, each have advantages and disadvantages in terms of cost, image quality, and versatility. In particular, it was found that the 3D scan-conversion method was better suited than the hybrid ray tracer for rendering shadows, while the converse seems to hold for rendering textured polygons. A single algorithm capable of rendering both shadows and textured polygons has not yet been developed.

A number of improvements to these methods can be suggested. The current implementation makes frequent use of tri-linear interpolation for resampling 3D data. A better filter would reduce the amount of blurring required during 3D scan-conversion, yielding sharper polygonal edges. In a similar vein, the shadow casting algorithm includes three successive resampling steps. By reorganizing the order of operations, one resampling can be eliminated, yielding crisper shadows. Finally, the solution proposed for handling volume samples near ray-polygon intersections treats the polygons locally as a plane perpendicular to the ray. Proper consideration of the angle the polygon makes with the ray would increase the accuracy of these calculations while adding only modestly to their expense.

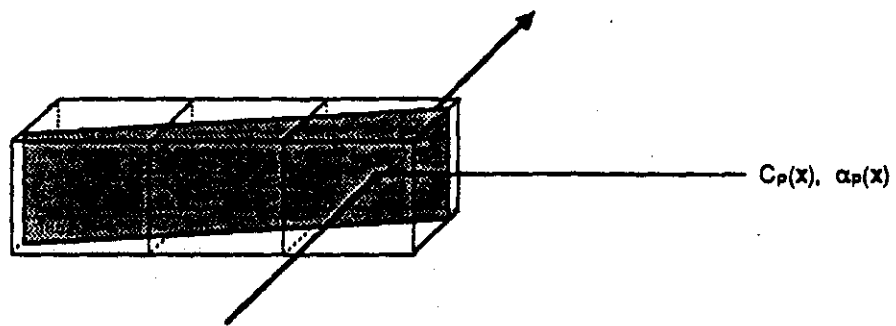
These studies have led to several welcome but unexpected results. Polygons rendered using 3D scan-conversion appear to have a finite thickness when rotated. Their opacity also varies with their angle relative to the view direction, as would a real slab of semi-transparent gel. Far from being distracting, these effects enhance the viewer's understanding of their shape and orientation. The embedding of polygons in volume data also seems to improve comprehension of the latter. For example, when the CT study of the head is bisected by a gridded, coronally-oriented (in the plane of the face) polygon and the ensemble is rotated, the presence of a backdrop of known shape and pattern improves appreciation of the fine structure of the sinuses and eye orbits. This suggests that, in addition to their primary role representing man-made or abstract entities, geometric primitives may be useful as diagnostic tools in the study of volume datasets.

Several strategies for treating mixtures of geometric and volume data remain unexplored. For example, geometric objects defined by extrusion of 2D profiles can be rendered using the shadow casting algorithm. If the profile is loaded into the array of initial light strengths, and all voxels in the volume data are assigned a slightly non-zero opacity, one obtains the effect of sculpted shafts of light passing through a dust-filled room. This technique could be used to visualize a radiation beam for cancer treatment.

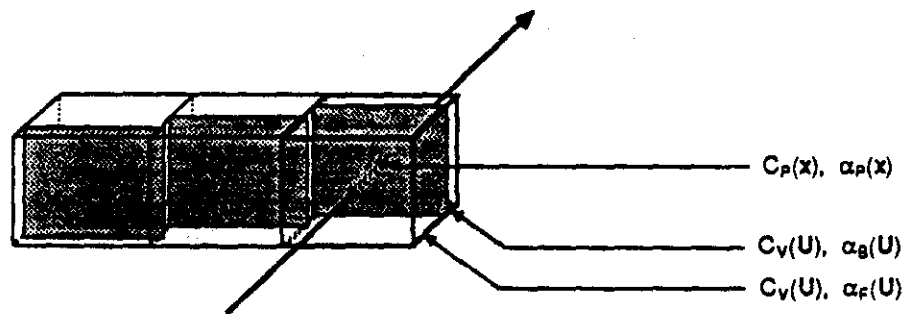




a - Slab of volume data along viewing ray



b - Polygon embedded in volume data



c - Approximate solution to hidden-volume problem

Figure 5.2: Rendering of polygon embedded in volume data

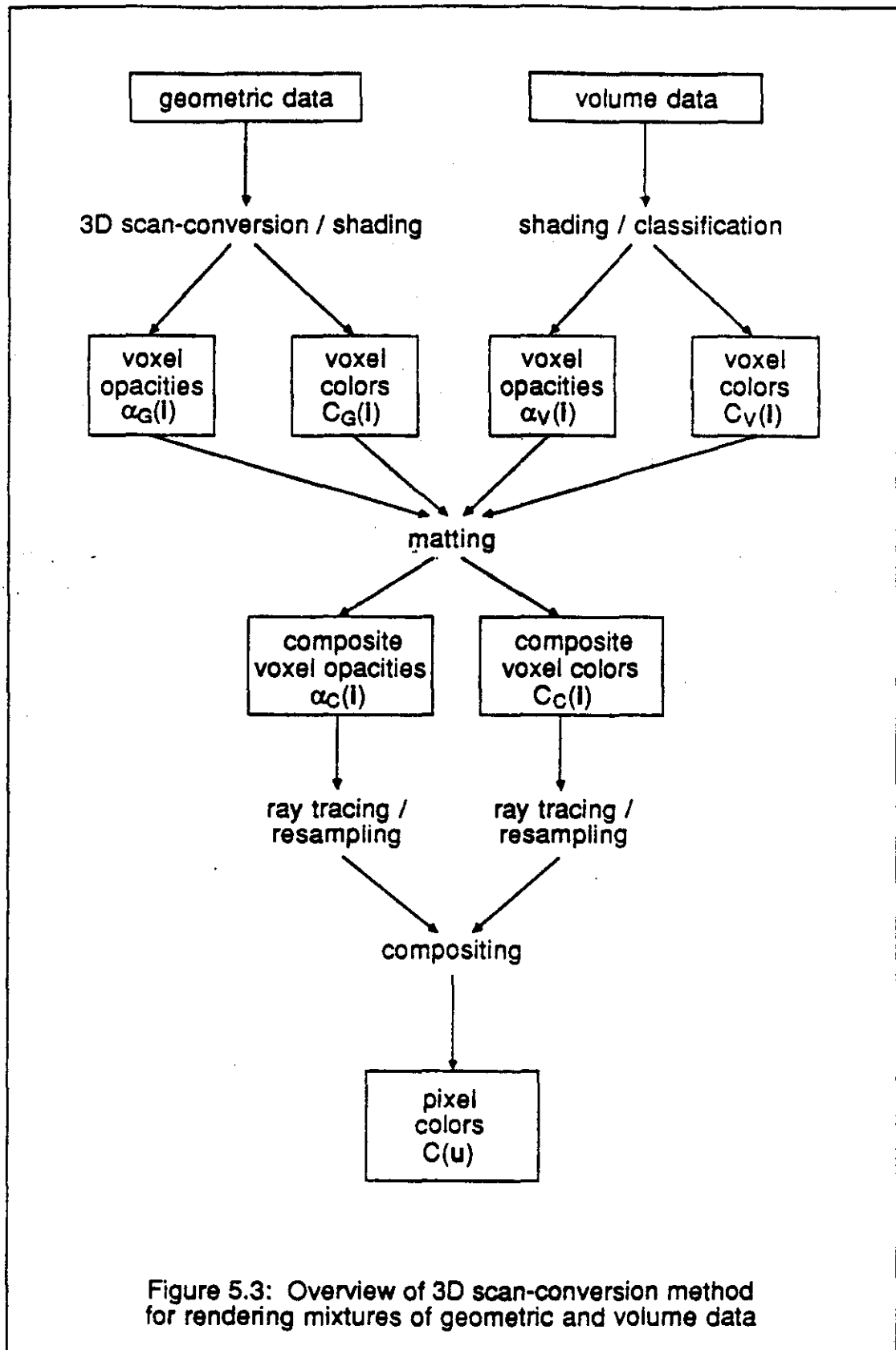


Figure 5.3: Overview of 3D scan-conversion method for rendering mixtures of geometric and volume data

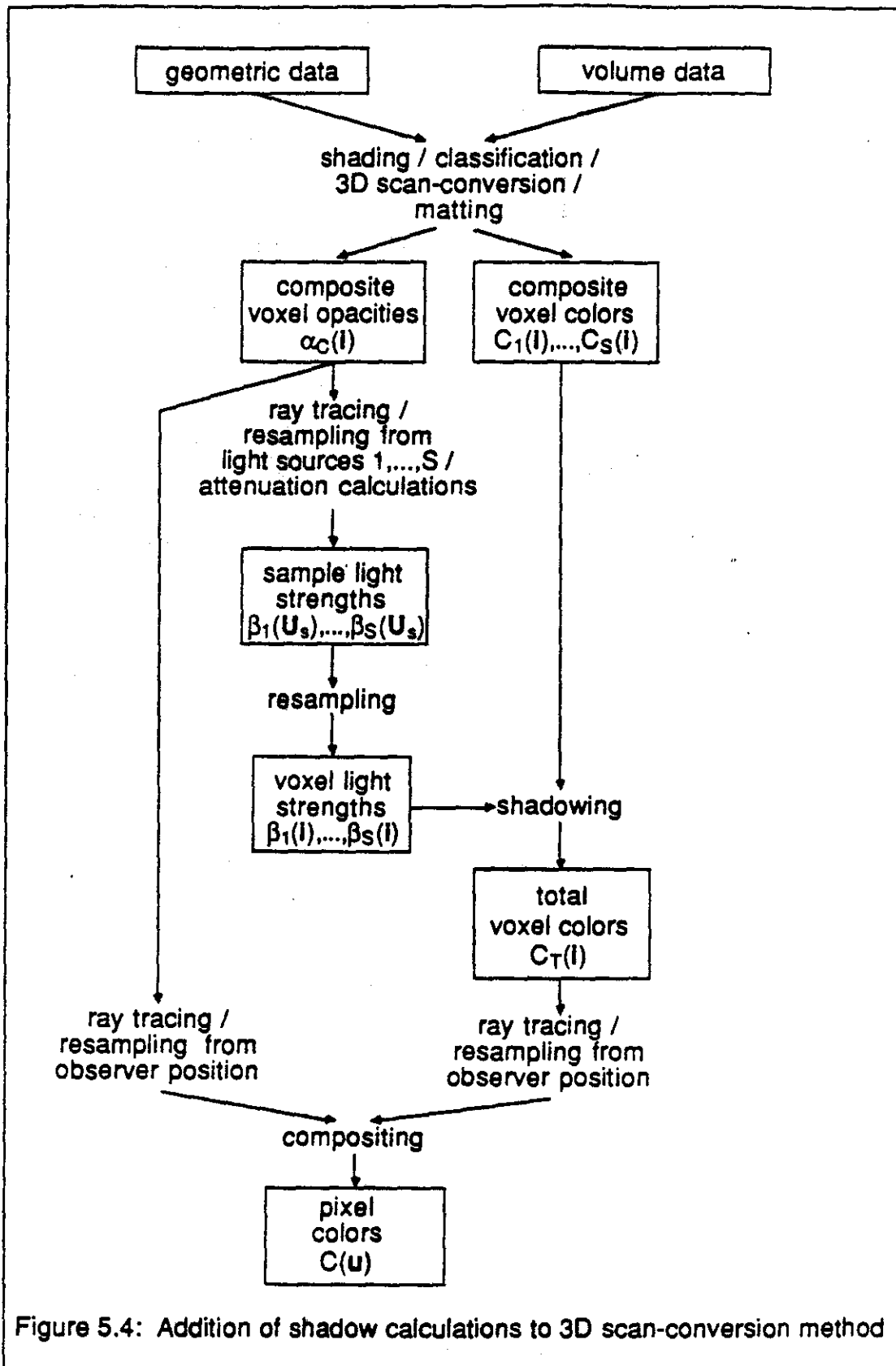
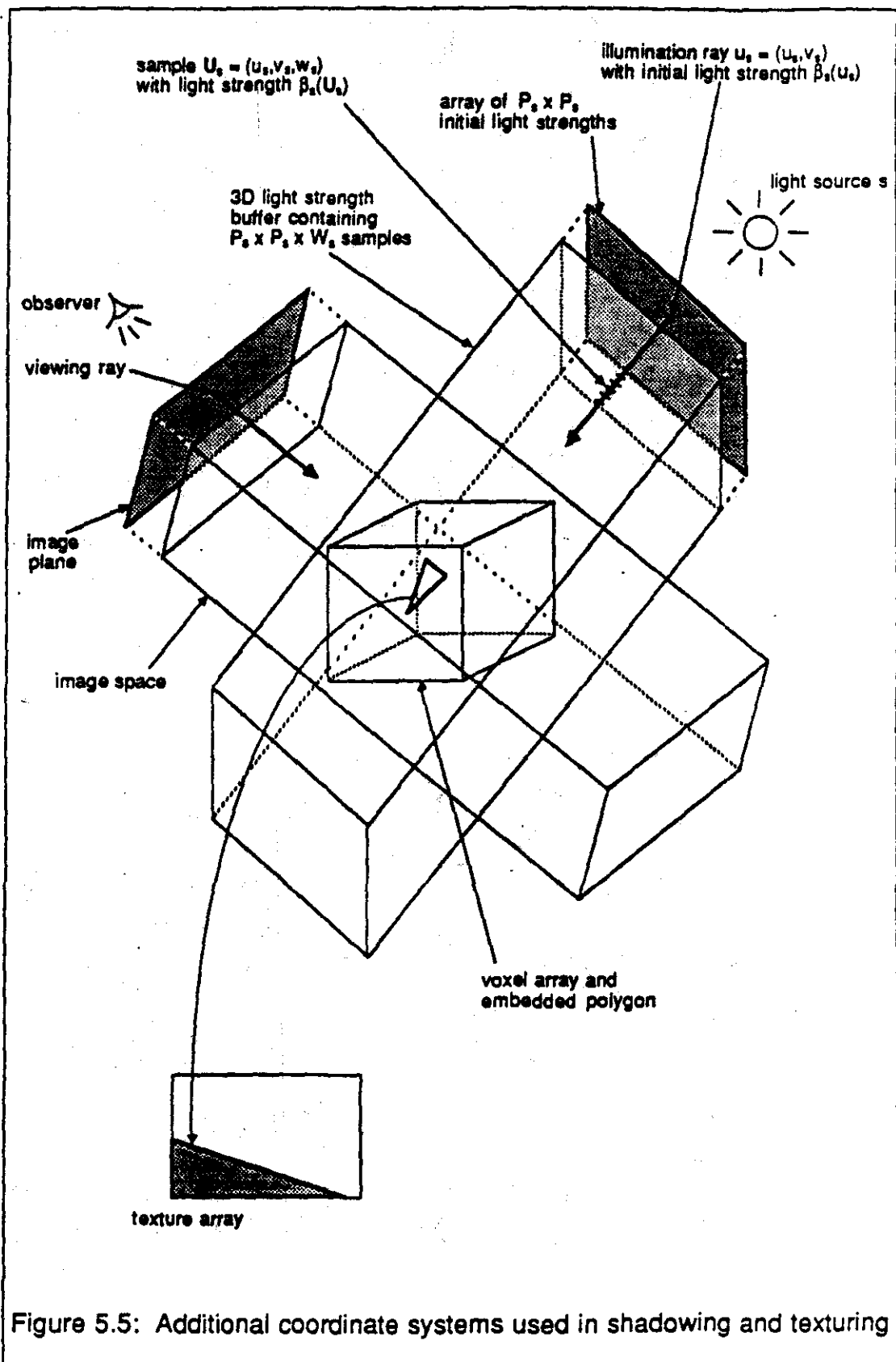


Figure 5.4: Addition of shadow calculations to 3D scan-conversion method



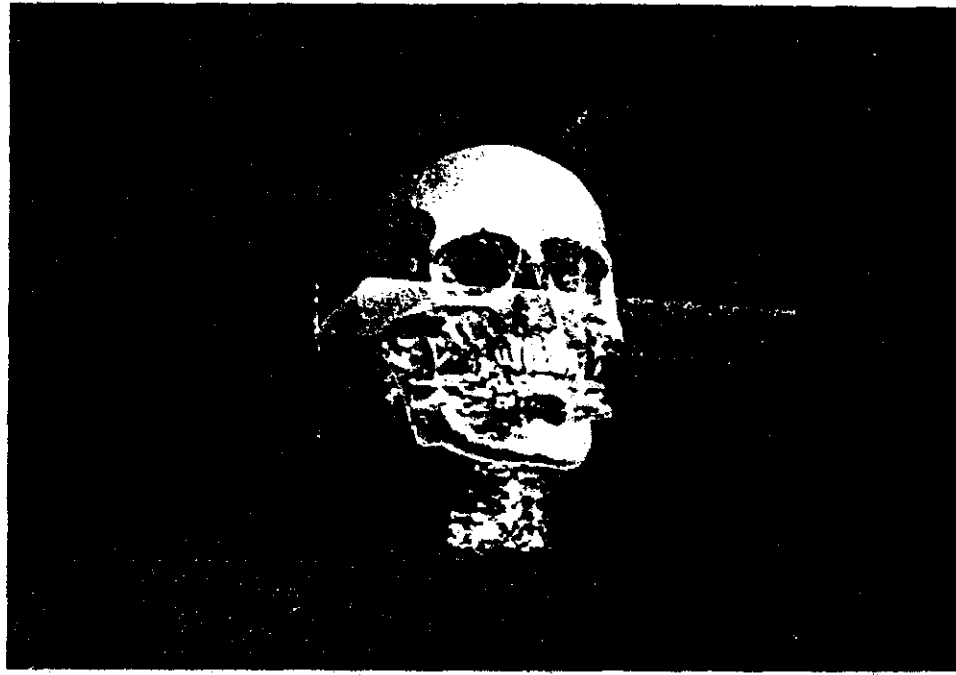


Figure 5.6a: Color rendering of CT dataset and embedded polygons, generated using hybrid ray tracer

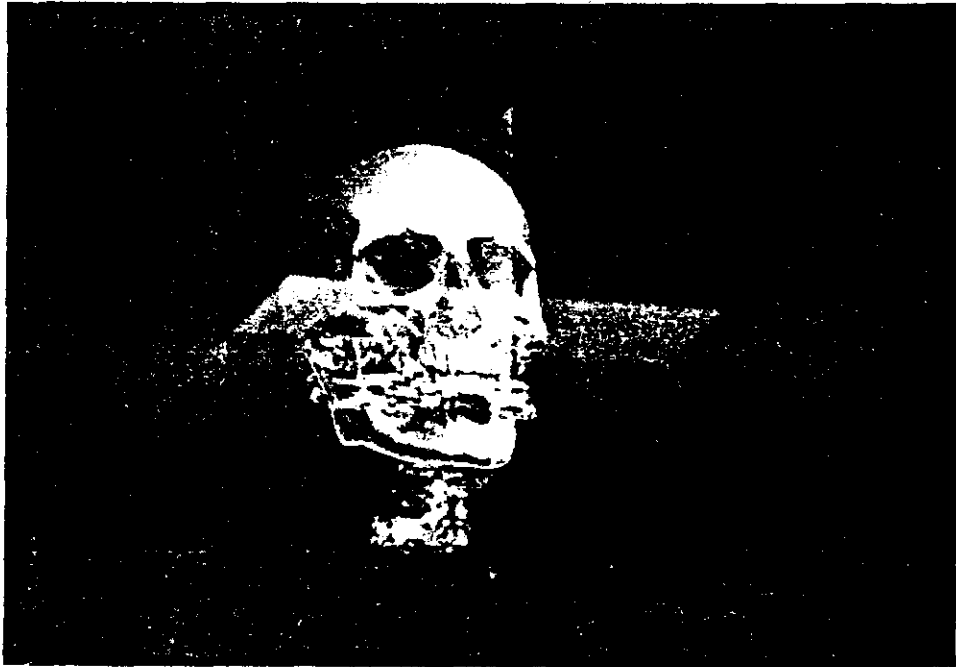
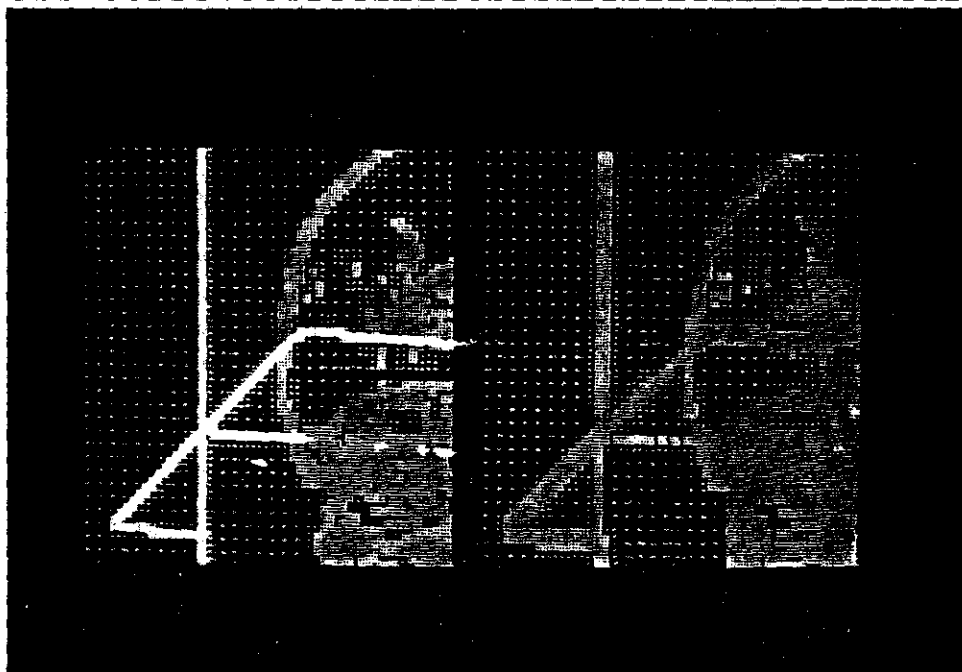


Figure 5.6b: Color rendering of CT dataset and embedded polygons, generated using 3D scan-conversion method



Figures 5.7a and 5.7b: Details from figures 5.6a and 5.6b, comparing image quality of hybrid ray tracer and 3D scan-conversion methods



Figures 5.8a and 5.8b: Visualization of where rays were cast to generate figures 5.6a and 5.6b

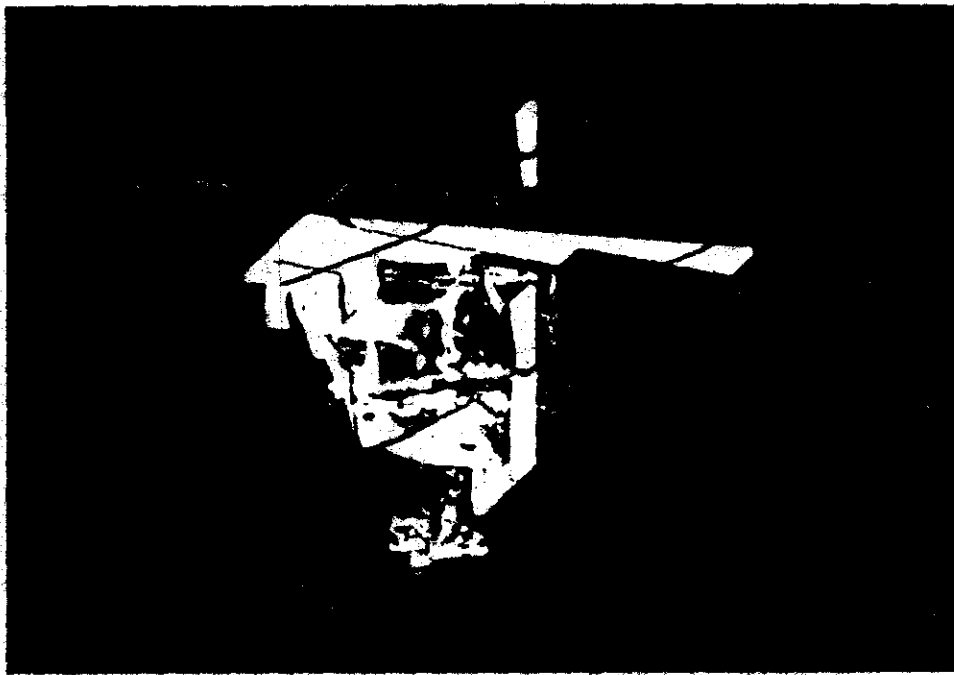


Figure 5.9: Color rendering with shadows of CT dataset and embedded polygons, generated using modified 3D scan-conversion method



Figure 5.10: Color rendering of CT dataset and textured polygons, generated using modified hybrid ray tracer

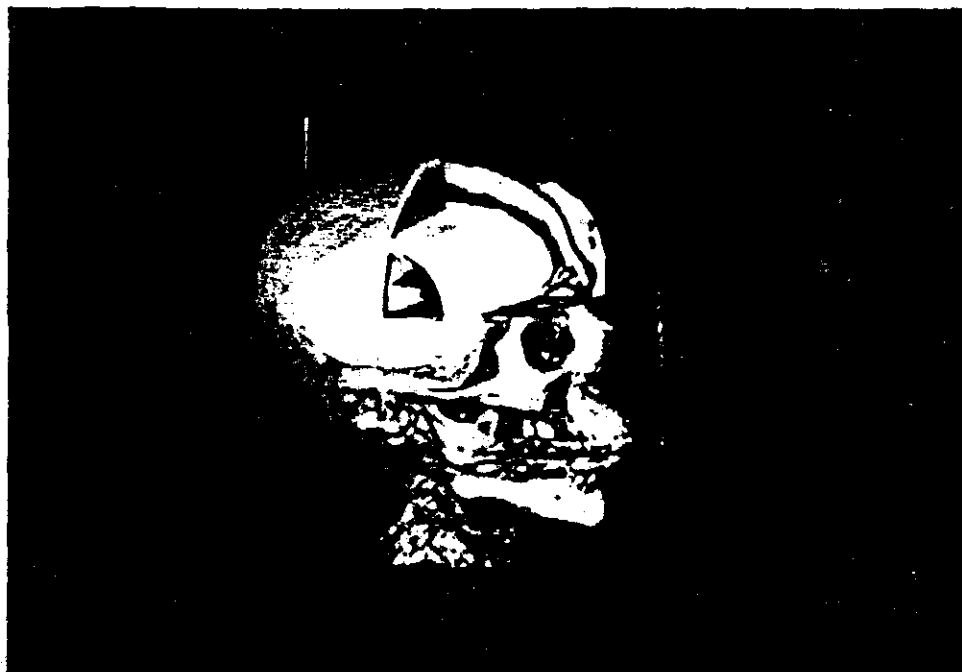


Figure 5.11: Color rendering of CT dataset showing bone, soft tissue, tumor (purple), and radiation treatment beam (blue)

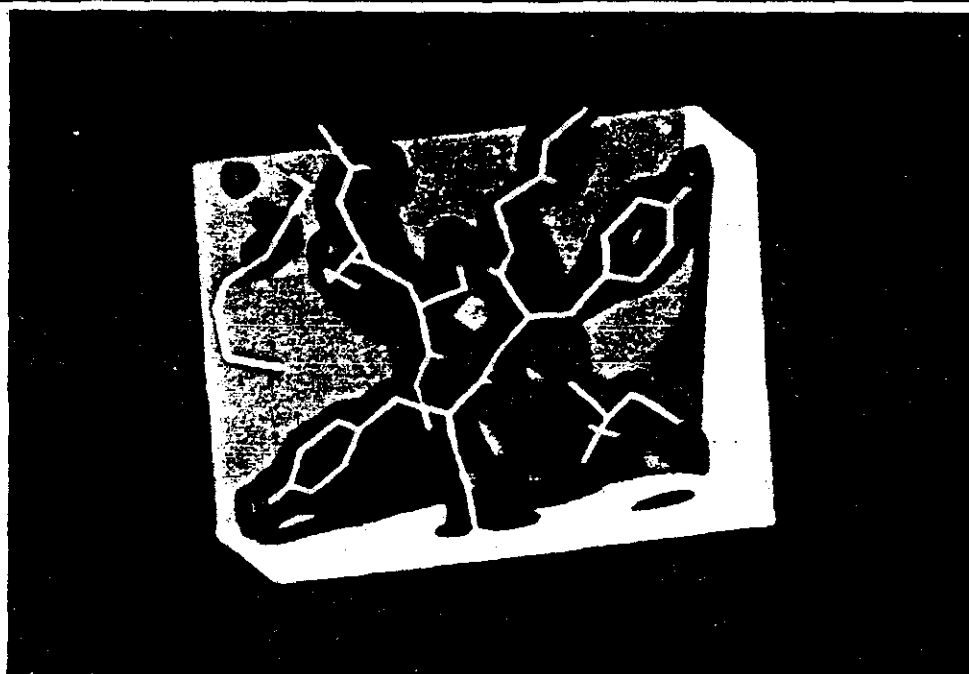


Figure 5.12: Color rendering with shadows of electron density map and embedded polygons

CHAPTER VI

FINAL SUMMARY AND TOPICS FOR FUTURE RESEARCH

Surface-based and binary voxel techniques for displaying sampled scalar fields of three spatial dimensions have been in use for about ten years and have been applied to a variety of scientific, medical, and engineering problems. The most prominent disadvantage of these techniques is poor image quality. They generally start by thresholding the incoming data, a process that often leads to artifacts in the generated image.

Within the past two or three years, the problem of low image quality has been largely solved with the development of volume rendering. The key characteristic of this approach is its use of partial object opacity, which eliminates the necessity of making a binary classification of the incoming data and thus eliminates many of the artifacts plaguing the other techniques.

Despite its advantages, volume rendering has proven to suffer from a number of problems. High on this list is the technique's computational expense. Since all voxels participate in the generation of each image, rendering time grows linearly with the size of the dataset. Published techniques take minutes or even hours to generate a single view of a large dataset using currently available workstation technology. Slow image generation has also constrained the design of convenient user interfaces for volume rendering. Yet another drawback of volume rendering is its lack of versatility. Many scientific problems require that sampled functions and analytically defined geometry appear in a single visualization. Strategies for rendering such mixtures have been suggested by various researchers, but none have yet been reported in the literature.

This thesis offers practical solutions to many of these problems. It presents an image-order volume rendering algorithm, demonstrates that it produces images of high quality, and proceeds to show how its computational expense can be reduced by taking advantage of spatial coherence in the 3D data and in its 2D projections. For the applications studied, these improvements speed up volume rendering by two orders of magnitude. For the special case of scenes consisting solely of opaque surfaces, image generation time has been observed to grow nearly linearly with the size of the image rather than linearly with the size of the dataset. This thesis also presents algorithms for mixing polygons and volume data in a single visualization, for casting shadows through volume data, and for embedding 2D textures in volume renderings in a variety of ways.

6.1. Comparison to Pixar volume rendering algorithm

Since the volume rendering algorithm developed by researchers at Pixar predates the algorithm described in this thesis, and because their volume rendering engine, the Pixar Image Computer, is widely used, it seems useful to briefly compare the two approaches. Pixar has not addressed the problem of displaying isovalue contour surfaces, so comparisons in this regard are impossible. We therefore concentrate on their technique for displaying region boundary surfaces as described in [Drebin88] and in the documentation accompanying their ChapVolumes software package.

Pixar's algorithm, like the one described in this thesis, consists of a shading and classification phase whose output is a 3D array of colors and opacities, and an image formation

phase whose output is a 2D image. Their shading and classification phase begins by estimating from the scalar value in each voxel an occupancy fraction for each of a set of materials that might be present in the voxel. A color, an opacity, and a density are then computed for the voxel by adding together the fraction of each material present in the voxel multiplied by a color, opacity, and density assigned to that material. Material density need not be related to physical density or to material opacity. It arises from an approximation of physical optics whose details are not important here. Its practical significance is that gradient vectors in the density array serve as surface normals during shading, and the magnitude of these vectors is used to scale the opacity of surfaces relative to the opacity of surrounding voxels.

The notion of explicitly computing material occupancy and material density is elegant, but the actual computations are not necessary. Pixar's shading and classification model and the model presented in chapter 2 are capable of producing identical results given appropriate parameter selections. Furthermore, the applicability of their technique to any particular dataset is constrained by the same material adjacency criteria as is described in section 2.2.2.2. Unfortunately, many details of their shading model are proprietary, and the ChapVolumes software package hides key shading parameters from the user, making verification of this hypothesis difficult. Subjective comparison of images suggests that Pixar usually assigns a lower opacity to surfaces and a higher opacity to voxels lying between surfaces than the examples in this thesis, and that they usually render their surfaces with less specular reflection than is used here.

One significant difference between the two approaches is that Pixar's surface normals are computed from classified data (their density array) whereas surface normal $N(i)$ in equation (2.1) is computed directly from the original data. The application of a non-linear operator prior to surface normal estimation in their technique has the potential for distorting apparent surface orientation. This is particularly true if the material occupancy classification contains errors. In the algorithm described in this thesis, shading and classification are entirely independent. Classification errors may produce surfaces that are too transparent or too opaque, but their apparent orientation will be correct.

Image formation in the Pixar algorithm is performed in object order. Specifically, they geometrically transform each slice of voxels from object space to image space using two-pass resampling techniques [Catmull80], project the slice onto the image plane, and blend it together with the projection formed by previous slices using compositing. The algorithm presented in this thesis operates in image order, tracing viewing rays from an observer position through the dataset. Once again, the details of their resampling algorithm are proprietary, making comparisons difficult. In a series of informal experiments, images generated by the two algorithms starting from identical arrays of color and opacity were found to be visually indistinguishable.

The use of an image-order rather than an object-order algorithm has significant computational advantages, however, as was demonstrated in chapters 3 and 4. Since Pixar's volume rendering technique is implemented on specialized hardware whereas the present algorithm is implemented in the C language on a general-purpose compute engine, experimental performance comparisons are nearly impossible. Shading and classification of a $256 \times 256 \times 113$ voxel dataset on the Pixar Image Computer using ChapVolumes software takes about two minutes, and generation of a single image from the resulting array of colors and opacities takes about one minute. These timings are roughly equal to the results reported in this thesis. It is presumed that the Pixar Image Computer makes up in hardware and optimized firmware the time it loses by employing a fundamentally less efficient algorithm.

6.2. Real-time volume rendering

The near-term prospects for real-time volume rendering are encouraging. Kaufman has written an excellent survey of architectures designed for rendering voxel data [Kaufman86a], including his own CUBE (CUBic frame Buffer) system currently under development [Kaufman88a]. Most of the machines he surveys start by thresholding the incoming data. At

this writing, the fastest system based on volumetric compositing is probably the Pixar Image Computer. Using a four channel SIMD processor [Levinthal84] and Pixar's ChapVolumes software package, the Image Computer can generate high-quality images in tens of seconds or minutes, depending on the size of the dataset. Algorithms for ray tracing geometrically defined scenes have been developed for a number of parallel machine architectures including the Connection Machine [Delaney88] and the MPP [Dorband88], but none of these implementations support the display of volume data. A parallelizable object-order volume rendering algorithm is presented in [Westover89], but no implementation achieving real-time update rates has yet been attempted.

In [Levoy89c], a design is presented for a workstation capable of rendering arbitrary mixtures of analytically defined geometry and sampled scalar fields of three spatial dimensions in real-time or near real-time. The design is based on the algorithms presented in this thesis. Speedups of two additional orders of magnitude are obtained by implementing these algorithms on Pixel-Planes 5, a massively parallel raster display engine incorporating custom logic-enhanced memory chips [Fuchs89b] currently under development at the University of North Carolina and scheduled for completion during the fall of 1989. Although Pixel-Planes 5 was not explicitly designed for volume rendering, its flexibility makes it surprisingly well suited to the task. According to preliminary estimates, the proposed workstation will provide update rates of between 1 and 20 frames per second (depending on the desired level of refinement) for datasets of useful size and complexity.

6.3. User interface design issues

User interfaces for existing volume rendering systems are constrained by the inability to generate images in real-time. Feedback during selection of rendering parameters is usually provided by *meta-visualizations* such as 2D plots of color and opacity versus input value, wire-frame representations of viewing frustums and motion paths, etc. These ancillary displays complicate the user interface and alienate prospective users. Teaming a computer technician with each user is not a satisfactory alternative, particularly in the medical field. Such intermediaries inhibit the frequent and informal experimentation that leads to insight.

If the workstation proposed in [Levoy89c] operates in real-time or near real-time as expected, these meta-visualizations can be omitted or relegated to a supporting role. Sequences of volume rendered images would serve as feedback to the user of changes made in rendering parameters. Ideally, the user interface should be trivial. As an example, radiological viewing stations generally provide only one interactive control device - a trackball having two degrees of freedom; one axis controls the the position of a density window, and the second axis controls its width. Medical professionals who have used the volume rendering algorithms described in this thesis suggest that in order for volume rendering workstations to gain acceptance in clinical settings, they should have no more than two or at most three such controls.

As an example of an interface tool that meets this criteria for simplicity, let us imagine that the user is provided with two six-degree-of-freedom input devices such as the Polhemus Navigation Sciences's 3SPACE tracker. One of these *bats* (3D plural of mouse, term suggested in [Ware88]) controls the position and orientation of a 3D region of interest, and the second bat controls the position and direction of a point light source. Using the workstation proposed in [Levoy89c], voxels inside (or outside) the region of interest could be highlighted by scaling voxel opacities as described in section 2.4 and illustrated by figures 2.13 and 2.14. User feedback would be provided by computing 20 crude volume rendered images per second on the proposed workstation.

One obvious addition to the above paradigm is interactive control over observer position. Both hands are already in use, but head position and orientation are still available as inputs to the system. The position and orientation information returned by a third Polhemus tracker affixed to a head-mounted display system would allow a user to move around and through a 3D scene as if

it were actually in the room with them. Since Pixel-Planes 5 is not expected to be capable of rendering a moving $256 \times 256 \times 256$ voxel dataset at 30 frames per second as required for a head-mounted display system, either the size of the dataset must be reduced, or additional speed-ups must be obtained through hardware or software improvements.

6.4. Visualization of multiple fields

An important problem not addressed in this thesis is how to visualize vector or multiple scalar fields. The use of volumetric compositing rather than a binary voxel representation greatly expands the variety of physical phenomena that can be accurately simulated. It is theoretically feasible to display surfaces having matte or reflective finishes, objects that are partially translucent, not merely partially transparent (the former passes light but diffuses it, making objects appear indistinct), and so on. In [Robertson85], realistic shading models are effectively used to visualize multiple 2D scalar fields by assigning one dataset to surface relief and another to surface albedo or color. Aside from the efforts of Hoehne [Hoehne87, Hoehne88a] in the medical field, little work has been done applying these techniques to three dimensions. One obvious approach would be to use the second dataset to modulate the color or opacity computed for voxels in the first dataset. A more sophisticated method would be to use the second dataset to perturb the normals, shininess, or other properties of the surfaces displayed from the first dataset. One might also use the second dataset to modulate parameters of a solid texture applied to the first dataset during rendering.

6.5. How correct is a volume rendering?

Current perspective suggests that the applicability of volume rendering to certain disciplines - diagnostic radiology in particular - hinges on answering the question: how correct is a volume rendering?

In many applications, the data to be visualized has no visible manifestation in nature. Internal anatomic surfaces are visible during surgery, but seldom in their entirety and seldom under the lighting and viewing conditions simulated in volume rendering algorithms. If we treat volume rendering as abstract visualization, adherence to a consistent physical model is not necessary. On the other hand, the human perceptual system expects sensory input to arise from physically plausible phenomena and forms interpretations on that basis. To insure an unambiguous interpretation, we should adhere to a plausible physical model.

The physical model underlying the shading and visibility calculations described in section 2.2 is a colored semi-transparent gel in which suspended reflective particles align to form the appearance of embedded surfaces. In [Sabella88], identical calculations are justified by a model consisting of varying density emitters. Neither model is particularly intuitive. Who has ever seen a gel containing aligned reflective particles or a 3D array of pinpoint light sources? The question therefore arises of how valid and useful our interpretation of these visualizations is.

A closely related issue is the concern raised in section 2.5 that surfaces appearing in volume rendered images are not renditions of surfaces present in the original scene, but are instead renditions of fuzzy surfaces as they exist in a bandlimited representation of the scene. Fuzzy surfaces having no discernible texture do not occur in daily life. Moreover, volume rendered surfaces do not look fuzzy; they look precise. The danger therefore exists of interpreting these visualizations incorrectly. In particular, features that appear to lie on anatomical surfaces may be superficial or may be embedded and are made visible by a partially transparent rendering of the overlying material. Along similar lines, volumetric compositing often causes surface silhouette edges to appear unnaturally sharp and misplaced by some fraction of a voxel from their true location. Diagnostic radiology requires making subtle judgements of feature size. The potential exists of making these judgements incorrectly.

One possible solution is to provide alongside each diagnostic image a calibration image containing volume renderings of 3D scan-converted geometrically defined objects and surfaces. Objects would be labeled as to density, shape, size, and orientation, and the image would be generated using rendering parameters identical to those used to generate the diagnostic image. Clinicians could then make quantitatively accurate judgements from the diagnostic image by visual comparison to the calibration image. With increasing experience, the clinician could be expected to make such judgements without the accompanying reference image. An alternative solution would be to fit geometric primitives to the data and automatically quantitate anatomic features from the geometric representation. We have eliminated fitting of geometric primitives as unreliable for presentation of entire datasets, but it might work well locally if guided and verified by volume renderings generated directly from the sample data.

REFERENCES

- [Amantides84] Amantides, J., "Ray Tracing with Cones," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 129-135.
- [Amantides87] Amantides, J. and Woo, A., "A Fast Voxel Traversal Algorithm for Ray Tracing," *Proc. Eurographics '87*, ed. G. Marechal, Elsevier Science Publishers B.V., North-Holland, 1987, pp. 3-10.
- [Arvo87] Arvo, J. and Kirk, D., "Fast Ray Tracing by Ray Classification," *Computer Graphics*, Vol. 21, No. 4, July, 1987, pp. 55-64.
- [Arvo88] Arvo, J. and Kirk, D., "A Survey of Ray Tracing Acceleration Techniques," *ACM SIGGRAPH '88 course notes*, Vol. 7, August, 1988.
- [Badt88] Badt, S.Jr., "Two Algorithms for Taking Advantage of Temporal Coherence in Ray Tracing," *The Visual Computer*, Vol. 4, 1988, pp. 123-132.
- [Bergman86] Bergman, L., Fuchs, H., Grant, E., and Spach, S., "Image Rendering by Adaptive Refinement," *Computer Graphics*, Vol. 20, No. 4, August, 1986, pp. 29-37.
- [Blinn82] Blinn, J.F., "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces," *Computer Graphics*, Vol. 16, No. 3, July, 1982, pp. 21-29.
- [Catmull74] Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Dissertation, University of Utah, Salt Lake City, 1974.
- [Catmull80] Catmull, E. and Smith, A.R., "3-D Transformations of Images in Scanline Order," *Computer Graphics*, Vol. 14, No. 3, July, 1980, pp. 279-285.
- [Chen85] Chen, L., Herman, G.T., Reynolds, R.A., and Udupa, J.K., "Surface Shading in the Cuberille Environment," *IEEE Computer Graphics and Applications*, Vol. 5, No. 12, December, 1985, pp. 26-32.
- [Cleary88] Cleary, J.G., and Wyvill, G., "Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision," *The Visual Computer*, Vol. 4, 1988, pp. 65-83.
- [Cline88] Cline, H.E., Lorensen, W.E., Ludke, S., Crawford, C.R., and Teeter, B.C., "Two Algorithms for the Three-Dimensional Reconstruction of Tomograms," *Medical Physics*, Vol. 15, No. 3, May/June, 1988, pp. 320-327.
- [Cook84] Cook, R.L., "Shade Trees," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 223-231.
- [Cook86] Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics*, Vol. 5, No. 1, January, 1986, pp. 51-72.
- [Csuri79] Csuri, C., Hackathron, R., Parent, R., Carlson, W. and Howard, M., "Towards an Interactive High Visual Complexity Animation System," *Computer Graphics*, Vol. 13, No. 2, August, 1979, pp. 289-299.

- [Delaney88] Delaney, H.C., "Ray Tracing on a Connection Machine," *Proc. 1988 ACM/INRIA International Conference on Supercomputing*, St. Malo, France, July, 1988, ACM Press, pp. 659-667.
- [Dippe84] Dippe, M. and Swensen, J., "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 149-158.
- [Dippe85] Dippe, M.A.Z. and Wold, E.H., "Antialiasing Through Stochastic Sampling," *Computer Graphics*, Vol. 19, No. 3, July, 1985, pp. 69-78.
- [Dorband88] Dorband, J.E., "3D Graphic Generation on the MPP," *Proc. 1987 ICS International Conference on Supercomputing*, International Supercomputing Institute, 1988, Vol. 1, pp. 305-309.
- [Drebin88] Drebin, R.A., Carpenter, L., and Hanrahan, P., "Volume Rendering," *Computer Graphics*, Vol. 22, No. 4, August 1988, pp. 65-74.
- [Duff85] Duff, Tom, "Compositing 3-D Rendered Images," *Computer Graphics*, Vol. 19, No. 3, July, 1985, pp. 41-44.
- [Feibush80] Feibush, E., Levoy, M., and Cook, R., "Synthetic Texturing using Digital Filters," *Computer Graphics*, Vol. 14, No. 3, July, 1980, pp. 294-301.
- [Fishman87] Fishman, E.K., Drebin, B., Magid, D., Scott, W.W., Jr., Ney, D.R., Brooker, A.F., Jr., Riley, L.H., Jr., St. Ville, J.A., Zerhouni, E.A., and Siegalman, S.S., "Volumetric Rendering Techniques: Applications for Three-Dimensional Imaging of the Hip," *Radiology*, Vol. 163, 1987, pp. 737-738.
- [Frieder85] Frieder, G., Gordon, D., and Reynolds, R.A., "Back-to-Front Display of Voxel-Based Objects," *IEEE Computer Graphics and Applications*, Vol. 5, No. 1, January, 1985, pp. 52-59.
- [Fuchs77] Fuchs, H., Kedem, Z.M. and Uselton, S.P., "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, Vol. 20, No. 10, 1977, pp. 693-702.
- [Fuchs89a] Fuchs, H., Levoy, M., Pizer, S., and Rosenman, J., "Interactive Visualization and Manipulation of 3D Medical Image Data," *Proc. NCGA '89*, Philadelphia, PA, April, 1989, Vol. 1, pp. 118-131.
- [Fuchs89b] Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., Tebbs, B., and Israel, L., "A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics*, Summer, 1989 (to appear).
- [Fuchs89c] Fuchs, H., Levoy, M., and Pizer, S., "Interactive Visualization of 3D Medical Data," *IEEE Computer*, August, 1989 (to appear).
- [Fujimoto86] Fujimoto, A., Tanaka, T., and Iwata, K., "ARTS: Accelerated Ray-Tracing System," *IEEE Computer Graphics and Applications*, Vol. 6, No. 4, April, 1986, pp. 16-26.
- [Gargantini82] Gargantini, I., "Linear Octrees for Fast Processing of Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 20, 1982, pp. 365-374.

- [Gargantini86] Gargantini, I., Walsh, T.R.S., and Wu, O.L., "Displaying a Voxel-Based Object via Linear Octrees," *Proc. SPIE*, Vol. 626, 1986, pp. 460-466.
- [Gauch88] Gauch, J., "Image Description via the Multiresolution Intensity Axis of Symmetry," *Proc. ICCV*, IEEE Catalogue #88CH2664-1, 1988, pp. 269-274.
- [Glassner84] Glassner, A.S., "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, Vol. 4, No. 10, October, 1984, pp. 15-22.
- [Glusker85] Glusker, P.J. and Trueblood, K.N., *Crystal Structure Analysis*, Oxford University Press, 1985.
- [Goldwasser85] Goldwasser, S.M., Reynolds, R.A., Bapty, T., Baraff, D., Summers, J., Talton, D.A., and Walsh, E., "Physician's Workstation with Real-Time Performance," *IEEE Computer Graphics and Applications*, Vol. 5, No. 12, December, 1985, pp. 44-57.
- [Goldwasser86a] Goldwasser, S., Reynolds, R.A., Talton, D., and Walsh, E., "Interactive 3-D Workstations for Biomedical Applications," *Proc. NCGA '86*, Anaheim, CA, May, 1986, pp. 61-70.
- [Goldwasser86b] Goldwasser, S., "Rapid Techniques for the Display and Manipulation of 3-D Biomedical Data," *NCGA '87 Tutorial*, Anaheim, CA, May, 1986.
- [Goodsell88] Goodsell, D.S., Mian, S., and Olson, A.J., "Rendering of Volumetric Data in Molecular Systems," Manuscript, 1988.
- [Gordon85] Gordon, D. and Reynolds, R.A., "Image Space Shading of 3-Dimensional Objects," *Computer Vision, Graphics and Image Processing*, Vol. 29, 1985, pp. 361-376.
- [Haines86] Haines, E.A. and Greenberg, D.P., "The Light Buffer: A Shadow-Testing Accelerator," *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, September, 1986, pp. 6-16.
- [Heckbert84] Heckbert, P.S. and Hanrahan, P., "Beam Tracing Polygonal Objects," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 119-127.
- [Heckbert86] Heckbert, P., "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, Vol. 6, No. 11, November, 1986, pp. 56-67.
- [Herman79] Herman, G.T. and Liu, H.K., "Three-Dimensional Display of Human Organs from Computer Tomograms," *Computer Graphics and Image Processing*, Vol. 9, No. 1, January, 1979, pp. 1-21.
- [Herman80] Herman, G.T., *Image Reconstruction from Projections*, Academic Press, New York, 1980.
- [Herman82] Herman, G.T., Reynolds, R.A., and Udupa, J.K., "Computer Techniques for the Representation of Three-Dimensional Data on a Two-Dimensional Display," *Proc. SPIE*, Vol. 367, 1982, pp. 3-14.
- [Hoehne86] Hoehne, K.H. and Bernstein, R., "Shading 3D-Images from CT Using Gray-Level Gradients," *IEEE Transactions on Medical Imaging*, Vol. MI-5, No. 1, March, 1986, pp. 45-47.

- [Hoehne87] Hoehne, K. H., Riemer, M. and Tiede, U., "Viewing Operations for 3D - Tomographic Gray Level Data." *Proc. CAR '87*, ed. H. Lemke et al., Springer, Berlin, 1987.
- [Hoehne88a] Hoehne, K.-H., Bomans, M., Tiede, U., and Riemer, M., "Display of Multiple 3D-Objects Using the Generalized Voxel-Model," *Proc. SPIE*, Vol. 914, 1988, pp. 850-854.
- [Hoehne88b] Hoehne, K.-H., personal communication, February, 1988.
- [Johns83] Johns, H.E. and Cunningham, J.R., *The Physics of Radiology*, Charles C. Thomas, 1983.
- [Kajiya83] Kajiya, J.T., "New Techniques for Ray Tracing Procedurally Defined Objects," *Computer Graphics*, Vol. 17, No. 3, July, 1983, pp. 91-102.
- [Kajiya84] Kajiya, J.T. "Ray Tracing Volume Densities," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 165-174.
- [Kajiya86] Kajiya, J.T., "The Rendering Equation," *Computer Graphics*, Vol. 20, No. 4, August, 1986, pp. 143-150.
- [Kaufman86a] Kaufman, A., "Voxel-Based Architectures for Three-Dimensional Graphics," *Proceedings of the IFIP 10th World Computer Congress*, Dublin, Ireland, September, 1986, pp. 361-366.
- [Kaufman86b] Kaufman, A. and Shimony, E., "3D Scan-Conversion Algorithms for Voxel-Based Graphics," *Proc. ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October, 1986, pp. 45-75.
- [Kaufman87a] Kaufman, A., "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *Computer Graphics*, Vol. 21, No. 4, July, 1987, pp. 171-179.
- [Kaufman87b] Kaufman, A., "An Algorithm for 3D Scan-Conversion of Polygons," *Proc. EUROGRAPHICS '87*, Amsterdam, Netherlands, August, 1987, pp. 197-208.
- [Kaufman88a] Kaufman, A. and Bakalash, R., "Memory and Processing Architecture for 3D Voxel-Based Imagery," *IEEE Computer Graphics and Applications*, Vol. 8, No. 6, November, 1988, pp. 10-23.
- [Kaufman88b] Kaufman, A., personal communication, December, 1988.
- [Kay86] Kay, T.L. and Kajiya, J.T., "Ray Tracing Complex Scenes," *Computer Graphics*, Vol. 20, No. 4, August, 1986, pp. 269-278.
- [Lee85] Lee, M.E., Redner, R.A., and Uselton, S.P., "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics*, Vol. 19, No. 3, July, 1985, pp. 61-67.
- [Levinthal84] Levinthal, A. and Porter, T., "Chap - A SIMD Graphics Processor," *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp. 77-82.
- [Levoy78] Levoy, M., "Computer-Assisted Cartoon Animation," Master's thesis, Department of Architecture, Cornell University, Ithaca, New York, 1978.

- [Levoy85] Levoy, M. and Whitted, T., "The Use of Points as a Display Primitive," Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, January, 1985.
- [Levoy87] Levoy, M., "Rendering of Surfaces from Volumetric Data," Technical Report 87-016, Computer Science Department, University of North Carolina at Chapel Hill, June, 1987.
- [Levoy88a] Levoy, M., "Direct visualization of Surfaces from Computed Tomography Data," *Proc. SPIE*, Vol. 914, 1988, pp. 828-841.
- [Levoy88b] Levoy, M., "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May, 1988, pp. 29-37.
- [Levoy88c] Levoy, M., "Efficient Ray Tracing of Volume Data," Technical Report 88-029, Computer Science Department, University of North Carolina at Chapel Hill, June, 1988.
- [Levoy88d] Levoy, M., "Volume Rendering by Adaptive Refinement," Technical Report 88-030, Computer Science Department, University of North Carolina at Chapel Hill, June, 1988.
- [Levoy88e] Levoy, M., "Rendering Mixtures of Geometric and Volumetric Data," Technical Report 88-052, Computer Science Department, University of North Carolina at Chapel Hill, December, 1988.
- [Levoy89a] Levoy, M., "Design for a Real-Time High-Quality Volume Rendering Workstation," Technical Report 89-010, Computer Science Department, University of North Carolina at Chapel Hill, February, 1989.
- [Levoy89b] Levoy, M., Letter to the editor of *IEEE Computer Graphics and Applications*, Vol. 9, No. 2, March, 1989, p. 91.
- [Levoy89c] Levoy, M., "Design for a Real-Time High-Quality Volume Rendering Workstation," *Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, May, 1989 (to appear).
- [Levoy89d] Levoy, M., "Volume Rendering by Adaptive Refinement," *The Visual Computer*, Vol. 5, No. 3, June, 1989 (to appear).
- [Lorenzen87] Lorenzen, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4, July, 1987, pp. 163-169.
- [Max86] Max, N., "Atmospheric Illumination and Shadows," *Computer Graphics*, Vol. 20, No. 4, August, 1986, pp. 117-124.
- [Meagher82] Meagher, D., "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, Vol. 19, 1982, pp. 129-147.
- [Mosher88] Mosher, C., personal communication, December, 1988.
- [Peachey85] Peachey, D.R., "Solid Texturing of Complex Surfaces," *Computer Graphics*, Vol. 19, No. 3, July, 1985, pp. 279-286.

- [Phong75] Bui-Tuong, Phong, "Illumination for Computer-Generated Pictures," *Communications of the ACM*. Vol. 18, No. 6, June, 1975, pp. 311-317.
- [Pizer86] Pizer, S.M., Fuchs, H., Mosher, C., Lifshitz, L., Abram, G.D., Ramanathan, S., Whitney, B.T., Rosenman, J.G., Staab, E.V., Chaney, E.L. and Sherouse, G., "3-D Shaded Graphics in Radiotherapy and Diagnostic Imaging," *Proc. NCGA '86*, Anaheim, CA, May, 1986, pp. 107-113.
- [Porter84] Porter, T. and Duff, T., "Compositing Digital Images," *Computer Graphics*. Vol. 18, No. 3, July, 1984, pp. 253-259.
- [Purvis86] Purvis, George D. and Culberson, Chris, "On the Graphical Display of Molecular Electrostatic Force-Fields and Gradients of the Electron Density," *Journal of Molecular Graphics*. Vol. 4, No. 2, June, 1986, pp. 89-92.
- [Reeves83] Reeves, W.T., "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects," *Computer Graphics*. Vol. 17, No. 3, July, 1983, pp. 359-376.
- [Reeves85] Reeves, W.T., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," *Computer Graphics*. Vol. 19, No. 3, July, 1985, pp. 313-322.
- [Reeves87] Reeves, W.T., Salesin, D.H., and Cook, R.L., "Rendering Antialiased Shadows with Depth Maps," *Computer Graphics*. Vol. 21, No. 4, July, 1987, pp. 283-291.
- [Reynolds89] Reynolds, A., Letter to the editor of *IEEE Computer Graphics and Applications*. Vol. 9, No. 2, March, 1989, pp. 90-91.
- [Robertson85] Robertson, P.K. and O'Callaghan, J.F., "The Application of Scene Synthesis Techniques to the Display of Multidimensional Image Data," *ACM Transactions on Graphics*. Vol. 4, No. 4, October, 1985, pp. 247-275.
- [Rubin80] Rubin, S.M. and Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics*. Vol. 14, No. 3, July 1980, pp. 110-116.
- [Sabella88] Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics*. Vol. 22, No. 4, August 1988, pp. 51-58.
- [Schlusselberg86] Schlusselberg, D.S. and Smith, W.K., "Three-Dimensional Display of Medical Image Volumes," *Proc. NCGA '86*, Anaheim, CA, May, 1986, Vol. III, pp. 114-123.
- [Scott87] Scott, W.W.Jr., Fishman, E.K., and Magid, D., "Acetabular Fractures: Optimal Imaging," *Radiology*. Vol. 165, 1987, pp. 537-539.
- [Shinya87] Shinya, M., Takahashi, T., and Naito, S., "Principles and Applications of Pencil Tracing," *Computer Graphics*. Vol. 21, No. 4, July, 1987, pp. 45-54.
- [Smith87] Smith, A.R., "Volume graphics and Volume Visualization: A Tutorial," Technical Memo 176, PIXAR Inc., San Rafael, California, May, 1987.
- [Troussel87] Troussel, Y. and Schmitt, F., "Active-Ray Tracing for 3D Medical Imaging," *Proc. EUROGRAPHICS '87*, pp. 139-149.

- [Upson86] Upson, C., "The Visual Simulation of Amorphous Phenomena," *The Visual Computer*, Vol. 2, 1986, pp. 321-326.
- [Upson88] Upson, C. and Keeler, M., "VBUFFER: Visible Volume Rendering," *Computer Graphics*, Vol. 22, No. 4, August 1988, pp. 59-64.
- [Wallace81] Wallace, B.A., "Merging and Transformation of Raster Images for Cartoon Animation," *Computer Graphics*, Vol. 15, No. 3, August, 1981, pp. 253-262.
- [Ware88] Ware, C. and Jessome, D.R., "Using the Bat: A Six-Dimensional Mouse for Object Placement," *IEEE Computer Graphics and Applications*, Vol. 8, No. 6, November, 1988, pp. 65-70.
- [Westover89] Westover, L., "Interactive Volume Rendering," *Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, May, 1989 (to appear).
- [Whitted80] Whitted, T., "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, Vol. 23., No. 6, June, 1980, pp. 343-349.
- [Williams78] Williams, L., "Casting Curved Shadows on Curved Surfaces," *Computer Graphics*, Vol. 12, No. 3, August 1978, pp. 270-274.
- [Williams82] Williams, T.V., *A Man-Machine Interface for Interpreting Electron Density Maps*. Dissertation, University of North Carolina, Chapel Hill, NC, 1982.
- [Yau83] Yau, M., and Srihari, S.N., "A Hierarchical Data Structure for Multidimensional Digital Images," *Communications of the ACM*, Vol. 26., No. 7, July, 1983, pp. 504-515.