Towards Accurate and Efficient Volume Rendering

Kevin Lawrence Novins
Ph.D Thesis

TR 93-1395
October 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

TOWARDS ACCURATE AND EFFICIENT VOLUME RENDERING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Kevin Lawrence Novins

January 1994

© Kevin Lawrence Novins 1994

ALL RIGHTS RESERVED

TOWARDS ACCURATE AND EFFICIENT

VOLUME RENDERING

Kevin Lawrence Novins, Ph.D.

Cornell University 1994

This thesis is concerned with improvements to algorithms for volume rendering; a technique that provides scientists with the means for visual exploration of three-dimensional data. Despite its numerous successes, and its increasing use within the scientific community, state-of-the-art volume rendering algorithms have many shortcomings. Difficulties include: ensuring the accuracy of the rendered images, producing images with modest computational resources, and rendering the diverse types of data that are currently being produced.

The work in this thesis was motivated by the demands of an ongoing visualization project in four-dimensional cardiac visualization. We present solutions to some key problems in ensuring accuracy and in producing algorithms that can scale to handle large datasets. Although the theoretical work in this thesis applies to arbitrary data topologies, our implementations have assumed that the data is defined by sample points on a regular rectilinear grid.

In the area of accuracy, we focus on the error that is introduced during volume

projection. This phase of the volume rendering process involves the evaluation of the emission-absorption volume rendering line integral. This thesis presents four techniques for *controlled precision volume integration.* These schema depart from existing approaches in that they provide error bounds along with the solutions they generate. In each case, the error analysis leads to an algorithm for evaluating the integral to any specified tolerance.

Our investigations into efficiency issues have resulted in two advances. First, an *adaptive error bracketing* scheme is presented that builds on the controlled precision volume integration methods. Using adaptive error bracketing, the solution for a viewing ray is continually refined until a user-specified error tolerance is met. The algorithm allows processing of the data without imposing a strict front-to-back or back-to-front evaluation order. Second, a suite of tools are presented that can be used to efficiently compute perspective projections of volume data. These include a paging strategy that is useful when a dataset is too large to fit into RAM memory and a *ray splitting* technique for adaptive supersampling. The latter technique ensures that all data features contribute to the final image while avoiding overcomputation in regions close to the eyepoint.

BIOGRAPHICAL SKETCH

Kevin L. Novins was born on December 2, 1963. He attended public schools in Westchester County, New York and graduated from Ardsley High School in 1981. He studied computer science at Harvard University, and received the Bachelor of Arts degree in 1985. After graduation from college he performed research in medical imaging at the North Carolina Memorial Hospital. He entered Cornell University in the fall of 1987 and received the Master of Science degree in 1990 and the Doctor of Philosophy degree in 1994.

ACKNOWLEDGEMENTS

First, I would like to thank the Chair of my thesis committee, Donald P. Greenberg, for making this work possible. He helped me to set my goals and worked to establish and maintain the research collaborations that were necessary for carrying them through. I appreciated his guidance, enthusiasm, dedication to quality, and unwavering support. I also appreciated his teaching me how to present my research properly. I am glad for all the times that we worked closely together. Don has created a wonderful environment at the Program of Computer Graphics and I am grateful that he gave me the opportunity to work and study there.

Much of this research could not have been performed without the help of Jim Arvo, who was an advisor and a friend throughout the process. It was Jim who first stimulated my interest in performing an error analysis of volume rendering. He assisted in high-level planning, in working out implementation issues, and in settling some of the final production details. Jim also helped me to sort out my thoughts when I got frustrated, and provided encouragement during research slowdowns. I am grateful to Jim for his contributions to this thesis and for all that he has taught me.

Another friend and advisor was François Sillion. François was instrumental in teaching me how to perform graduate-level research, and helped to get me started on this dissertation. I would like to thank him for helping me to become more of a scientist and less of a hacker.

I would also like to thank David Salesin, who was a colleague in developing the adaptive error bracketing algorithm. His enthusiasm was key to the success of that project, and he helped a great deal to move me along towards graduation.

The collaborative research project in cardiac visualization that I discuss in Chapter 6 was the impetus for this research. It was a pleasure to work with Dr. Richard Devereux of the Cornell Medical Center, Dr. Riccardo Pini of the University of Firenze, and Steven Leavitt of Hewlett-Packard. They convinced me of the importance of the project and provided me with the education that I needed in the medical and engineering aspects of ultrasonic imaging.

I would like to thank Daniel Huttenlocher and Anthony Reeves for serving on my thesis committee.

Several of the results presented here have been published in the form of research papers. I am grateful to my co-authors for their contributions, particularly since sections of text and figures have been incorporated into this thesis. Specifically, I would like to thank: François Sillion and Donald Greenberg, my co-authors for "An Efficient Method for Volume Rendering Using Perspective Projection" [NSG90]; Jim Arvo, my co-author for "Controlled-Precision Volume Rendering" [NA92]; and Jim Arvo and David Salesin, my co-authors for "Adaptive Error Bracketing for Controlled Precision Volume Rendering" [NAS92].

Program of Computer Graphics staff members provided a great deal of support for me and for this research. Hurf Sheldon was particularly helpful in ensuring that I always worked in the best possible environment. Jim Ferwerda generously provided invaluable assistance with many aspects, including the preparation of slides and videotapes. I would also like to thank Ellen French, Fran Brown, Janet Brown, and Emil Ghinger for their help.

The CT pelvis dataset used in several of the figures was provided by the Department of Radiation Oncology at the North Carolina Memorial Hospital. The cardiac dataset of Figures 2.7 and 6.8 were provided by the Cornell Medical Center. The electron density dataset of Figure 5.26 was provided by Duncan McRee of the Scripps Clinic and obtained as part of the Chapel Hill volume rendering test dataset. The photograph in Figure 3.5 of Andrew James Monks was provided by Michael Monks and Marietta Christie.

This work was partially supported by the NSF grants "Interactive Computer Graphics Input and Display Techniques" (CCR-8617880) and "Visualization for Supercomputing: A Graphics Workstation Approach" (AC-8715478). It was also supported by the NSF/DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219). The work was performed on equipment donated by the Hewlett-Packard Corporation.

I would like to thank my friends and lab-mates for sharing wonderful times, and for providing essential support during some not-so-wonderful times.

Finally, I would like to thank my parents, Rochelle and Jay Novins, and my brother and sister, Doug and Kara, for their constant love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Over the past ten years, great advances have been made in scanning techniques that produce volume data, such as computed tomography, magnetic resonance imaging and ultrasonic imaging. These developments have brought large quantities of data into the hands of physicians and scientists, much of which can only be interpreted visually. This thesis is concerned with providing tools for the visual exploration of their data. To be useful, the tools must be intuitive, provide accurate renditions, and present the results in a timely fashion with modest computational resources.

The work presented in this thesis relates to a subfield of scientific visualization called volume rendering. Volume rendering algorithms can be distinguished from other approaches in that images are created directly from a three-dimensional array of raw data, without fitting a surface as an intermediate step. For the display of data that is not surface-oriented, such as the output of a tomographic scanner or the result of a fluid simulation of gaseous phenomena, volume visualization has been the method of choice. It has been successfully applied to data from medicine, chemistry, astronomy, meteorology, and many other disciplines.

Despite these numerous successes, and its increasing use within the scientific community, state-of-the art volume rendering algorithms have many shortcomings as a visualization tool. Developments are needed in the following areas:

Accuracy. Current algorithms employ many approximations for which the impact on the final image is not known. Error is introduced during data acquisition, reconstruction from point samples, projection onto screen-space, and display. Volume rendering algorithms cannot currently be relied upon in fields where high accuracy is essential.

Efficiency. Volume rendering requires a great deal of calculation, disk storage and random-access memory. The algorithms become impractical when used with high resolution dynamic datasets. Efficiency improvements are essential for ensuring that visualization can be performed outside of special-purpose computing centers. Ideally, algorithms should be fast enough and computing resources powerful enough to provide scientists with low-cost tools for interactive manipulation of the images.

Arbitrary data formats. Volume rendering algorithms are currently inappropriate for a large body of scientific data. The technique has only gained acceptance for visualizing scalar-valued three-dimensional fields. Additionally, most algorithms impose the limitation that data samples lie on a regular rectilinear grid.

The above difficulties become clear when applying volume rendering techniques to "real-world" scientific data. The work in this thesis was motivated by problems that arose as part of an ongoing project in cardiac imaging. In collaboration with researchers at the the Cornell Medical Center, ESAOTE Biomedica, and Hewlett-Packard, we are developing techniques for dynamic, three-dimensional imaging of cardiac structures via ultrasound. This method has great potential since it is both non-invasive and low-cost, making it feasible for diagnosis in a doctor's office. The project places several new demands on existing volume rendering methods. Since the application is medical, we need to rely on the images for accurate diagnosis. Since time effectively adds a fourth dimension to the already large dataset, the storage demands are excessive, particularly for real-time playback. Finally, the data sample points do not lie on a regular rectilinear grid.

For this thesis, we concentrated on the issues of accuracy and efficiency as they relate to volume rendering. While our solutions are not a panacea, they do provide important advances in each of these areas. Although the theoretical work in this thesis applies to arbitrary data topologies, the implementations have assumed that the data is defined by sample points on a rectilinear grid.

In the area of accuracy, we focus on the error that is introduced during volume projection. This phase of the volume rendering process is concerned with the evaluation of the emission-absorption volume rendering integral. We present techniques for evaluating this integral to any specified precision. Tools for computing the integral along the intersection of a viewing ray with a voxel are extended to consider the intersection of a ray with multiple voxels. Having solved the latter problem, we can compute point samples of the illumination function across the image plane to any specified precision. Our approaches represent a departure from existing volume rendering algorithms in that they provide quantitative assessment and allow the user control over the quality of the solutions.

Our investigations into efficiency have resulted in two advances. The first is an adaptive error bracketing scheme that builds on the controlled-precision volume projection methods. Using adaptive error bracketing, the solution for a single viewing ray is continually refined until a user-specified error tolerance is met. This avoids overcomputation. For the first time, this algorithm allows processing of the data without imposing a strict front-to-back or back-to-front evaluation order. Our studies have shown that adaptive error bracketing saves computational effort when working with smooth datasets or when computing low-accuracy images.

We also present efficient volume rendering algorithms for computing perspective projections. Perspective projections are desirable because they provide a depth cue that is easy to comprehend. Previous research in volume rendering has concentrated on the less intuitive parallel projection model since it has been both easier to compute and provides a regular sampling of the scene. Our results address both these problems. We present an optimal paging strategy that is useful when computing perspective projections of datasets that are too large to fit into RAM memory. An adaptive supersampling solution to the problem of perspective ray divergence is also presented. This ensures that all features in the back of the dataset are accounted for, while avoiding overcomputation in regions towards the front of the dataset.

The organization of the thesis is as follows. Chapter 2 presents a review of theoretical models for volume illumination. This includes a discussion of the emission-absorption model, upon which the work in this thesis is based. Chapter 3 contains an overview of the volume rendering process and describes existing algorithms. Chapter 4 presents our results in approximating the volume projection problem to any specified precision. Chapter 5 is devoted to efficient computation of volume-rendered images. It includes techniques for perspective projection and the adaptive error bracketing algorithm for controlled-precision volume projection. In Chapter 6 we present the results-to-date of our cardiac imaging project.

The thesis concludes in Chapter 7 with a summary and a discussion of future research directions.

Chapter 2

Volume Illumination Models

The process of rendering is essentially a simulation of light interacting with an environment. In rendering for visualization, the goal is to communicate scientific information. The computational nature of the simulation affords the programmer the luxury of choosing the rules that will govern the simulation, thus allowing a great deal of control in devising ways to meet this goal. The existence of a set of rules is important, since the abstract nature of the visualization means that errors can only be measured in terms of deviation from the assumed model. This is in contrast to methods for realistic image synthesis where the rules are the laws of physics governing the transfer of light energy, and errors can be measured in terms of deviation from experimental data. In this chapter, we present the *emission-absorption* illumination model that has been widely adopted [Luc92,NH92,VPR$^+$92] and is the basis for the work in this thesis. We also discuss other illumination models that have been used in the context of volume rendering.

The basic illumination problem is depicted in Figure 2.1. Consider a ray of light passing through a medium between points a and b. We would like to know the intensity of light at wavelength λ that is incident on point a in the direction of ray $\vec{ab}$ passing through the material between a and b. We denote this quantity $I_\lambda(a, b)$. Color images are generated by computing $I_\lambda(a, b)$ at several wavelengths. In the following discussion, we assume that $I_\lambda(a, b)$ is computed independently for each wavelength and will drop the λ subscripts for notational convenience.

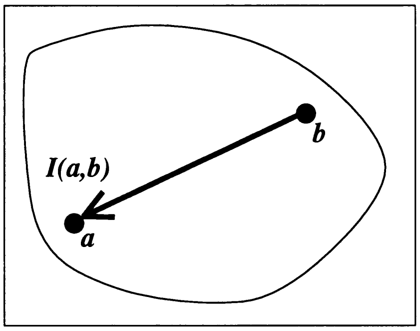


Figure 2.1: The volume illumination problem.

2.1 Emission-Absorption Model

A simple model of volume illumination was originally developed as a model of radiative transfer [Cha60,Sob63]. It is based on a physical model of an emitting and absorbing medium, and does not consider scattering effects. The work in this thesis is based on this model.

The model requires that two nonnegative functions are defined across the volume. The absorption function, $\alpha(x, y, z)$, determines the fraction of light that is absorbed by the medium along a differential ray segment passing through (x, y, z).

It is normally assumed to be wavelength-independent. The emittance function, denoted $\epsilon(x, y, z)$ determines the wavelength-dependent emission of the medium along a differential ray segment passing through (x, y, z).

In the following discussion, the variables t, u, and v will be distance parameters along the ray $\vec{ab}$. The functions α and ϵ will correspondingly be reparameterized by distance along the ray. By fixing b, the endpoint of the ray, we can write the differential equation for $I(t, b)$, the intensity of light at point t along the ray $\vec{ab}$ due to material between t and b:

$$-\frac{d}{dt}I(t,b) = -\alpha(t)\, I(t,b) + \epsilon(t). \tag{2.1}$$

This equation states that the change in intensity, as we move towards the eyepoint, across a differential ray length dt is equal to the light emitted minus the fraction of incoming light that is absorbed. This differential equation can be solved using an integrating factor [RB74]. We sketch the solution process below.

We begin by applying the integrating factor,

$$\kappa(t) = e^{-\int_b^t \alpha(v)\, dv},$$

to both sides of the differential equation (2.1),

$$\frac{d}{dt}I(t,b)\,\kappa(t) - \alpha(t)\,\kappa(t)\,I(t,b) = -\epsilon(t)\,\kappa(t). \tag{2.2}$$

Since,

$$\kappa'(t) = -\alpha(t)\, e^{-\int_b^t \alpha(v)\, dv},$$

we may rewrite Equation (2.2),

$$\frac{d}{dt}I(t,b)\,\kappa(t) + \kappa'(t)\,I(t,b) \;=\; -\epsilon(t)\,\kappa(t)$$

$$\frac{d}{dt}[I(t,b)\,\kappa(t)] = -\epsilon(t)\,\kappa(t).$$

Integrating both sides of the equation from b to t yields,

$$I(t,b)\,\kappa(t) = -\int_b^t \epsilon(u)\,\kappa(u)du + I_b,$$

where I_b is a constant of integration. We continue by isolating $I(t,b)$ on the left-hand side,

$$I(t,b) = -\int_b^t \epsilon(u)\,\kappa(u)du\,\kappa^{-1}(t) + I_b\,k^{-1}(t). \tag{2.3}$$

Since,

$$\kappa^{-1}(t) = e^{-\int_t^b \alpha(v)\,dv},$$

we may continue,

$$\begin{aligned} I(t,b) &= -\int_b^t \epsilon(u)\,e^{-\int_b^u \alpha(v)\,dv}\,e^{-\int_t^b \alpha(v)\,dv}du + I_b\,e^{-\int_t^b \alpha(v)\,dv} \\ &= -\int_b^t \epsilon(u)\,e^{-\int_t^u \alpha(v)\,dv}\,du + I_b\,e^{-\int_t^b \alpha(v)\,dv} \\ &= \int_t^b \epsilon(u)\,e^{-\int_t^u \alpha(v)\,dv}\,du + I_b\,e^{-\int_t^b \alpha(v)\,dv}. \end{aligned} \tag{2.4}$$

If we solve Equation (2.4) for $t = a$, we obtain the expression for $I(a,b)$:

$$I(a,b) = \int_a^b e^{-\int_a^u \alpha(v)\,dv}\,\epsilon(u)\,du + I_b\,e^{-\int_a^b \alpha(v)\,dv} \tag{2.5}$$

The constant I_b is the intensity of backlighting in the ray direction impinging at point b. Equation 2.5 summarizes the volume illumination model used in this thesis. We will normally assume that there is no backlighting. In this case, we may drop the second term and write:

$$\boxed{I(a,b) = \int_a^b e^{-\int_a^u \alpha(v)\,dv}\,\epsilon(u)\,du.} \tag{2.6}$$

The right side of Equation (2.6) shall henceforth be referred to as the *volume rendering integral.* An image computed using the emission-absorption model is given in Figure 2.2.

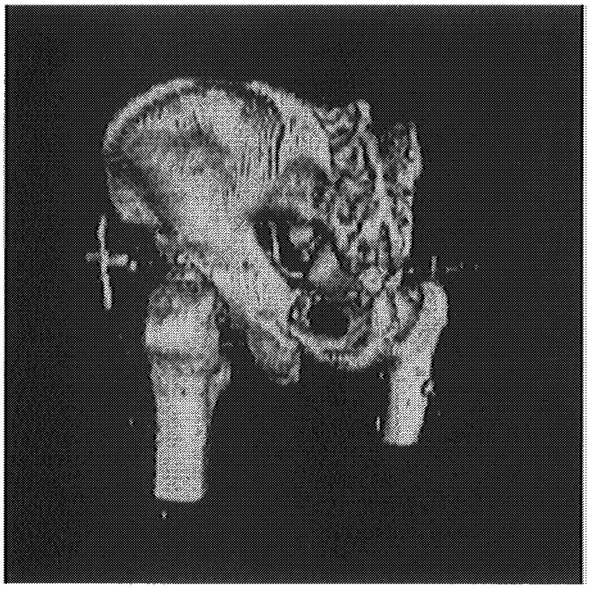


Figure 2.2: Image of the pelvis dataset computed using the emission-absorption model.

2.2 Other Volume Illumination Models

The emission-absorption model is one of many volume illumination models that appear in the literature. Its main attractions are its physical basis, simplicity and its generality. In this section, we review other models in current use and compare their performance with that of the emission-absorption model.

2.2.1 Maximum-Intensity Projection

The simplest projection model for volume rendering is that of *maximum-intensity projection* [Sab88,VPR+92]. Using maximum-intensity projection, the intensity of the ray between points a and b is proportional to the highest density, $\rho(t)$, encountered between points a and b. Thus,

$$I(a,b) = k \max_{t=a}^{b} \rho(t),$$

where k is an arbitrary scaling constant. In this form, maximum-intensity projection produces only gray-scale images. However, color is often added to ameliorate the fact that the technique does not provide any depth information via occlusion. For example, each pixel can be colored based on the distance of the peak value from the eyepoint [Sab88].

Maximum-intensity projection is useful in application domains where it is important to locate extremal values in the data. Such a situation occurs in medical angiography, where the goal is to detect arterial blockage. The blood stream is injected with a contrast agent, thus making it the densest object in the body, and a tomographic scan is performed. By employing maximum-intensity projection in rendering the resulting data, a physician can ensure that the view of the blood stream is not occluded by less dense structures.

This model's major drawback is that it presents a narrow view of the volume; discarding all information except that which involves the most dense material in the dataset. Thus, the technique remains of interest in only a few highly specialized fields, and even in these, it is used as only one part of a suite of visualization modes.

It cannot be considered a truly general visualization technique. An image of the pelvis dataset computed using maximum-intensity projection is given in Figure 2.3.

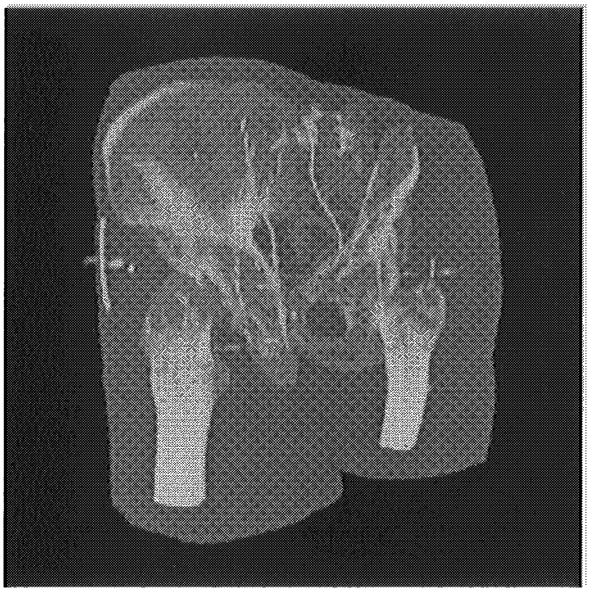


Figure 2.3: Image of pelvis dataset computed using maximum-intensity projection.

2.2.2 Attenuation Projection

Using attenuation projection [Lev92], only the absorption of the media is modeled. We can obtain an expression for $I(a,b)$ by starting with the emission-absorption model of Equation (2.5) and setting the emission coefficient, $\epsilon(u)$, to zero yielding,

$$I(a,b) = I_b \, e^{-\int_a^b \alpha(v)\, dv}. \tag{2.7}$$

Note that in order to use this model, the backlight intensity, I_b must be nonzero. Equation (2.7) shows that using the attenuation projection model, only the sum of the density along the ray is considered; the order in which the material is encountered is irrelevant. As with maximum-intensity projection, there is no occlusion.

The effect of attenuation projection is similar to that of an X-ray. In fact, simulation of X-rays is one important application of this model [SNC90]. An example of attenuation projection is given in Figure 2.4. Because of its lack of shading and depth cues, this model is not completely satisfactory for communicating $3D$ information. Nonetheless, it holds attraction due to its easy evaluation relative to those models that include emissive effects.

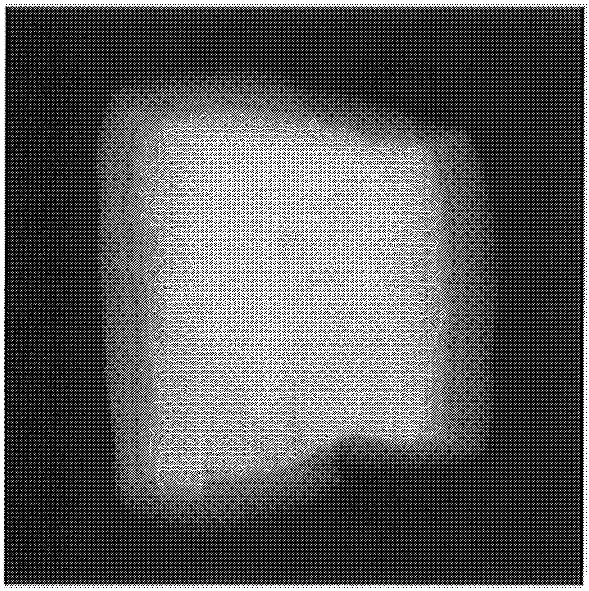


Figure 2.4: Image of the pelvis dataset computed using attenuation projection.

2.2.3 Low-Albedo Models

Another popular class of volume illumination models is based on the concept of the volume media as a varying density field of low-albedo particles that reflect light [Bli82,KV84,Sab88]. Albedo, or reflectivity, is the fraction of light impinging on a particle that is reflected. The fundamental assumption of this class of models is that since particles are of low albedo, all light paths from light source to eyepoint which include more than one reflection off a particle can be ignored. If a particle reflects at most a fraction, Ω, of the incoming light, a light path which bounces off n particles reflects at most Ω^n of the incoming light. Therefore, mathematically, the low-albedo approximation states that

$$\Omega^n \approx 0,\ n \geq 2.$$

When using these models, absorption is typically related to particle density, denoted $\rho(u)$ for any given ray. Emission is determined by computing a one-bounce light reflection from the light source, off the particle, to the eyepoint. The light reflection model varies from simple phase functions that depend only on the angle between the light source and the eyepoint [Bli82,KV84] to more complex models that also consider the orientation of the particles [Lev88,DCH88,Sab88]. The reflectance function is denoted $\Phi(u)$. Choice of Φ will be covered in more detail in Chapter 3. Sometimes, shadowing of the light source by material in the volume is accounted for by a function, $V(u)$, which computes the optical distance from point u along the ray to the light source [KV84].

The low-albedo expression for $I(a,b)$ is

$$I(a,b) = \int_a^b e^{-\int_a^u \rho(v)\,dv}\, V(u)\, \Phi(u)\, \rho(u)\, du.$$

This expression can be adapted for use with more than one primary light source by summing expressions for $I(a,b)$ over all light sources.

The generality of the emission-absorption model allows it to capture the effects of the above low-albedo model, even though the emission-absorption model does not explicitly consider light reflection. If we define,

$$\alpha(u) \equiv \rho(u) \tag{2.8}$$

$$\epsilon(u) \equiv V(u)\Phi_\lambda(u)\,\rho(u), \tag{2.9}$$

we see that the low-albedo model is equivalent to the emission-absorption model. Equation 2.9 effectively moves the calculation of the light reflection into a preprocess that sets the emission coefficients at each point in the volume. Figure 2.5 illustrates how light is reflected off particles using the low-albedo model. Figure 2.6 shows how this is simulated using the emission-absorption model. In this fashion, the effects of any illumination model that allows only a fixed number of reflections in between the light source and the eyepoint may be captured using the emission-absorption model.

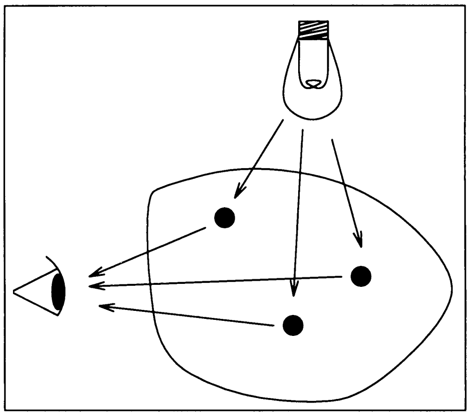
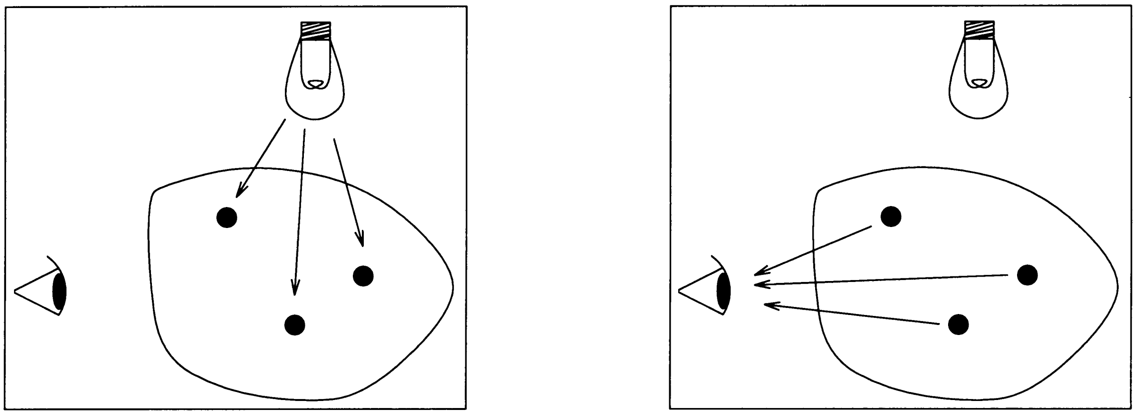


Figure 2.5: The low-albedo methods model a single reflection of light off of the particles in the medium.



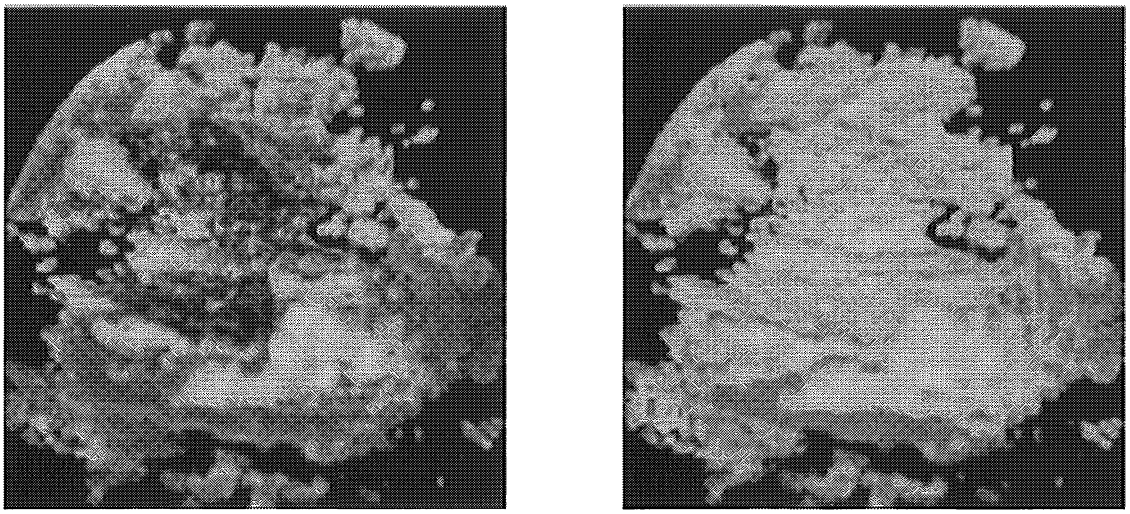
(a) (b)

Figure 2.6: Computing the low-albedo model using the emission-absorption model. During a preprocess (a) the light reflection model is computed, yielding emission values throughout the volume. Afterwards, (b) the emission-absorption model is run using these emission values.

2.2.4 Global Illumination

Although the emission-absorption and low-albedo models are physically-based, they express only a small range of natural lighting phenomena such as occlusion, simple shading, and optical depth. The next step in increased realism is to account for multiple scattering in the illumination model. Such models are called *global illumination* models because the illumination at any given volume element depends on the illumination of every other volume element.

Global illumination has been given little attention by researchers in scientific visualization; however, the problem has been studied from the point of view of realistic image synthesis. Kajiya and Von Herzen found that multiple scattering was essential for realistic rendering of clouds, since water droplets are a high-albedo medium [KV84]. Researchers in radiosity have used volume techniques to produce photorealistic scenes that include smoke and fog [RT87,Rus88]. Unfortunately, the extensive computational and memory requirements of these algorithms make them practical for only very small volumes (currently on the order of $32 \times 32 \times 32$ voxels) and clearly not sufficient for rendering modern scientific datasets.



(a) (b)

Figure 2.7: Comparison of image (a) with shadowing and (b) without shadowing.

An additional reason for the lack of interest in global illumination by the visualization community is the lack of clear evidence that more realistic lighting effects lead to improved image comprehension. In order to get a sense of the value of increased realism in volume-rendered images, we performed a comparison between images created with and without shadowing of the light source by the volume. We rendered several animation sequences of a $64 \times 64 \times 64$ dataset of cardiac ultrasound data, and tested several light source positions. Some rotation sequences were computed with the light source fixed relative to the object and others were computed with the light source fixed relative to the eyepoint. Frames computed with and without shadowing from the latter type of sequence is shown in Figure 2.7. There are two lights, one 45 degrees to the left and one 45 degrees to the right of the viewer. They are also inclined 45 degrees above the viewer. The relative positions of viewer, volume environment, and lights are shown in Figure 2.8.

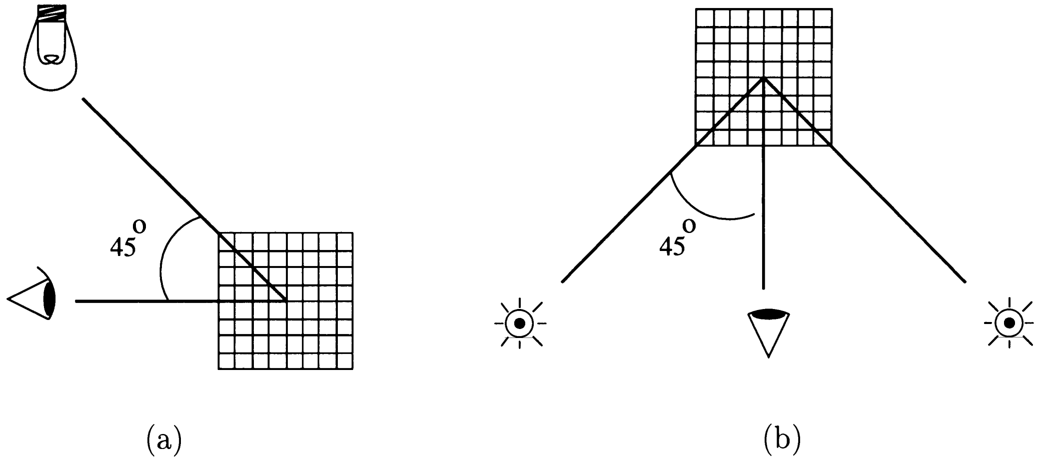


Figure 2.8: Geometry of Figure 2.7: (a) side view; (b) top view.

In all our experiments, we found that shadows did not improve comprehension of the scene, and frequently degraded comprehension. Two specific problems were:

1. Shadows obscured regions of the dataset. The effects of texture and shading in attenuated areas were minimized, thereby making perception of surface orientation difficult.

2. Unfamiliarity with the object meant that shadows appeared in unexpected places. Viewers often incorrectly interpreted shadowed areas as "dirty voxels".

Other researchers have incorporated shadows in volume rendering systems [Lev89, SN89, ASK92]. Their results demonstrate that shadows *are* effective when the shadows are cast by approximately convex objects onto orthogonal planes. In such cases, shadows give cues about the positioning of the volume objects in space relative to the planes as well as providing silhouettes of the object. None of these researchers demonstrate the use of shadows for highly concave scenes, such as the ventricle of the heart shown in Figure 2.7. More study is necessary, but our results show that improved realism does not guarantee improved comprehension.

Recent work by Krueger [Kru90] suggests that scattering terms provided by global illumination models may be useful for *data feature enhancement.* Using this scheme, scattering is used not to produce photorealistic images but instead to give the user textural cues about statistical properties of the volume data.

2.3 Discussion

In summary, the emission-absorption model was chosen for the work in this thesis because

- it is physically-based and can model familiar lighting effects such as occlusion, shading, and optical depth;
- it is general enough to subsume two of the more popular volume illumination models: attenuation projection and the low-albedo approximations;
- it gives the user great flexibility by allowing emission and absorption to be decoupled; and
- its simple form is conducive to error analysis.

The only major drawback of the model is that it cannot capture all the nuances of global illumination. However, we hope that understanding the accuracy and efficiency issues for the simpler model will act as a first step towards evaluating more complex models. In the meantime, more work is necessary to establish that the added complexity and expense of the global illumination is worthwhile in terms of visualization applications.

Having presented the model of illumination upon which this work is based, we continue in Chapter 3 by exploring existing algorithms for evaluating the model and rendering images.

Chapter 3

Volume Rendering Algorithms

In the previous chapter, we described ideal models of light transfer in an environment. In this chapter we shall see how these models are incorporated into a complete volume rendering algorithm. This chapter is organized around the three basic steps of volume rendering: *mapping*, *reconstruction*, and *projection*. These steps can be envisioned as part of a rendering pipeline, as illustrated in Figure 3.1.

The input to this pipeline is an array of raw data values at a discrete number of sample points in space. During mapping, raw data samples are interpreted to produce samples of the parameters that are required by the illumination model. For the emission-absorption model, these parameters will be α and ϵ. The reconstruction phase generates 3D scalar fields from the point samples of the illumination model parameters. Finally, during the projection phase, the illumination model is evaluated and a 2D image of the 3D data is produced. This image is then sampled for display on limited-resolution devices.

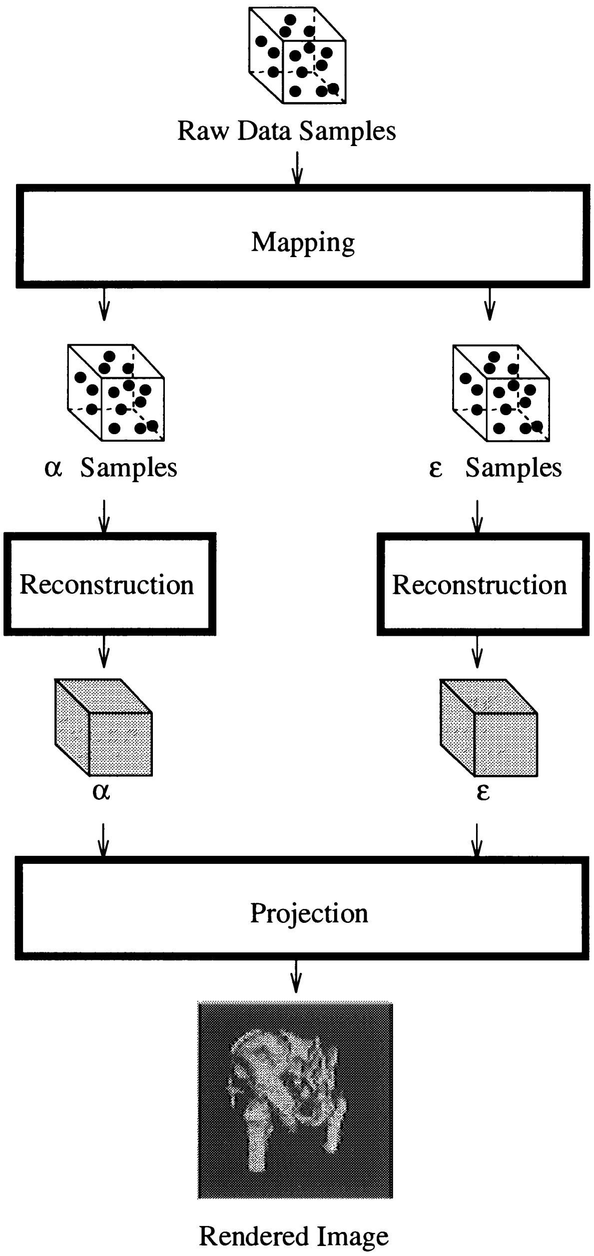


Figure 3.1: Pipeline showing the three basic stages of volume rendering.

Although the phases of mapping, reconstruction and projection are common to all algorithms, it is important to keep in in mind that this pipeline is only a conceptualization of the process. No volume rendering algorithm operates in such discrete stages, and many perform these steps in a different order.

3.1 Mapping

The input to a volume rendering algorithm is usually produced by scientific scanning equipment or by numerical simulations. The types of data that are input to volume rendering algorithms are extraordinarily diverse. For example, three-dimensional medical data available with current scanning technology include:

- Radiographic density (computed tomography)
- Relaxation constants (magnetic resonance imaging)
- Interfaces between media with different acoustical impedances (ultrasound)
- Fluid flow (Doppler imaging)
- Metabolic activity (positron emission tomography)

These data values generally do not correspond to the input parameters required by volume illumination models. Even when there is some similarity, the direct mapping does not necessarily communicate the desired information in the dataset. For example, radiographic density measured by a CT scanner could be used as the density constant ρ for particle-based low-albedo models (Section 2.2.3). However, if the physician only wishes to examine bone anatomy, the dense soft tissue would obstruct the view. Instead, a mapping which only assigns density to bone matter would be appropriate. Thus, mapping of raw data to rendering parameters is another type of abstraction that is used in volume rendering.

The emission-absorption model requires that raw data be interpreted and mapped to emission (ϵ) and absorption (α) values. The generality of the emission-absorption model allows a great deal of flexibility in this mapping. The most common approach has been to first classify each voxel according to what materials it contains, and then to assign emission and absorption values based on the desired visual properties of each material.

3.1.1 Classification

The problem of classification is essentially one of object recognition. Raw data values are examined and each voxel is marked with the material or materials that it contains. It is closely related to the field of computer vision, where video pixels are examined in order to recognize objects in a projected scene. The techniques used for classification depend heavily on the type of input data. Nevertheless, three general strategies have proved to be most popular for scientific visualization.

The simplest of these methods is called *thresholding.* This technique divides the raw data values into discrete ranges that correspond to different materials. The boundaries of these ranges are called thresholds. In order to classify the data values, ξ, into N materials, we sort the materials in order of increasing raw data range, $m_1 \ldots m_N$. We then define $N + 1$ thresholds $t_0 \ldots t_N$ and perform the classification indicated in Table 3.1.

Table 3.1: Classification via thresholding.

Raw Data Range	Material Type
$t_0 \leq \xi \leq t_1$	m_1
$t_1 \leq \xi \leq t_2$	m_2
$\vdots$	$\vdots$
$t_{N-1} \leq \xi \leq t_N$	m_N

For each material m_i, thresholding makes a binary decision at each sample point: "Does this sample point represent material m_i?" This is called *binary classification.* There are two problems with binary classification. The first is that it creates sharp boundaries between different materials, resulting in a misleading blocky appearance in the final image if the material boundaries are not aligned with the sample points. Furthermore, it is susceptible to classification errors due to the "partial volume effect".

The partial volume effect occurs in measured data, where samples do not represent values at infinitesimal points in space. Rather, they represent the values across small finite volumes. A sample volume may therefore contain more than one type of material. In such a domain, a binary classification scheme is bound to make inappropriate "all-or-none" decisions. The decision it makes is not even guaranteed to correspond to one of the materials present in the sample volume. For example, if materials with non-adjacent raw data ranges occupy the same sample volume, the resulting average data value might indicate to the threshold scheme that the sample represents a material whose data range falls between that of the two component materials.

An improvement on thresholding, presented by Levoy [Lev88], compensates for the partial volume effect. Let ξ_i be the the raw data value output by the scanner if a voxel is completely occupied by material m_i. His method assumes that

- at most two materials share a sample volume; and
- if m_i and m_j share a sample volume then there does not exist another material m_k with a ξ_k value between ξ_i and ξj.

The technique classifies a raw data value ξ by first determining the materials m_i and m_j such that

$$\xi_i \leq \xi < \xi_j.$$

The volume represented by ξ is then classified as being

$$\frac{\xi - \xi_i}{\xi_j - \xi_i} \times 100$$

percent material m_i, with the remainder classified as being m_j. These perfect conditions tend not to occur in real datasets that have more than two types of materials. However, with data such as medical CT scans, the conditions are approximately held and impressive visual results have been achieved using this technique despite the inaccuracies.

A generalization of Levoy's method is based on a technique known as maximum-likelihood classification. This heuristic technique requires that probability density functions $P_i(\xi)$ are given *a priori* for each material m_i. The value of $P_i(\xi)$ is the probability the raw data value ξ will be output by the scanner for a voxel that contains some amount of material m_i. A major drawback of this technique is the fact that $P_i(\xi)$ are rarely known in advance, and must be estimated. Assuming that the $P_i(\xi)$ functions are known, the probability that data value ξ will represent material m_i is estimated by

$$p_i(\xi) = \frac{P_i(\xi)}{\sum_j P_j(\xi)}.$$

The maximum-likelihood algorithm classifies the data value as being the material with the highest $p_i(\xi)$ value. Researchers at PIXAR [DCH88] adapted this scheme to compensate for the partial volume effect by classifying each data value as a mixture of materials. The fraction of each component material is given by $p_i(\xi)$. Since

$$\sum_i p_i(\xi) = 1,$$

the entire sample volume is accounted for with this scheme. In practice, the abstraction of mapping of probabilities to fractional occupancy works quite well. The maximum-likelihood classification scheme is easy to evaluate, and has the additional benefit of generalizing to multidimensional input. However, it is limited to domains where values of $P_i(\xi)$ are available in advance, and, even in the best of conditions, is only an estimate.

Another classification scheme presented by Levoy [Lev88] is useful for visualizing data isosurfaces. This problem is more difficult than simply marking sample points with values exactly on the surface, since the isovalue might fall in between sample points, leaving holes in the rendered image. Levoy's method avoids this pitfall by giving the isosurface a thickness, thus classifying voxels that are close to the isovalue as partially containing the isosurface. Distance from the isosurface is estimated by using the magnitude of the gradient as an approximate rate of change in the neighborhood around a sample point. Using this estimate, the distance of a sample at (x, y, z) from an isosurface with value ξ_{iso} is given by

$$d = \left| \frac{\xi(x,y,z) - \xi_{\text{iso}}}{|\nabla \xi(x,y,z)|} \right| .$$

Techniques for computing gradients will be covered in Section 3.1.3. Levoy suggests that a voxel's fraction of isosurface fall off linearly between $d = 0$ and $d = r$, where r is the desired thickness of the isosurface. This technique has been demonstrated to produce realistic isosurfaces of electron density maps [Lev88].

The automatic classification schemes listed above have been employed to produce many successful volume-rendered images. However, their performance at object-recognition does not approach human abilities. For example, in medical imaging, a radiologist can easily identify the boundaries of different types of soft tissue on CT scans, such as the liver and the kidney. This type of distinction is not possible using the techniques presented in this chapter. Often, such classifications must be performed manually, with trained users identifying objects on individual slices of data. More sophisticated automatic techniques such as model-based recognition are being explored in fields such as robotics and computer vision. They employ domain-specific knowledge to aid in classification. In a medical imaging system, this knowledge might include the expected shape of the kidney. It is likely that model-based recognition will find future application in scientific visualization, however to date they have not been widely adopted. This is partly due to the fact that most scientists are interested in seeing their data in essentially "raw" form, with a minimum of automatic interpretation.

3.1.2 Absorption

Once the volume is classified, the mapping of raw data to absorption may be performed based on the desired transparency of each material. Since absorption is measured per unit length, it is also necessary to consider the expected thickness of the material. The relationship between transparency, T, absorption, α, and material thickness, d can be derived starting with Equation (2.7), which we reproduce here:

$$I(a,b) = I_b e^{-\int_a^b \alpha(v)\,dv}.$$

This equation can be used to compute transparency by assuming a unit backlight value, I_b,

$$T(a,b) = e^{-\int_a^b \alpha(v)\,dv}. \tag{3.1}$$

We then introduce the absorption coefficient and material thickness,

$$T = e^{-\int_0^d \alpha\,dv},$$

and solve the integral:

$$T = e^{-\alpha d}.$$

Solving for α yields an expression for the attenuation coefficient, given the desired transparency and material thickness,

$$\alpha = -\frac{\log(T)}{d}.$$

If the classification of a sample point yields more than one material, the absorption coefficient for the sample point can be set to the weighted average of the absorption coefficients of the component materials. The weights are determined by the fractional occupancy of each material.

3.1.3 Emission

The emission function is typically determined by two factors: self-emission and reflected light. It therefore corresponds to the "source term" found in the radiative transfer literature. Self-emission is an inherent glow of the material. This could be used to highlight muscle tissue by making it emit red light. Self-emission is useful for generating colored fog-like effects, but gives no surface information.

Reflected light is determined by a light reflection model. The simplest light reflection models are based on phase functions. Phase functions depend only on the *phase angle*, defined as the angle between the viewer and the light source as measured from the reflecting particle. For example, if the particle is spherical, significantly larger than a wavelength of light, and the surface is diffuse, the reflectivity will depend on the fraction of the illuminated hemisphere of the particle that is visible from the eyepoint. Thus, the phase function will decrease monotonically with increasing phase angle. Figure 3.2 shows views of an illuminated particle with two different phase angles. To relate this to a familiar effect, note that the shapes in Figure 3.2 are identical to those associated with phases of the moon. Phase functions have been used successfully to render images of clouds and fog [Bli82, KV84,RT87]. Their usefulness is limited for visualization, since shading depends only on the relative positions of the particle, light source and eyepoint. It does not reveal any information about the particular dataset.

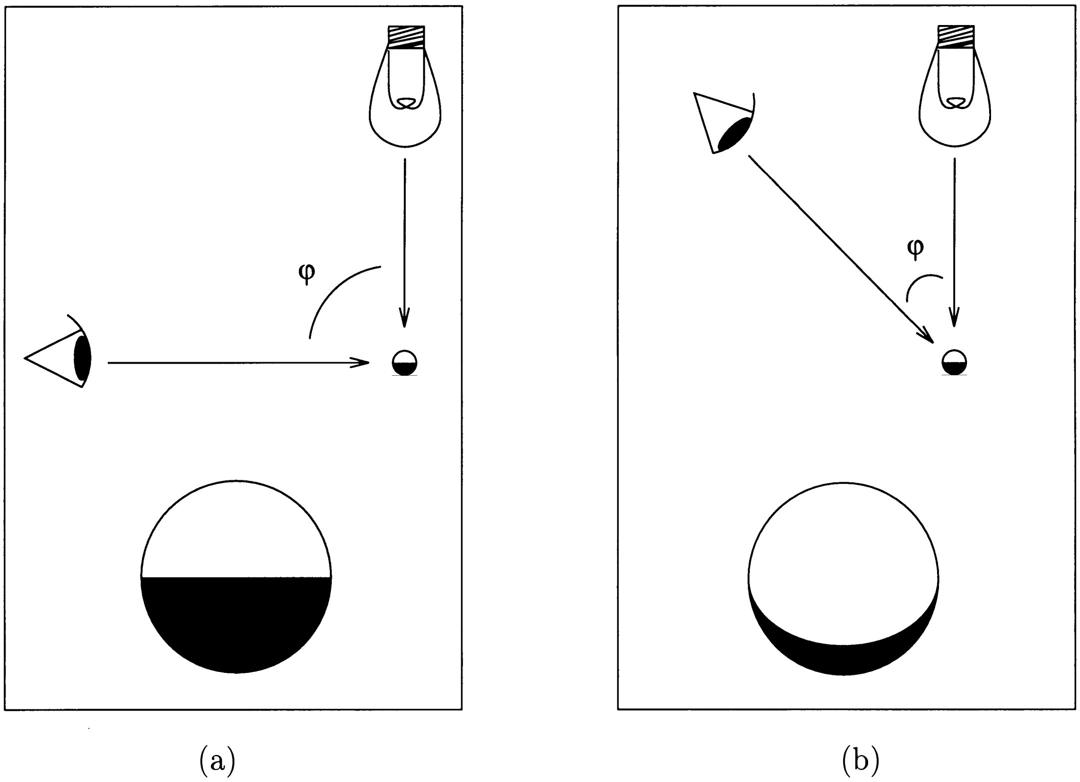


Figure 3.2: View of a particle with two different phase angles, φ: (a) 90 degrees, (b) 45 degrees. Above is a top view of the scene, showing the phase angle. Below is the view of the particle from the eyepoint.

An important advance in the power of volume rendering came when gradient shading was introduced [HB86,Lev88,DCH88]. Surface reflection models, which had been used in computer graphics for years, shade objects in such a way that humans intuitively discern the orientation of their surfaces. These lighting models always depend on the *normal* to the surface in question. The problem for volume rendering is that since there are no explicitly-defined surfaces, there is no inherent concept of surface normal. The solution has been to use the gradient of the volume data as a substitute for the surface normal. The gradient will always point in the direction of maximum change in the data. Thus, if a boundary exists in the data, the gradient will point perpendicular to it.

Since the gradient of the data is not known independently, it is computed using a discrete approximation. The two most common techniques are forward-differencing [DCH88],

$$G(x,y,z) = \left\{ \begin{array}{c} \frac{v(x+\Delta x,y,z)-v(x,y,z)}{\Delta x}, \\ \frac{v(x,y+\Delta y,z)-v(x,y,z)}{\Delta y}, \\ \frac{v(x,y,z+\Delta z)-v(x,y,z)}{\Delta z} \end{array} \right\}$$

and central differencing [Lev88],

$$G(x,y,z) = \frac{1}{2} \left\{ \begin{array}{c} \frac{v(x+\Delta x,y,z)-v(x-\Delta x,y,z)}{\Delta x}, \\ \frac{v(x,y+\Delta y,z)-v(x,y-\Delta y,z)}{\Delta y}, \\ \frac{v(x,y,z+\Delta z)-v(x,y,z-\Delta z)}{\Delta z} \end{array} \right\}.$$

On a regular rectangular grid, computation is simplified by computing the gradient only at grid points, and choosing Δx, Δy and Δz to correspond to the sample spacing along each dimension.

Two problems exist in using the gradient as a surface normal in shading. The first is that light reflection models assume that the surface normals point outward from the "front" of the surface. When using the gradient as a surface normal, this direction is automatically determined to be towards the side with higher density. This is not necessarily the side that the user considers the front. The problem can be avoided by using a light reflection model that reflects equally on both sides of the surface, or by manually inverting the gradient directions in regions where an error is made.

A more thorny problem is that the gradient operator acts as a high-pass filter, which serves to emphasize noise in the image. The visual result is that the normals jiggle and do not give the impression of a smooth surface. Note that the noisy gradient is not always a disadvantage; it adds texture to surfaces which, along with a perspective projection can be used as a depth cue. Two approaches to minimizing noise in the gradient have been proposed. First, the gradient can be taken on a smoothed version of the dataset [DCH88], while the unsmoothed version is used for the other steps in rendering. Another approach is to compute the gradient over a larger domain than used in forward-differencing and central-differencing [HBP+90].

There is some debate in the literature about whether the gradient should be taken on the classified data or on the raw data values. Using classified data ensures that the shading is consistent with the objects that are displayed [DCH88]. Levoy argues that by using the raw data values, errors in classification can be minimized, since proper shading based on the "real" values is always present [Lev88]. This issue remains unsettled. For the images presented in this thesis, we use the gradient of the raw data, however, use of the classified gradient would not change the algorithms or analysis contained herein.

Once the gradient is computed, any of a number of local shading models may be used. The most popular is the Phong shading model [BT75], which has the general form,

$$b = k_a + k_d \cos(\theta) + k_s \cos^n(\phi),$$

where θ is the angle between the vector pointing towards the eyepoint, and the surface normal, and ϕ is the angle between the vectors pointing towards the viewpoint and towards the direction of mirror reflection. These vectors and angles are illustrated in Figure 3.3. The Phong model is the sum of three terms, an ambient term, a diffuse term, and a specular term. There are also four constants: k_a, k_d, k_s and n. The ambient term is simply the constant k_a, and acts as an approximation to the effects of multiple scatter. The diffuse term models ideal Lambertian reflection and is weighted by the constant k_d. The effect of a specular highlight is approximated by the third term, which is weighted by the constant k_s. The sharpness of the highlight is controlled by the constant n.

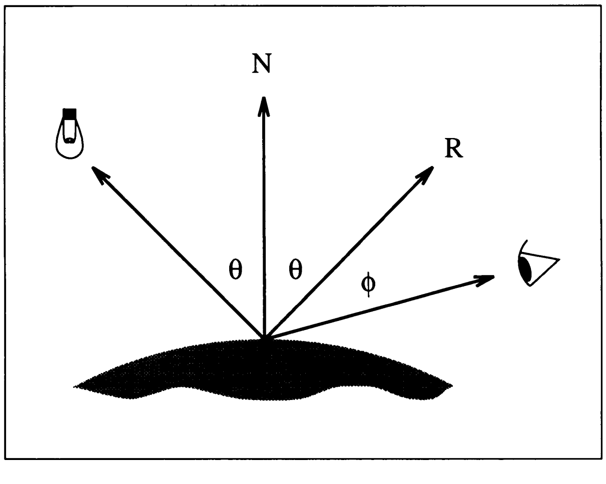


Figure 3.3: The angles used in Phong illumination, θ and ϕ, are shown in the above diagram. The surface normal is N and the direction of the maximum specular highlight is R. After Foley *et. al.* [FvFH90].

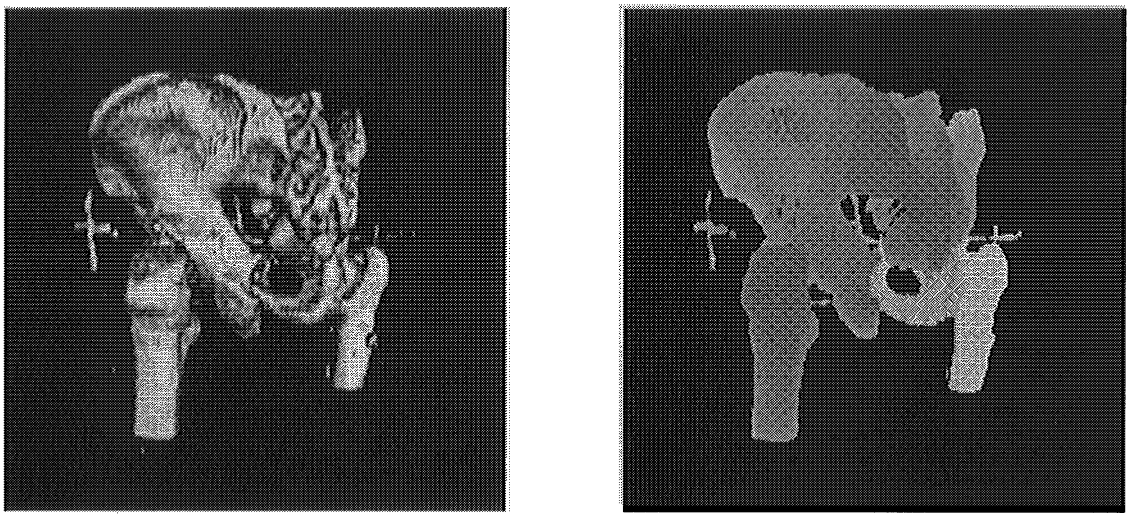
The Phong lighting model is not physically-based, however it does produce realistic shading effects. The diffuse term and particularly the specular term are highly sensitive to deviations in surface normal and thus communicate the orientation of the surface quite well. This model also lends itself to easy evaluation, since the cosine terms may be evaluated using dot products.

Research has led to more realistic local illumination models than the Phong model [CT82,HTSG91]. Although there are no technical barriers to their application, the use of these models has not been reported in the volume rendering literature. Increased computational cost and lack of interest in photorealism among the visualization community are the likely causes for researchers remaining with the less sophisticated model.

As a demonstration of the effectiveness of applying local illumination models based on the gradient, we present in Figure 3.4 an image computed using gradient shading and the same image computed using a phase function.

3.2 Reconstruction

Although it is possible to produce volume data that is defined by mathematical functions in three dimensions, the overwhelming majority of volume data is presented as sample points in 3D space. If these datasets were projected without any type of reconstruction, the resulting image would appear as a scatterplot. Instead, we wish to reconstruct the input to the renderer so that the final image gives the impression of objects that occupy volumes of space. Construction and reconstruction of functions defined on a continuous domain from point samples is a well-studied problem in the field of signal processing. What follows is a brief



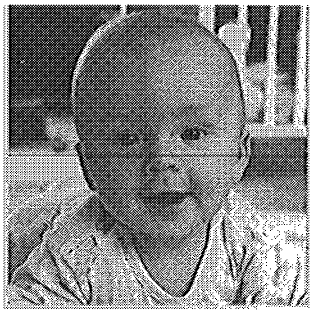
(a) (b)

Figure 3.4: Comparison of image (a) with gradient shading and (b) with shading by a phase function.

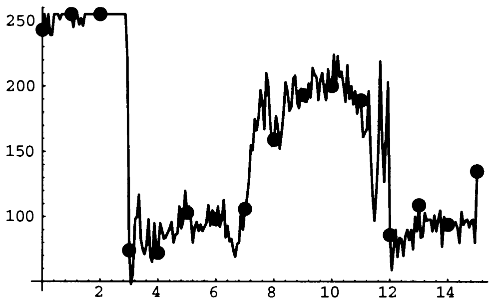
overview of the techniques used in volume rendering and their benefits and costs. More detailed information about the reconstruction problem, can be found in an introductory signal processing text, such as the one by Oppenheim and Schafer [OS75].

Signal processing theory states that a function may be perfectly reconstructed from its samples if it is band-limited and if it is sampled at twice the maximum frequency of the data. There are two practical difficulties in applying this ideal reconstruction to sampled volume data. The first is that the ideal reconstruction filter is a sinc function, which has infinite extent. Thus, to reconstruct any point in 3D space, we must consider the values of all sample points, not just those in a local neighborhood. This quickly becomes computationally intractable. Another difficulty is that real-life datasets tend not to represent band-limited scenes. Any scene that contains a sharp boundary between two media will represent a step function in 3D which has infinite extent in the frequency domain. Use of ideal reconstruction on a dataset which is not band-limited will cause ringing artifacts.

The effects of the different filters that we will present in this chapter will be demonstrated using a one-dimensional reconstruction problem. The problem is the reconstruction of a scanline of a 2D image that has been intentionally undersampled to reflect the kind of difficulties that are faced when visualizing real-world scientific data. Figure 3.5 shows the original image, and a plot of the selected scanline showing the samples that were taken. Note that the original image was digitally scanned, so it is also made up of a discrete number of samples. The plot of the scanline in Figure 3.5b was created by joining the sample points with linear segments. Figure 3.6 shows the result of reconstruction with a sinc filter applied to these samples. Note that the band-limited output of this filter causes ringing in areas where the actual function is flat.

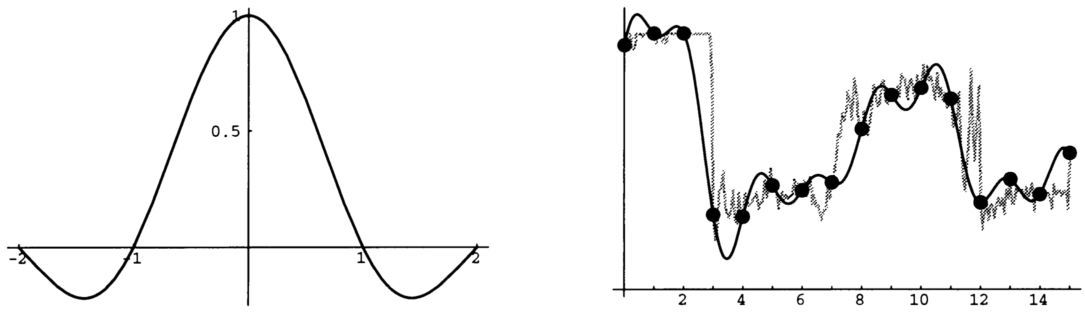


(a)



(b)

Figure 3.5: Sample reconstruction problem: (a) 2-D digitized photograph indicating scanline to be reconstructed. (b) Raw scanline data indicating subsampling points.

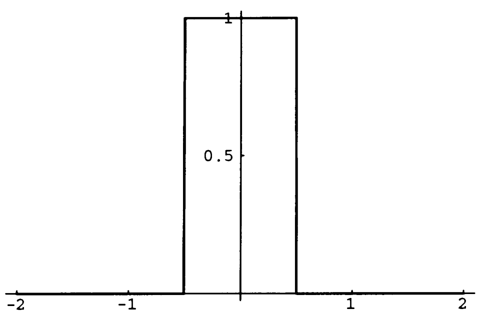


(a) (b)

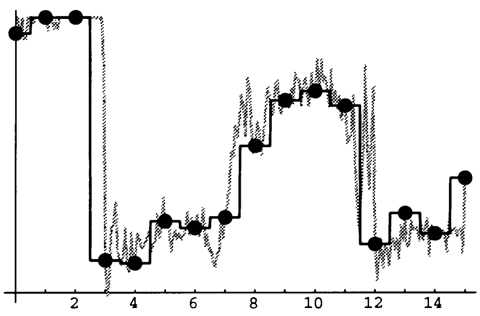
Figure 3.6: Reconstruction with a sinc filter: (a) Filter shape in the x interval [-2,2], (b) reconstructed scanline.

Nearest-neighbor interpolation is the most basic form of reconstruction used in volume rendering [SSW86,Sab88]. This method states that the value of the function at any point in three-dimensional space is equal to the value of the nearest sample. Nearest-neighbor interpolation has the advantage of being very easy to evaluate. However, the reconstructed signal is distorted, with sharp discontinuities between cells. This gives images computed using nearest-neighbor interpolation a distracting, blocky appearance. Figure 3.7 shows the effect on the sampled one-dimensional data. Another feature of nearest-neighbor interpolation is that the only output values it produces are those that exist in the raw data – no values are synthesized. This is an attractive feature to those more concerned with the accuracy of the output values than with the spatial positioning of those values.

The degree of continuity between cells can be increased by employing trilinear interpolation. This three-dimensional analog of linear interpolation operates on hexahedral cells with sample values at the corners. A trilinear function has the



(a)



(b)

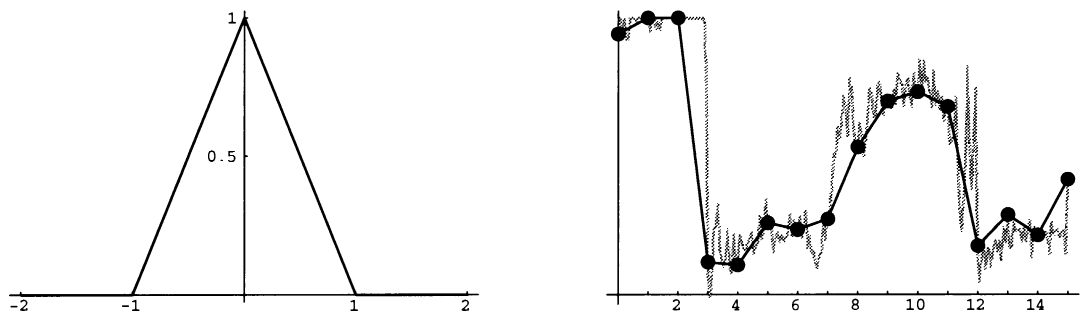
Figure 3.7: Sample scene reconstructed with nearest-neighbor interpolation: (a) Filter shape, (b) reconstructed scanline.

form:

$$f(x, y, z) = c_0 + c_1x + c_2y + c_3z + c_4xy + c_5yz + c_6xz + c_7xyz.$$

The coefficients $c_0 \ldots c_7$ are computed by solving a linear system. The requisite eight equations may be written by substituting the x,y,z and f values at the eight corners into the above equation. If the grid-spacing is regular, the matrix may be inverted as a preprocess [UK88]. Trilinear interpolation represents a good trade-off between smoothness of the output function and computational cost. It is by far the most popular form of interpolation used in current volume rendering algorithms [Lev88,UK88,WG91]. However, it has some drawbacks. It is limited by its requirement that there exist hexahedral cells, therefore it is not appropriate for irregularly-sampled data. Furthermore, the derivative discontinuity at cell boundaries can result in perceptual anomalies such as Mach-banding [Rat72]. The result of linear interpolation is shown in Figure 3.8.

Piecewise constant and trilinear interpolation are both forms of a general interpolation strategy called polynomial spline interpolation. These result in the



(a) (b)

Figure 3.8: Reconstruction with linear interpolation: (a) Filter shape, (b) reconstructed scanline.

definition of a separate polynomial function for each cell. The order of continuity between cells can be increased by increasing the degree of the polynomial function within each cell. To obtain a C_1 output function, tricubic interpolation may be used. In this case, the function can be defined across a hexahedral cell by considering all the sample points in a 3x3x3 cell neighborhood, for a total of 64 data values. The use of this computationally intensive technique has not been reported in the volume rendering literature.

Another approach is to use a Gaussian filter kernel [Wes89,Wes90,LH91]. In its ideal form, a Gaussian filter kernel has infinite extent. In practice, however, it is clipped so that it extends over only a few sample points. Gaussian interpolation has the drawback that its output value at the sample points may differ from the input values at these points. Furthermore, the extreme smoothing that takes place tends to give the dataset a blurry appearance. Figure 3.9 demonstrates Gaussian interpolation on the one dimensional example.

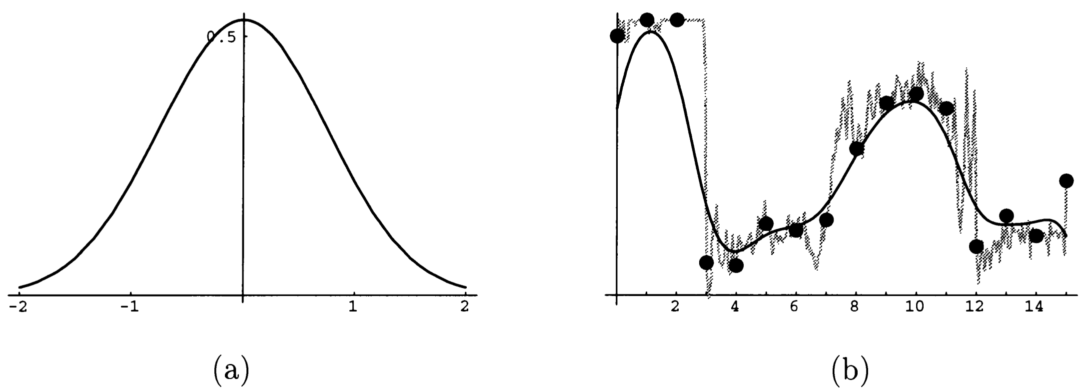


Figure 3.9: Sample scene reconstructed with Gaussian interpolation: (a) Filter shape, (b) reconstructed scanline.

3.3 Projection Algorithms

During the projection phase of volume rendering, the three-dimensional scene is mapped onto a two-dimensional image plane. In volume rendering, projections are defined by a view transformation and an illumination model. The view transformation determines the set of points in space that are mapped to a single point on the image. The illumination model (several of which were discussed in Chapter 2) determines the color that results from this many-to-one mapping.

3.3.1 View Transformations

The two most popular view transformations are orthographic projection and perspective projection. Both require the definition of an image plane. Using an orthographic projection, the set of points that share an image-space mapping are those that lie along a common line perpendicular to the image plane. Using perspective projection, a center of projection must be defined. The set of points that share an image-space mapping are those which lie along a common line that passes through the center of projection. Further information about projective geometry may be found in introductory computer graphics texts, such as the one by Foley *et. al.* [FvFH90].

Perspective projections are desirable since they provide familiar perceptual cues. However, orthographic projections are usually simpler to compute because the sets of points that form an equivalence class after projection lie along parallel lines. This parallel nature of the projection enables the programmer to uniformly sample world space by uniformly sampling screen space. Further discussion of projection will be presented in Chapter 5.

Regardless of the choice of view transformation, a two-dimensional reconstruction step must be added to the rendering process. The illumination function is defined at all points on the image plane. However, the image must be stored in limited memory and must be displayed on devices with only a finite number of rectangular picture elements called *pixels*. It is assumed that each of these pixels displays a single constant value across the entire pixel area, although in reality there is some variation. Typically, the images will be stored as point samples that correspond to the function's values at pixel centers. Alternatively, the point samples may correspond to an average of function values across pixel area. Further information about generating pixel values may be found in Foley *et. al.* [FvFH90].

3.3.2 Partitioning the Integral

Before discussing specific approaches to computing projections, we present a technique for partitioning the volume rendering integral so that solutions to the integral for individual ray segments may be combined to form a solution for an entire ray. Partitioning in this way is employed in nearly all existing volume projection algorithms.

To make the partition, we first rewrite the integral using the transparency function, $T(a, b)$ that was first defined in Equation (3.1),

$$T(a,b) \;=\; e^{-\int_a^b \alpha(v)\,dv}. \tag{3.2}$$

Recall that this function has a physical interpretation; it is the transparency or optical depth of the material between points a and b along the ray. The volume rendering integral can now be rewritten.

$$I(a,b) \;=\; \int_a^b T(a,u)\,\epsilon(u)\,du. \tag{3.3}$$

Observe first that transparency obeys a simple multiplicative rule. In particular, for any point x in the ray segment $[a, b]$, we have

$$T(a,b) \;=\; T(a,x)\,T(x,b),$$

as is easily verified from equation (3.2).

We can therefore discretize the ray into n contiguous segments $[d_0, d_1]$, $[d_1, d_2]$, $\ldots$, $[d_{n-1}, d_n]$ at distances $a = d_0 < d_1 < \cdots < d_n = b$ from the eye point, and rewrite equation (3.3) as follows:

$$\begin{aligned} I(d_0,d_n) &= \sum_{i=1}^{n} \int_{d_{i-1}}^{d_i} T(d_0,u)\,\epsilon(u)\,du \\ &= \sum_{i=1}^{n} \int_{d_{i-1}}^{d_i} T(d_0,d_{i-1})T(d_{i-1},u)\,\epsilon(u)\,du \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} T(d_0, d_{i-1}) \int_{d_{i-1}}^{d_i} T(d_{i-1}, u)\, \epsilon(u)\, du \\
&= \sum_{i=1}^{n} T(d_0, d_{i-1})\, I(d_{i-1}, d_i).
\end{aligned}
$$

Finally, by introducing the notation $I_k = I(d_{k-1}, d_k)$ and $T_k = T(d_{k-1}, d_k)$ we can rewrite this equation more succinctly:

$$
\begin{aligned}
I(d_0, d_n) &= \sum_{i=1}^{n} I_i\, T(d_0, d_{i-1}) \\
&= \sum_{i=1}^{n} I_i \prod_{j=1}^{i-1} T_j,
\end{aligned}
\tag{3.4}
$$

where, $\prod$, the product operator is defined,

$$
\prod_{i=1}^{n} a_i = a_1 \times a_2 \times \cdots \times a_n.
$$

Equation (3.4) expresses the overall intensity of the ray in terms of the intensities and transparencies of the discrete segments.

Equation (3.4) can also be expressed in terms of image compositing operators. These general-purpose operators are used to overlay semitransparent images. In order to use them, it is necessary to store tuples that associate intensity values and transparency values with one another. We shall make use of two of the operators, **over** and **under** defined by Porter and Duff [PD84]. Given pixels A and B, with color and transparency values $[A_c, A_t]$ and $[B_c, B_t]$ the operators are defined,

$$
\begin{aligned}
A\ \textbf{over}\ B &\equiv [A_c + (B_c \times A_t),\ A_t \times B_t] \\
A\ \textbf{under}\ B &\equiv [(A_c \times B_t) + B_c,\ A_t \times B_t],
\end{aligned}
$$

and correspond to image overlays and underlays. Our notation is slightly different from Porter and Duff's since we use transparency in our tuples rather than opacity.

If we define C_i to be a tuple containing both the intensity and opacity value for the segment i,

$$C_i = [I_i,\ T_i]\,,$$

then Equation (3.4) may now be written:

$$[I(d_0, d_n),\ T(d_0, d_n)] \quad = \quad C_1 \textbf{ over } (C_2 \textbf{ over } (\ldots(C_n)\ldots)).$$

This formulation is significant, since fast hardware implementations of image compositing operators are available on most graphics workstations.

3.3.3 Ray Casting Algorithms

The first category of projection algorithms that we will consider are the ray casting algorithms [Lev88,Sab88,UK88]. These algorithms work by point-sampling the image plane. At each of these point samples, the illumination is determined by firing a ray through all object space points that are mapped to the image-space sample point. The ray is broken into small regions for which approximations of I_i and T_i are computed. The results are composited using the techniques of the previous section.

Ray casting algorithms generate approximations of I_i and T_i using quadrature rules that will be described in Chapter 4. Evaluation of these rules require that either

- α and ϵ are reconstructed at sample points along each ray, or
- an analytic expression for the reconstructed signal in generated for each region.

In either case, the cost of reconstruction dominates the computation.

Research in ray casting has stressed efficiency improvements. In standard ray tracing, Arvo and Kirk [Gla89] found three basic categories of improvements. Two of these categories have direct analogies in volume ray casting, as noted by Levoy [Lev89]. Algorithms in the first category seek to reduce the number of samples of the image plane that are taken. For example, adaptive supersampling uses a heuristic to determine regions of the image plane where the illumination function is slowly varying [Whi80,Lev90a]. The second category of efficiency improvements seek to reduce the time spent computing the illumination value along each ray. Octrees can help by enabling the algorithm to skip over empty regions or make coarse approximations in regions that experience little variation of emission and absorption [Lev90a,DH92]. Early ray-termination algorithms can be used when rays are traced from front to back. In this case, once a user-specified amount of dense material has been encountered, the remainder of the ray is ignored since any structures encountered would largely be occluded. In standard ray tracing, most speedup techniques introduce no new inaccuracies, however this is not the case in volume ray casting. The main drawback to all the above techniques is that they introduce approximation error into the final image without characterizing its magnitude.

There are several drawbacks to the ray casting approach. Since rays are normally traced one-at-a-time, references to memory are not well-localized, causing paging problems with large datasets. Furthermore, the algorithm operates in screen space, and typically does not adjust based on the resolution of the dataset. Thus, the scene tends to be oversampled in some instances, and undersampled in others. We address both these problems in Chapter 5. Additionally, ray casting does not lend itself to distributed-memory parallel implementations, since each processor requires access to the entire dataset.

Despite these disadvantages, ray tracing remains one of the most popular volume rendering algorithms. It is conceptually quite simple, and easy to implement. Ray casting software can be tuned to change projection type or reconstruction filter with a minimum of effort. By varying the number of samples taken per pixel or the quadrature rule that is employed along each ray, the user has the opportunity to trade rendering time for image quality.

3.3.4 Data-Space Algorithms

Whereas ray-casting algorithms have a main loop that iterates once per image-space sample, data-space algorithms iterate once per data sample. Ray casting algorithms ask how each data sample affects the current image plane sample. Data-space algorithms ask how each image-space sample point is affected by the current data-space sample.

The Pixar Algorithm

The approach introduced by Duff *et. al* [DCH88] is optimized to employ image compositing hardware. The basic algorithm requires that

- Data is stored in a regular, rectangular grid;

- An orthographic projection along one of the principal axes of the dataset is desired; and
- Screen resolution is equal to the 2D resolution of the dataset along the axis of projection.

However, the algorithm is more general than the above restrictions imply. Many of the preconditions may be satisfied by employing additional resampling operations. Irregular data may be resampled onto a regular grid. Views that are not aligned with data axes may be computed by rotating the data within the grid so that the view may be achieved along one of the principal axes. These rotations may be performed efficiently using the three-pass technique described by Hanrahan [Han90]. During each pass, a 1D shearing operation is performed along one of the axes, as shown in 2D in Figure 3.10. Perspective views may be computed by predistorting the dataset, so that orthographic projections of the predistorted dataset exhibit familiar perspective depth effects.

Once the data has been manipulated so that all the preconditions apply, the data is processed and stored into a single three-dimensional array of (r, g, b, α) values, where r, g, and b are determined by computing the value of I_i for a ray passing through the center of each voxel along the view direction. The α values are determined by computing the corresponding T_i value. Then each two-dimensional slab of the data can be treated as an image with an associated alpha channel and composited using the **over** and **under** operators defined in Section 3.3.2.

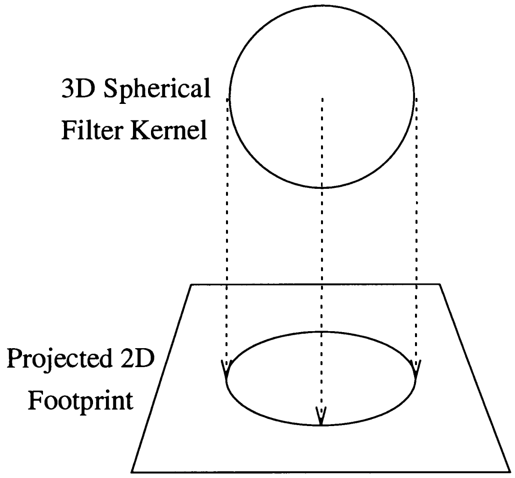
poor choice for both very small and very large datasets.

Splatting

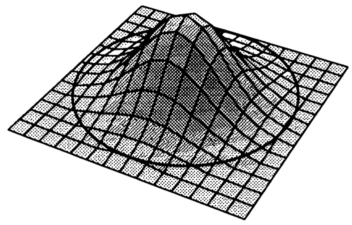
Splatting is a data-space technique that seeks to exploit the coherence that is present in certain types of reconstructed data. Assuming the samples are regularly-spaced, the 3D functions of emission and absorption are simply sums of scaled, translated copies of the reconstruction kernel. The main concept in splatting, which was introduced by Westover [Wes89,Wes90], is that each of these scaled and translated copies of the reconstruction kernel can be projected individually, while summation (or compositing) occurs in screen space. The coherence that splatting exploits arises when computing an orthographic view. In this case, the size and shape of the projected filter kernel does not vary with 3D position, so it can be projected once as a preprocess.

We now consider the steps involved in the splatting algorithm. The reconstruction filter, usually a Gaussian [Wes90], is integrated in depth, thus creating a generic screen-space footprint that is stored in a lookup-table at subpixel resolution. This process is illustrated in Figure 3.11. The data samples are considered from front-to-back or from back-to-front. The footprint is centered at the data point's screen-space projection, and its values are scaled based on the emission and absorption values at the data point. Finally it is composited into a final image using **over** or **under** operations.

This algorithm differs from other approaches to volume rendering in that the processes of reconstruction and projection are reordered. Unfortunately, the emis-



(a)



(b)

Figure 3.11: A preprocess to the splatting algorithm is the creation of a generic footprint table. (a) A 3D reconstruction filter over a spherical domain is projected down to a 2D footprint. (b) The generic footprint table stores the integral of the reconstruction filter at a discrete number of sample points. After Westover [Wes90].

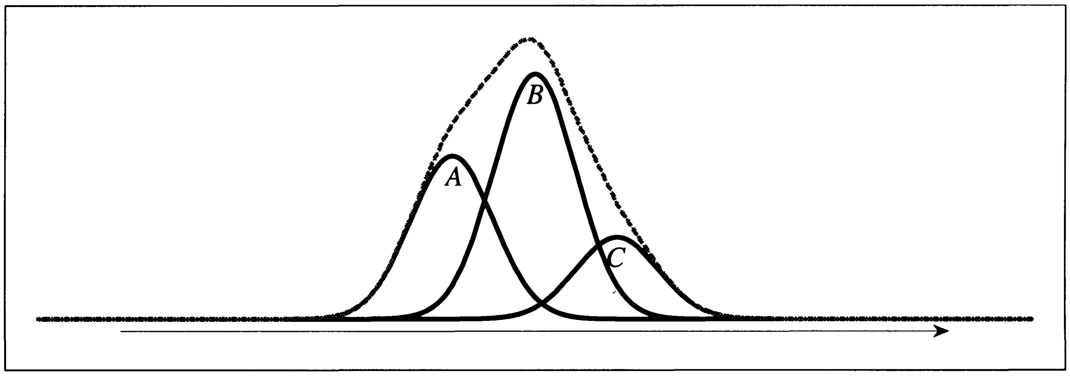
sion-absorption and low-albedo models of volume rendering are non-linear in α and ϵ, hence reconstruction and projection are not associative in the general case. They are associative when filter kernels do not overlap in space, as noted by Laur and Hanrahan [LH91], but such filters cannot ensure C^1 or higher continuity.

Splatting can be seen as an approximation to the emission-absorption model. Filters that overlap in space are approximated by those that are spaced so that they do not. This is illustrated in Figure 3.12. In Figure 3.12a, a function along a ray is shown, along with the scaled and translated filter kernels which are its addends. This function can be considered emission or absorption. The filter kernels are clipped Gaussians. Note that these curves overlap with one another along the ray axis. Therefore, although the ray encounters most of curve A before curve B, portions of curve A extend into the domain of curve B and vice-versa. The splatting algorithm renders each of these curves as if they do not overlap along the ray axis, as shown in Figure 3.12b. This approximation can result in values of illumination that are either too bright or too dark, depending on the data. Studies of the magnitude of this error have not been published.

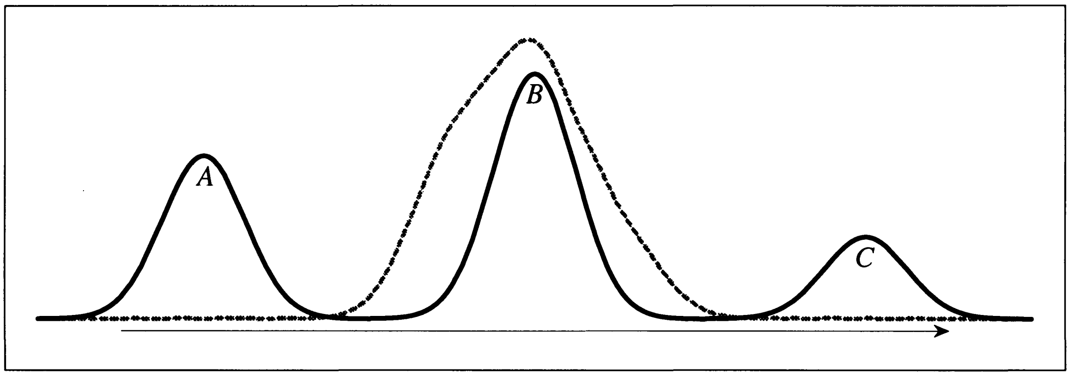
Images produced by splatting tend not to have the same level of detail that is present in ray-traced images. It is not clear how much of this loss of detail is due to the use of a wide Gaussian filter kernel, and how much is due to the approximation error described above.

The initial implementations of splatting required regular, rectangular datasets and orthographic projections. At the cost of adding additional approximation error, Laur and Hanrahan introduced modifications to handle rectangular hierarchical data and perspective projections [LH91]. A further algorithmic modification they introduced was the approximation of the splats by Gouraud-interpolated (r, g, b, α) polygons. This enables the use of hardware polygon engines for rendering the splats.

Splatting is attractive because it is easy to implement, runs quickly, is easily parallelized, and can be optimized to use rendering hardware. However, it is also an approximate technique for which the magnitude of the error is unknown. Furthermore, this error cannot be minimized by modifying the input parameters. Splatting is best used to create a "quick-preview" visualization engine, useful in situations where where interactive feedback is required.



(a)



(b)

Figure 3.12: Errors due to the reordering of reconstruction and projection in splatting. (a) A function defined by truncated Gaussian interpolation (dashed line) is the sum of several truncated Gaussians (solid lines A, B and C). (b) The splatting algorithm does not allow filters to overlap in depth. To compensate, the truncated Gaussians must be translated relative to one another so that they no longer intersect. Recovery of the function is no longer possible.

Cell-by-cell processing

Instead of considering data samples individually, the cell-by-cell processing algorithms consider spatial regions whose value is interpolated from the same set of data samples. In the case of trilinear interpolation, the regions are rectilinear cells, with data samples at each of the eight corners. The cells are considered in front-to-back or back-to-front order, while the image is accumulated in a buffer. In contrast to splatting, there is no overlap in depth between the cells. Thus, the problem of splatting error described in the previous section is avoided.

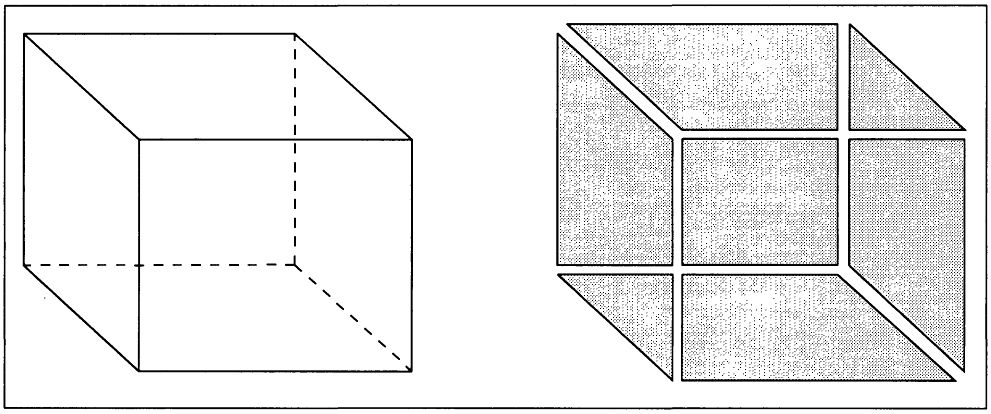


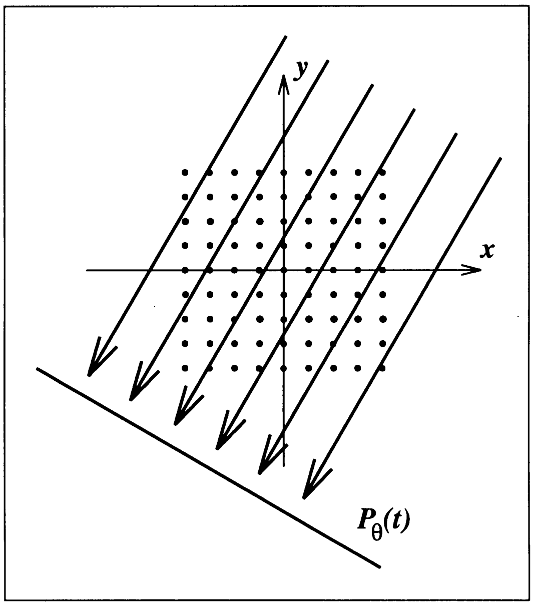
Figure 3.13: Decomposition of a cell into subcells. The projected rectilinear cell at left is into divided into the seven subcells sharing the same front and back face, an shown in the exploded view on the right. After Shirley and Tuchman [ST90].

The cell-by-cell processing techniques operate by dividing the cell into regions called *subcells* that share the same front and back face as shown in Figure 3.13. The illumination function is assumed to vary slowly within each subcell. Upson and Keeler's method [UK88] is a scanline algorithm that divides each scanline into spans along subcell boundaries. The techniques of Shirley and Tuchman [ST90], and Wilhelms and Van Gelder [WG91] consider the entire screen-space projection of a subcell at a time. They make the approximation that the illumination function varies bilinearly within each projected subcell. Illumination values are computed at each region's extremal points to determine the bilinear interpolation coefficients. Shirley and Tuchman's technique divides projections of tetrahedral cells into triangles, while the approach employed by Wilhelms and Van Gelder divides projections of rectilinear cells into quadrilaterals and triangles. Projections using either method can be computed using graphics hardware that renders Gouraud-shaded (r, g, b, α) polygons. Wilhelms and Van Gelder make the additional observation that when computing an orthographic view, the number, size and shape of projected subcells does not vary. Thus the decomposition into subcells need only be computed once. They call this technique *coherent projection*.

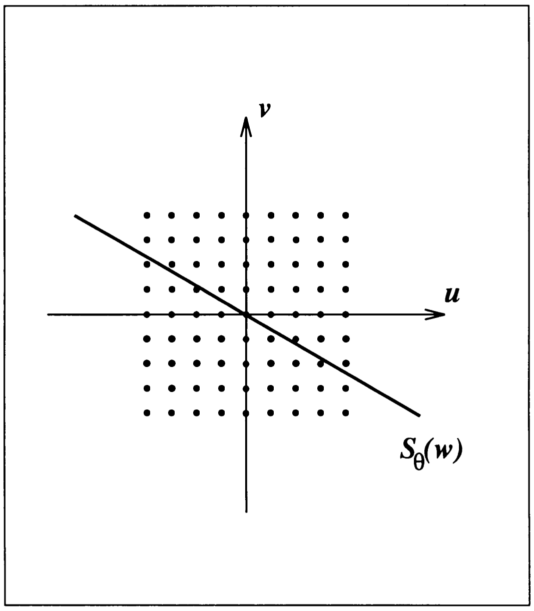
Cell-by-cell techniques derive their efficiency from the fact that the illumination function varies slowly within each subcell. They operate best when each cell projects onto many pixels. Therefore, they should be considered when screen resolution is higher than data resolution. The main drawback is that, with the exception of some experimental results [WG91], analysis of the error introduced by a piecewise bilinear approximation to the illumination function has not been performed. When using graphics hardware to render the subcells, error is also introduced due to the use of fixed-precision arithmetic.

3.3.5 Frequency Domain Algorithms

All the above algorithms for volume projection operate in the spatial domain. Recently, frequency-domain techniques have been introduced [Mal90,Lev92,Mal93, TL93]. These algorithms are based upon the Fourier slice theorem. This theorem relates projections of a datset to its Fourier transform, and is illustrated using a 2D example in Figure 3.14. We first define $P_\theta(t)$ to be a parallel projection of the 2D dataset, as shown in Figure 3.14a. The Fourier slice theorem states that the Fourier transform of $P_\theta(t)$ is equal to a slice through the Fourier transform of the data, inclined at an angle θ and passing through the origin, as shown in Figure 3.14b. Further details of the Fourier slice theorem may be found in textbooks such as the one by Kak and Slaney [KS88].



(a)



(b)

Figure 3.14: The Fourier slice theorem. (a) A parallel projection at angle θ in the spatial domain, $P_\theta(t)$, is the inverse Fourier transform of $S_\theta(w)$, a slice through the origin of the Frequency domain at angle θ (b).

Fourier volume rendering employs the three-dimensional Fourier slice theorem. The steps involved in computing a 2D projection, $P_{\theta,\phi}(s,t)$ of a sampled function

$f(x, y, z)$ using the Fourier volume rendering technique are:

1. Compute the discrete Fourier transform $F(u, v, w) = \mathcal{F}[f(x, y, z)]$.

2. Let $S_{\theta,\phi}(q, r)$ be a regular resampling of $F(u, v, w)$ at screen resolution, along the plane passing through the origin at angles θ, ϕ.

3. Let $P_{\theta,\phi}(s, t)$ be $\mathcal{F}^{-1}[S_{\theta,\phi}(q, r)]$.

Note that the use of this technique does not obviate the need for a reconstruction phase. Although the 3D function is never reconstructed in the spatial domain, it is reconstructed in the frequency domain when computing $S_{\theta,\phi}(s, t)$ in step 2.

The model of projection that is assumed by the Fourier slice theorem does not correspond to any of the volume illumination models presented in Chapter 2. It is simply the integral of the density function along the ray, i.e.,

$$I(a, b) = \int_a^b \rho(t)dt.$$

This expression may easily be transformed into the attenuation-only model presented in Equation 2.7 by taking the negative exponential of the result. As noted in Chapter 2, this model is extremely limited in application since it does not admit shading or occlusion. Thus, the projection model is a major drawback of the technique. Levoy has investigated techniques for improving the appearance of Fourier-rendered images by introducing depth-cueing and linear approximations to diffuse shading. Totsuka and Levoy implement these effects more efficiently by filtering in the frequency domain [TL93].

If we restrict our focus to computing the attenuation-only model, Fourier volume rendering is attractive because of the improvement in worst-case asymptotic complexity over both image-space and data-space algorithms. Assume a volume dataset with n^3 samples, and an image with m^2 pixels. The worst-case asymptotic complexity for data-space algorithms is $O(n^3)$ and $O(m^2 n)$ for image-space algorithms. Fourier rendering techniques require that the dataset first be transformed using a 3D DFT; an $O(n^3 \log n)$ preprocess that is performed only once per dataset. Projections are then be computed by computing $O(m^2)$ interpolations in the frequency domain, and computing an inverse 2D DFT in $O(m^2 \log m)$ time. If we consider the initial Fourier transform as an independent preprocess, then Fourier volume rendering can be considered an $O(m^2 \log m)$ process.

There are several drawbacks to using the technique as well. The speedups that are theoretically possible with Fourier volume rendering are difficult to achieve in practice, due to the cost of reconstruction in the frequency domain [Lev92,TL93]. The technique is also quite stringent in its preconditions: The density function must be sampled on a regular, rectangular grid and an orthographic projection model must be used. The technique is relatively new, and further development of the algorithm may overcome these disadvantages.

3.4 Summary

The work presented in this thesis builds upon the techniques presented in this chapter. The methods we explore assume that the inputs to the renderer are point-sampled functions of emission and absorption. Thus we do not assume any particular strategy for mapping. For the sample images, we used either maximum-likelihood or isosurface classification, along with gradient shading based on the

Phong model. For reconstruction, we use trilinear interpolation, although our methods will be general enough to handle any polynomial spline interpolation. The projection algorithms that we develop are extensions to ray casting. We felt that this was the best choice, because of its flexibility, and the ability to control error.

Chapter 4

Controlled-Precision Volume Rendering

The process of volume rendering scientific data involves many approximations. In some cases, the approximations are introduced intentionally, such as when the programmer seeks to increase processing speed. Other approximations are inherent in the model of illumination; for example there is no known analytic solution for the volume rendering integral of Equation (2.6). Approximations in the rendering process can lead to image artifacts that are not faithful to the data being rendered. At present, there are no tools to evaluate these artifacts quantitatively, so their presence and severity are evaluated subjectively. Often, the possibility of inaccuracy in rendering is not considered at all.

In this chapter, we introduce the notion of controlled-precision volume rendering. Algorithms with controlled precision differ from previous approaches in that they include both quantitative assessment and user control over the quality of the rendered image. Measures of quality are in terms of maximum deviation from the mathematically exact solution to the rendering model. Controlled-precision volume rendering has several advantages over existing algorithms:

- Using controlled-precision rendering, images can be computed that are guaranteed to have no visible rendering artifacts. Artifacts are image features that would not be present in the mathematically exact solution.
- Controlled-precision methods can efficiently compute images by halting computation when the image meets the desired precision.
- Controlled-precision methods can be used to compute reference solutions that can serve to evaluate other volume rendering algorithms quantitatively.

Guarantees of accuracy are essential in fields where scientists draw critical conclusions about their data based on the images that are produced. For example, in medical imaging, physicians must be able to determine if a small, anomalous dark spot on an image is indicative of disease or is merely a rendering artifact. In fact, physicians should not have to worry at all about the presence of rendering artifacts. Since controlled-precision techniques can provide reliable solutions to any specified accuracy, they can solve these problems by ensuring that images are computed to the accuracy of the display device.

In the first section of this chapter we present controlled-precision techniques for evaluating the model of volume illumination along a ray passing through a single voxel. In the second section, we extend these results to include intersections of rays with an entire volume. In the final section, we address the problem of accurate computation of pixel values.

4.1 Controlled-Precision Volume Integration

We now turn our attention to controlled-precision methods for solving the volume rendering integral, introduced as Equation (2.6) in Chapter 2.

$$I(a,b) = \int_a^b e^{-\int_a^t \alpha(v)\,dv}\, \epsilon(t)\, dt. \tag{4.1}$$

For convenience, we define $f(t)$ to be the integrand of Equation (4.1),

$$f(t) = e^{-\int_a^t \alpha(v)\,dv}\, \epsilon(t). \tag{4.2}$$

Note that each ray has a different function of α and ϵ and hence a different $f(t)$.

No general solution to $\int_a^b f(t)\,dt$ using a finite number of arithmetic operations and trigonometric functions is known. Thus, we must rely on some form of numerical quadrature. Although many standard techniques exist, previous approaches to solving this integral for volume rendering have not included error bounds. Many volume rendering algorithms evaluate the integral using a piecewise constant approximation — the domain of integration is broken into small segments where ϵ and α are approximated by constant values, k_ϵ and k_α [Lev88,Sab88,WG91]. Under this restriction, the integral of each of the segments reduces to a simple arithmetic expression,

$$\begin{aligned} I(a,b) &= \int_a^b f(t)\,dt \\ &= \int_a^b e^{-\int_a^t k_\alpha\,dv}\, k_\epsilon\, dt \end{aligned}$$

$$
\begin{aligned}
&= k_\epsilon \int_a^b e^{-k_\alpha [t-a]}\, dt \\
&= -\frac{k_\epsilon}{k_\alpha} \left[e^{-k_\alpha [b-a]} - e^{-k_\alpha [a-a]} \right] \\
&= \frac{k_\epsilon}{k_\alpha} \left[1 - e^{-k_\alpha [b-a]} \right]. \qquad (4.3)
\end{aligned}
$$

Note that the piecewise constant functions for which the technique is named are approximations to α and ϵ. The resulting approximation to the integrand is *not* piecewise constant.

If $\alpha(v)$ is symbolically integrable, a similar arithmetic expression may be found by dividing the domain of integration into small regions where ϵ is assumed to be related to α by a constant $k_{\alpha\epsilon}$ [MHC90,WG91]:

$$
\begin{aligned}
I(a,b) &= \int_a^b f(t)\, dt \\
&= \int_a^b e^{-\int_a^t \alpha(v)\, dv}\, k_{\alpha\epsilon}\alpha(t)\, dt \\
&= k_{\alpha\epsilon} \int_a^b e^{-\int_a^t \alpha(v)\, dv} \alpha(t)\, dt \\
&= -k_{\alpha\epsilon} \left[e^{-\int_a^b \alpha(v)\, dv} - e^{-\int_a^a \alpha(v)\, dv} \right] \\
&= k_{\alpha\epsilon} \left[1 - e^{-\int_a^b \alpha(v)\, dv} \right].
\end{aligned}
$$

The approach employed by Williams and Max [WM92] approximates α and ϵ in small regions by piecewise linear segments. The solution of the integral for each segment includes an error function that can be approximated to any specified precision. Another common technique that is applicable when α is symbolically integrable is to solve the inner integral of Equation (4.1) directly, and to approximate the outer integral using a deterministic quadrature formula [KV84,UK88, Kru90], or via Monte Carlo methods [KK89]. While all these strategies are known to increase in accuracy as the number of segments or samples increases, no precise error bounds have been given in the literature.

We present four techniques for computing approximate solutions to the volume rendering integral with bounded error. The first method, which we call *piecewise constant bounds* propagates bounds on α and ϵ using interval arithmetic. The remaining three methods are the trapezoid rule, Simpson's rule, and a power series method. In all cases, by bounding the magnitudes of the derivatives of f, we are able to obtain remainder terms that in turn bound the approximation error. The convergence properties of these algorithms vary considerably. The method of piecewise constant bounds is practical only for low-precision solutions. The trapezoid rule and Simpson's rule are most efficient for low- and medium-precision solutions. The power series method efficiently computes very high precision results, making it the method of choice for computation of highly accurate solutions. Such results may be used as reference standards by which the output of other algorithms may be compared. Note that much of the work in this section is based on standard numerical techniques. The details of standard methods are included for reasons of completeness.

4.1.1 The Integrand and its Derivatives

We begin by exploring the integrand and its derivatives. In this section, we focus our attention on individual voxels and defer discussion of collections of voxels until Section 4.2. We assume reconstruction by polynomial spline interpolation.

The functions $\alpha(v)$ and $\epsilon(t)$ are therefore polynomial. Since α is a polynomial, the exponent of Equation (4.2) evaluates to a polynomial, and we rewrite the integrand in a more convenient form,

$$f(t) = e^{p(t)}q(t) \tag{4.4}$$

where p and q are polynomials. In order to give a sense of the typical shapes of these functions, graphs of $p(t)$, $q(t)$ and $f(t)$ for two functions $\alpha(v)$ and $\epsilon(t)$ are given in Figure 4.1.

A prerequisite for computing solutions to the volume rendering integral with error bounds will be bounding the derivatives of $f(t)$ for all t in an interval $[a,b]$. In addition, the power series method will require exact values of the derivatives of $f(t)$ at $t = 0$. For this reason, we begin our discussion by presenting a formula for the n^{th} derivative of f, denoted $f^{(n)}$, in terms of the derivatives of p and q.

Theorem 4.1 (Derivatives of the Integrand) *Let $f(t) = e^{p(t)}q(t)$, where p and q are n times differentiable. Then*

$$f^{(n)}(t) = n! \sum_{j=0}^{n} \frac{q^{(j)}(t)\, A_{n-j}(t)}{j!} \tag{4.5}$$

where $A_{n-j}(t)$ is defined recursively by

$$\begin{aligned} A_0(t) &= e^{p(t)} && (4.6) \\ A_n(t) &= \frac{1}{n} \sum_{k=1}^{n} \frac{p^{(k)}(t) A_{n-k}(t)}{(k-1)!}. && (4.7) \end{aligned}$$

Proof: The theorem follows from the fact that

$$A_n(t) = \frac{d^n}{dt^n} \frac{e^{p(t)}}{n!},$$

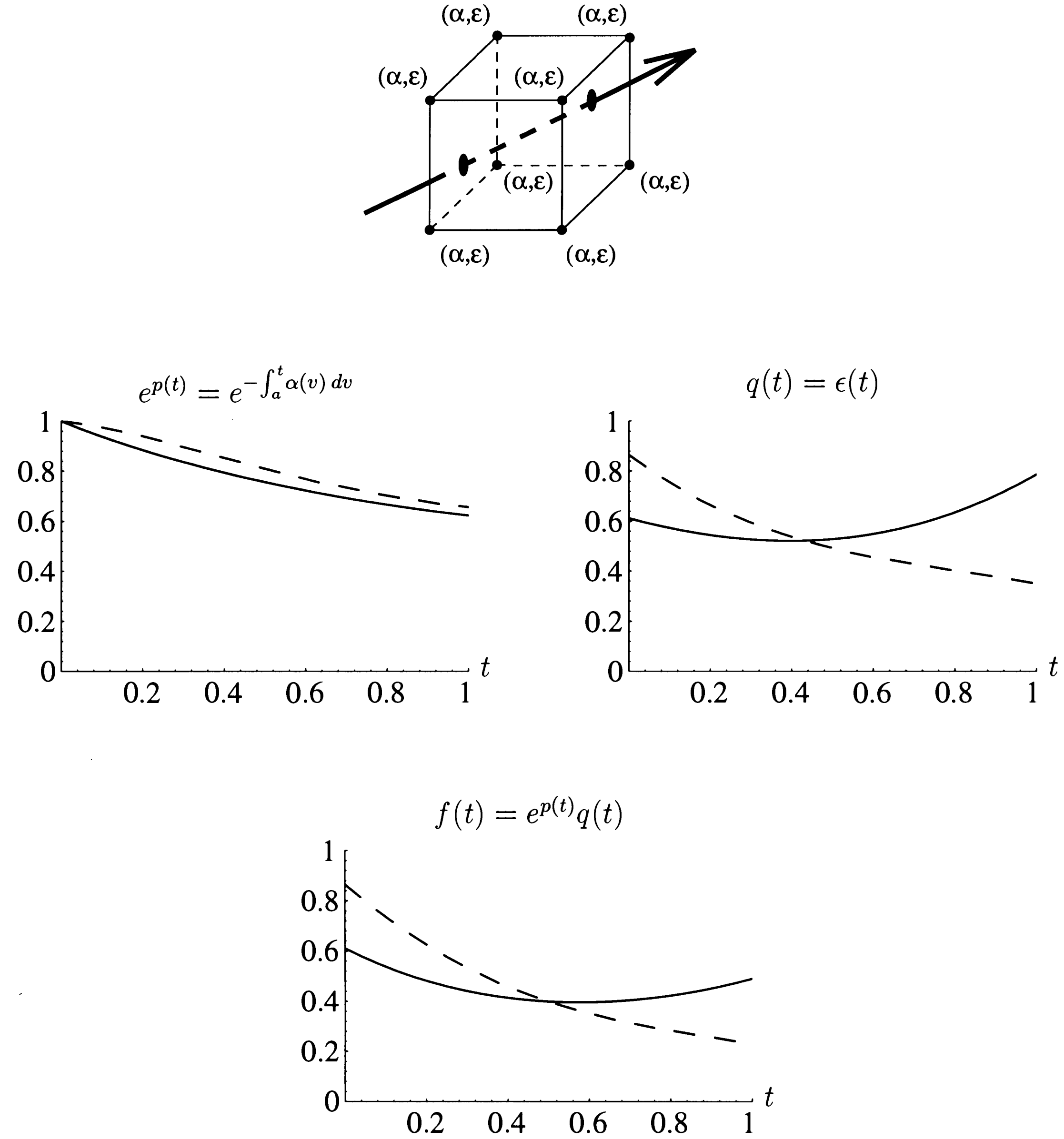


Figure 4.1: Graphs of the volume rendering integrand. (top) A ray is passed through a rectilinear voxel with samples of α and ϵ at the corners. (middle) This gives rise to functions of attenuation, $e^{p(t)}$, and emission, $q(t)$ along the ray. (bottom) These two functions are multiplied to obtain the volume rendering integrand, $f(t)$ (bottom). The solid and dashed lines present these functions for two different input voxels.

and from the formula for the n^{th} derivative of a product,

$$\frac{d^n}{dt^n} fg = \sum_{i=0}^{n} f^{(i)} g^{(n-i)} \begin{pmatrix} n \\ i \end{pmatrix}.$$

Equation (4.5) may then be verified by induction. □

Although equations (4.5)-(4.7) are formulated to give values of the derivatives of f at a point, they can be modified to give bounds on the derivatives of f over an interval. The only change necessary is to replace instances of derivatives of p and q with bounds on those derivatives over the interval. Derivatives of p and q are polynomials, so bounds can be obtained using the following rule. Given a polynomial $\mathcal{P}(t)$ of degree D with

$$\mathcal{P}(t) = \mathcal{P}_0 + \mathcal{P}_1 t + \ldots + \mathcal{P}_D t^D \tag{4.8}$$

an upper bound on $\mathcal{P}$ for $t \in [a, b]$ is given by

$$|\mathcal{P}_0| + |\mathcal{P}_1 c| + \ldots + |\mathcal{P}_D c^D| \tag{4.9}$$

where $c = \max(|a|, |b|)$. This is a coarse bound but has the advantage of requiring no root finding.

4.1.2 Piecewise Constant Bounds

In Equation (4.3), we showed that the volume rendering integral reduces to a simple arithmetic expression in ray segments where α and ϵ are constants. The method of piecewise constant bounds relies on the fact that this formula can also be used to form upper and lower bounds on $I(a, b)$, given piecewise constant bounds on $\alpha(u)$ and $\epsilon(u)$.

We begin by dividing the domain of integration, $[a, b]$, into n subregions, $[d_0, d_1]$, $[d_1, d_2]$, ..., $[d_{n-1}, d_n]$ as in Section 3.3.2. Let $\boldsymbol{\alpha}_k$ be an interval $[\alpha_k^-, \alpha_k^+]$ such that $0 \leq \alpha_k^- \leq \alpha(t) \leq \alpha_k^+$ for all t in $[d_{k-1}, d_k]$. We solve for an interval $\boldsymbol{T}_k = [T_k^-, T_k^+]$ bracketing the exact transparency T_k in the k^{th} subregion using standard interval arithmetic [Moo66].

$$\begin{aligned} \boldsymbol{T}_k &= e^{-\int_{d_{k-1}}^{d_k} \boldsymbol{\alpha}_k \, dv} \\ &= e^{-\boldsymbol{\alpha}_k (d_k - d_{k-1})}. \end{aligned} \tag{4.10}$$

The rules of interval arithmetic are quite simple to define in this case, since the two operations, integration and negative exponentiation, are monotonic. To get the upper bound on the interval for $\boldsymbol{T}_k$, we use the lower bound on $\boldsymbol{\alpha}_k$ and vice-versa. These rules correspond to the intuitive notion that transparency and absorption are inversely related.

Similarly, if we bracket the emission function ϵ within the segment $[d_{k-1}, d_k]$ by an interval $\boldsymbol{\epsilon}_k = [\epsilon_k^-, \epsilon_k^+]$, then it is also possible to solve for $\boldsymbol{I}_k = [I_k^-, I_k^+]$, an interval containing the exact intensity in the k^{th} subregion, I_k, as follows:

$$\begin{aligned} \boldsymbol{I}_k &= \int_{d_{k-1}}^{d_k} e^{-\int_{d_{k-1}}^{t} \boldsymbol{\alpha}_k \, dv} \, \boldsymbol{\epsilon}_k \, dt \\ &= \boldsymbol{\epsilon}_k (1 - \boldsymbol{T}_k)/\boldsymbol{\alpha}_k. \end{aligned} \tag{4.11}$$

Once again, the interval arithmetic corresponds to an intuitive notion. Maximum intensity is obtained using maximum emission and minimum absorption. Minimum intensity is obtained using minimum emission and maximum absorption. Mathematically, the computation of $\boldsymbol{I}_k$ is a little more complicated than for $\boldsymbol{T}_k$

due to the possibility of a zero in the denominator. Equation (4.11) can be broken into three cases:

$$\boldsymbol{I}_k = \begin{cases} \boldsymbol{\epsilon}_k(1-\boldsymbol{T}_k)/\boldsymbol{\alpha}_k & \text{if } \alpha_k^- \neq 0 \text{ and } \alpha_k^+ \neq 0 \\ \left[\epsilon_k^-(1-T_k^+)/\alpha_k^+,\ \epsilon_k^+(d_k-d_{k-1})\right] & \text{if } \alpha_k^- = 0 \text{ and } \alpha_k^+ \neq 0 \\ \boldsymbol{\epsilon}_k(d_k-d_{k-1}) & \text{if } \alpha_k^- = 0 \text{ and } \alpha_k^+ = 0. \end{cases} \tag{4.12}$$

We can then bound the overall intensity for the ray with an interval

$$\boldsymbol{I}(a,b) = \left[I^-(a,b), I^+(a,b)\right]$$

by rewriting Equation (3.4) with interval arithmetic:

$$\boldsymbol{I}(a,b) = \sum_{i=1}^{n} \boldsymbol{I}_i \prod_{j=1}^{i-1} \boldsymbol{T}_j. \tag{4.13}$$

Finally, by introducing the notation for the width and midpoint of an interval,

$$\begin{aligned} \| \boldsymbol{I}(a,b) \| &\equiv I^+(a,b) - I^-(a,b) \\ \overline{\boldsymbol{I}}(a,b) &\equiv \frac{1}{2}\left(I^+(a,b) + I^-(a,b)\right), \end{aligned}$$

we may state:

$$\left| I(a,b) - \overline{\boldsymbol{I}}(a,b) \right| \leq \frac{1}{2} \| \boldsymbol{I}(a,b) \| .$$

This means that the midpoint of the interval may be used an as approximation to $I(a,b)$, with a margin of error of $\pm\frac{1}{2} \| \boldsymbol{I}(a,b) \|$.

Up until now, we have assumed that across any segment, upper and lower bounds on the values of $\alpha(v)$ and $\epsilon(t)$ are available. If they are not, cheap bounds may be computed by sampling the function at the midpoint of the segment, and using bounds on the first derivative to determine the maximum range of the function within the segment. Using this method carries an additional benefit. The sample points may be used to compute a standard piecewise constant approximation to the intensity, which we call $\widetilde{I}(a, b)$. We have found that $\widetilde{I}(a, b)$ serves as a better estimate to $I(a, b)$ than $\overline{\boldsymbol{I}}(a, b)$, even though mathematically the margin of error is larger.

The method of piecewise constant bounds is valuable because it enables us to compute a reliable error term for the piecewise constant approximation. In Figure 4.2, we show the volume rendering integrand, and the approximation to the integrand which is used for the piecewise constant technique. Recall that the name of the method refers to how $\alpha(v)$ and $\epsilon(t)$ are estimated. The approximation to the integrand itself is not piecewise constant.

The method of piecewise constant bounds can be used to compute an approximation and remainder term once a partitioning of the domain of integration has been established. It is not truly a controlled-precision technique, since it currently lacks an algorithm for determining in advance a partitioning, given a desired level of accuracy. For situations where an image must be computed to a prescribed precision, the following methods are more suitable.

4.1.3 Trapezoid and Simpson's Rule

The trapezoid rule and Simpson's rule are two instances of Newton-Cotes quadrature formulæ [SB80,Act70]. These rules generate a polynomial spline approximant to the integrand, $f(t)$. The integral of the spline is easily computed, and is used

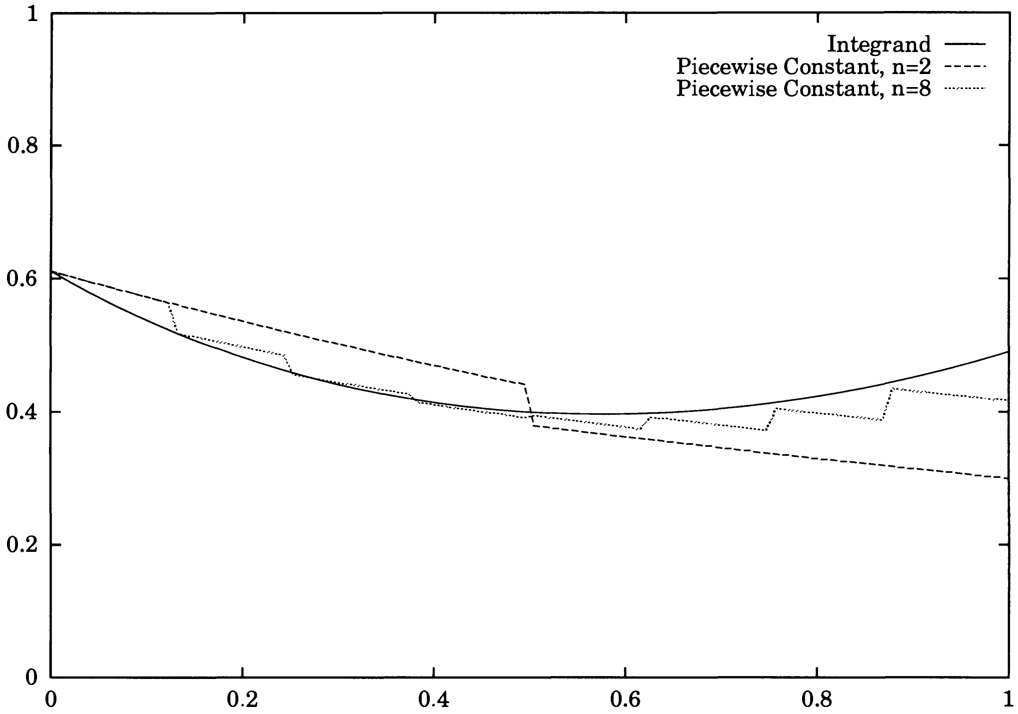


Figure 4.2: The piecewise constant approximation. The true integrand is compared to the approximation used for the piecewise constant method with two and eight segments. In this graph, the horizontal axis is distance along the ray, t.

as an approximation to $\int_a^b f(t)\,dt$. The spline is formed by evaluating $f(t)$ at $n+1$ evenly-spaced sample points, $a, a+h, a+2h, \ldots, a+nh = b$. If a linear spline is used, the resulting Newton-Cotes formula is the trapezoid rule. If a quadratic spline is used, the result is Simpson's rule. In the following subsections, we present these formulæ along with their corresponding remainder terms.

Trapezoid Rule

Using the trapezoid rule, we may write:

$$\begin{aligned} I(a,b) &= \int_a^b f(t)dt \\ &= T - r_T \end{aligned} \tag{4.14}$$

where T, the trapezoid approximation, is defined by

$$T = h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+ih)\right] \tag{4.15}$$

and r_T, the remainder, is defined by

$$r_T = \frac{b-a}{12} f''(c)\, h^2 \tag{4.16}$$

for some c between a and b. Approximations to a sample volume rendering integrand using the trapezoid rule with $n = 2$ and $n = 6$ are shown in Figure 4.3. Because c is unknown, r_T cannot be computed exactly. However, using the results from Section 4.1.1 we can compute an upper bound on $|f''(c)|$. Substituting this bound into Equation (4.16) yields a bound on the magnitude of r_T.

To obtain a solution to the volume rendering integral to within a given tolerance δ, we set $r_T = \delta$ in Equation (4.16) and solve for the appropriate step size h. This approach is perhaps the simplest method for controlled-precision integration, requiring very little computation in addition to the basic trapezoid rule. Pseudo code is given in Figure 4.4.

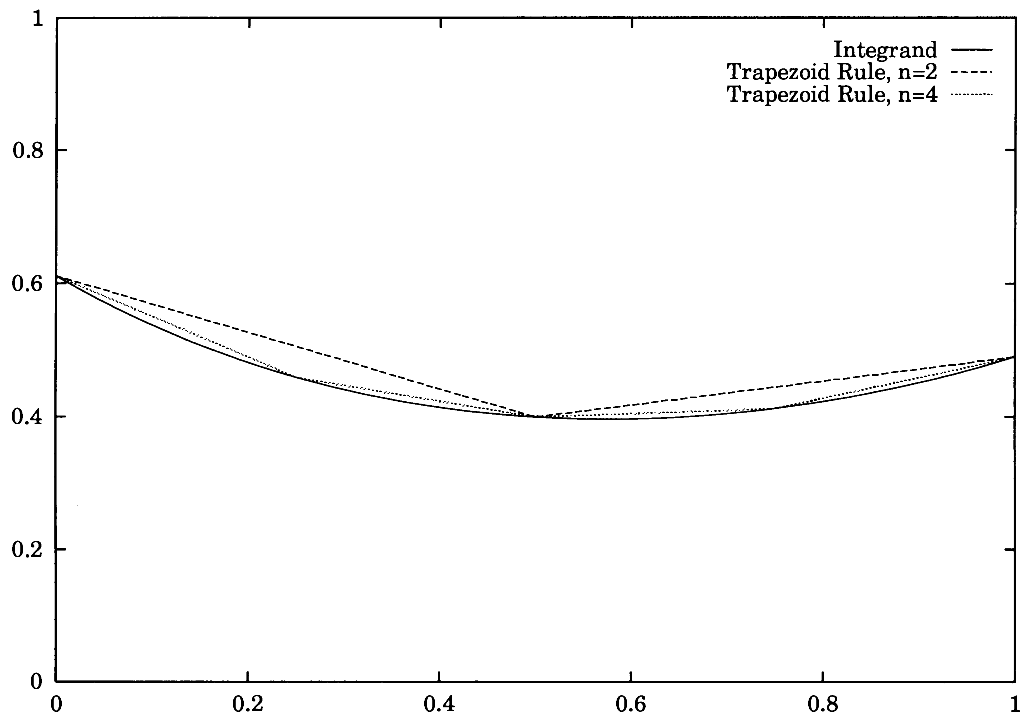


Figure 4.3: The trapezoid rule. The true integrand is compared to the approximation used for the trapezoid rule with two and four segments. In this graph, the horizontal axis is distance along the ray, t.

Simpson's Rule

Using Simpson's rule, we may write:

$$I(a,b) = \int_a^b f(t)dt \tag{4.17}$$

$$= S - r_S \tag{4.18}$$

where S, the Simpson's rule approximation, is defined by

$$S = \frac{h}{3}\left[\left(4\sum_{i=1}^{n/2} f\left(a+(2i-1)h\right)\right) + \right.$$

<u>*TrapezoidRule*(p, q, a, b, δ)</u>
[*Given polynomials p and q, compute the volume rendering integral from a to b within tolerance δ using the trapezoid rule.*]
$\beta \leftarrow$Upper bound on $f''(c)$ for $c \in [a, b]$
$h' \leftarrow \sqrt{12\delta / ((b-a)\beta)}$ [*h' is largest step size that will ensure δ*]
$n \leftarrow \lceil (b-a)/h' \rceil$ [*Divide range into regular steps of size $\leq h'$*]
$h \leftarrow (b-a)/n$ [*h is the step size for this partition*]
Compute T using Equation (4.15)
return T

Figure 4.4: Pseudo-code for the trapezoid rule algorithm.

$$2\left(\sum_{i=1}^{n/2} f\left(a+2ih\right)\right)+f\left(a\right)+f\left(b\right)\Bigg]$$

and r_S, the remainder, is defined by

$$r_S = \frac{b-a}{180} f^{(4)}(c)\, h^4 \tag{4.19}$$

for some c between a and b.

Here n must be even. Approximations to the integrand using Simpson's rule with $n = 2$ and $n = 4$ are shown in Figure 4.5. As with the trapezoid rule, Equation (4.19) may be used to compute a bound on the approximation error for a given step size h, or to compute the step size necessary to ensure a given accuracy. The pseudo-code in Figure 4.4 may be used with only slight modification.

4.1.4 Power Series

We have developed another method for controlled precision volume integration based on a power series expansion. It is obtained from Taylor's theorem coupled

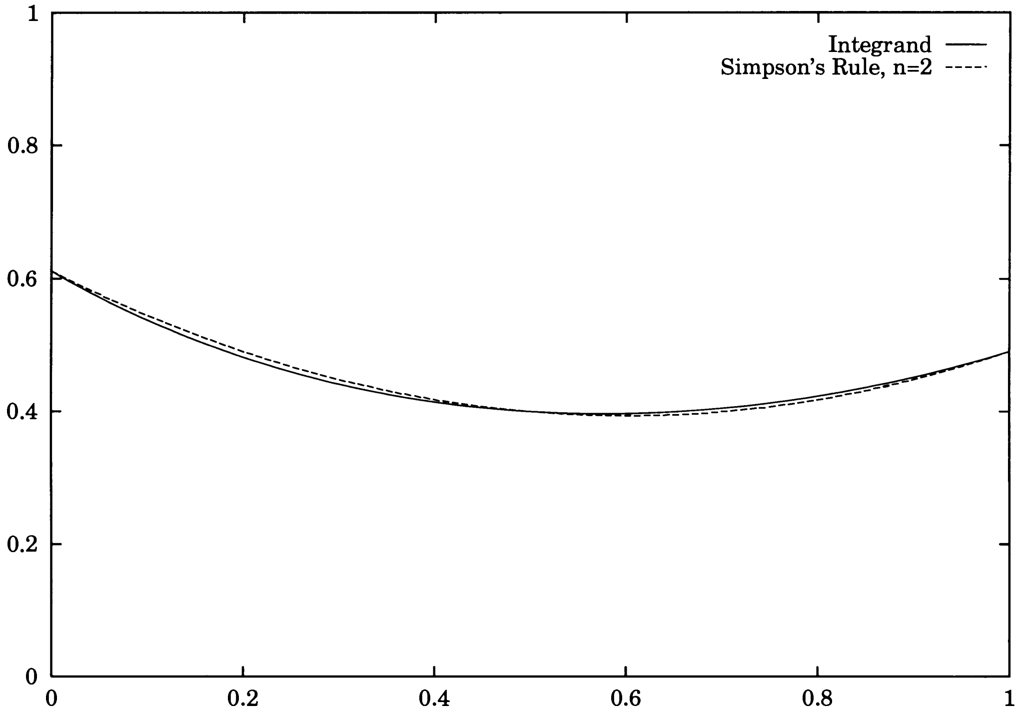


Figure 4.5: Simpson's rule. The true integrand is compared to the approximation used for Simpson's rule with two segments. In this graph, the horizontal axis is distance along the ray, t.

with Lagrange's form of the remainder:

Theorem 4.2 (Taylor-Lagrange) *Let f be a function that is C^{n+1} on the interval $[0,t]$. Then $f(t)$ can be expressed as the sum of an n-term power series $P_n(t)$ and a remainder term $R_n(t)$,*

$$f(t) = P_n(t) + R_n(t) \tag{4.20}$$

where

$$P_n(t) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!}\, t^k \tag{4.21}$$

$$R_n(t) = f^{(n+1)}(c)\,\frac{t^{n+1}}{(n+1)!} \tag{4.22}$$

for some $c \in [0, t]$. □

Proofs of this theorem may be found in most introductory calculus texts, for example the one by Thomas and Finney [TF79]. Approximations to the volume rendering integrand using a two term and 10-term power series are shown in Figure 4.6.

We apply Theorem 4.2 to the problem of approximating $I(a, b)$ by letting f be the volume rendering integrand and writing

$$I(a,b) = I_n(a,b) + \int_a^b R_n(t)dt. \tag{4.23}$$

where the n-term approximation $I_n(a, b)$ is given by

$$I_n(a,b) = \int_a^b P_n(t)dt.$$

Because R_n is bounded above for all n, and

$$\lim_{n\to\infty} R_n(t) = 0$$

for all t, as shown in Appendix B, the integral of R_n in Equation (4.23) must also go to zero. Thus, as n increases, $I_n(a, b)$ provides an arbitrarily close approximation to $I(a, b)$. We derive the series and remainder terms for the volume rendering integral in the following subsections.

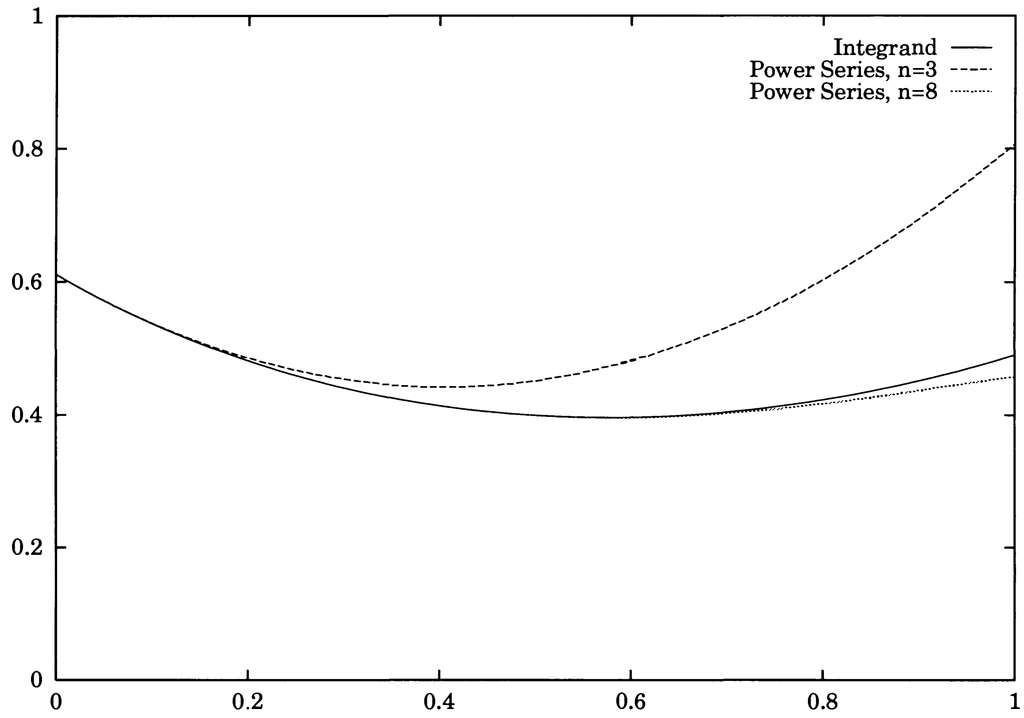


Figure 4.6: The power series approximation. The true integrand is compared with its three-term and eight-term Taylor series approximation. In this graph, the horizontal axis is distance along the ray, t.

The Series Term

The first step in forming the series term is to substitute the expression for $f^{(n)}$, derived from Theorem 4.1, into Equation (4.21), giving

$$P_n(t) = \sum_{k=0}^{n} \sum_{j=0}^{k} \frac{q^{(j)}(0)\, A_{k-j}(0)}{j!}\, t^k \tag{4.24}$$

where

$$A_0(0) = e^{p(0)} \tag{4.25}$$

$$A_n(0) = \frac{1}{n} \sum_{k=1}^{n} \frac{p^{(k)}(0) A_{n-k}(0)}{(k-1)!}. \tag{4.26}$$

Since p and q are polynomials, these equations may be simplified. Let p and q be defined by

$$\begin{aligned} p(t) &= p_0 + p_1\,t + \ldots + p_N\,t^N \\ q(t) &= q_0 + q_1\,t + \ldots + q_M\,t^M. \end{aligned}$$

For any polynomial Δ defined as in Equation (4.8), we have

$$\mathcal{P}^{(n)}(0) = \begin{cases} n!\mathcal{P}_n & \text{if } n \leq D \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, Equations (4.24)-(4.26) yield

$$P_n(t) = \sum_{k=0}^{n} \sum_{j=0}^{\min(k,M)} q_j\,A_{k-j}(0)\,t^k \tag{4.27}$$

$$A_0(0) = e^{p_0} \tag{4.28}$$

$$A_n(0) = \frac{1}{n} \sum_{k=1}^{\min(n,N)} k\,p_k A_{n-k}(0). \tag{4.29}$$

Finally, integrating $P_n(t)$ we obtain

$$I_n(a,b) = \sum_{k=0}^{n} \sum_{j=0}^{\min(k,M)} q_j\,A_{k-j}(0)\,\frac{b^{k+1} - a^{k+1}}{k+1} \tag{4.30}$$

which approximates $I(a,b)$. Pseudo code for computing $I_n(a,b)$ is given in Figure 4.7. Since M and N are fixed by the degrees of p and q, the complexity of this algorithm is $O(n)$. The accuracy of this approximation is provided by the remainder term.

The Remainder Term

In this section we derive a bound on $R_n(t)$, the remainder term introduced in Theorem 4.2, when the integrand f is given by Equation (4.4).

<u>*Approximate_I*</u>(p, q, a, b, n)

[*Given polynomials p and q, compute the volume rendering integral from a to b via an n-term power series.*]

$A_0 \leftarrow e^{p_0}$

$I_n \leftarrow q_0\, A_0\, (b - a)$

for $k \leftarrow 1$ **to** n **do**

$A_k \leftarrow Compute\_A(p, k, A_0, \ldots, A_{k-1})$

$I_n \leftarrow I_n + \sum_{j=0}^{\min(k,M)} q_j A_{k-j} \left(b^{k+1} - a^{k+1}\right) / k + 1$

end for

return I_n

<u>*Compute_A*</u>$(p, k, A_0, \ldots A_{k-1})$

[*Given polynomial p and terms* $A_0 \ldots A_{k-1}$, *compute* A_k *as in Equation (4.29).*]

$A_k \leftarrow \frac{1}{k} \sum_{j=1}^{\min(k,N)} j\, p_j\, A_{k-j}$

return A_k

Figure 4.7: Pseudo-code for computing $I_n(a, b)$.

As outlined in Section 4.1.1, a bound on $f^{(n)}(t)$ for $t \in [a, b]$ can be obtained from Equations (4.5)-(4.7). We begin by substituting this bound for $f^{(n+1)}$ into Equation (4.22), giving

$$R_n(t) \;=\; t^{n+1} \sum_{j=0}^{\min(n+1,M)} \frac{\overline{q}^{(j)}\,\overline{A}_{n+1-j}}{j!}, \tag{4.31}$$

where $\overline{q}^{(j)}$ is the upper bound on the j^{th} derivative of q from Equation (4.9). $\overline{A}_{n+1-j}$ is an upper bound on A_{n+1-j} computed via the recurrence relation

$$\overline{A}_0 \;=\; e^{\overline{p}} \tag{4.32}$$

$$\overline{A}_n \;=\; \frac{1}{n} \sum_{k=1}^{\min(n,N)} \frac{\overline{p}^{(k)}\overline{A}_{n-k}}{(k-1)!}. \tag{4.33}$$

The remainder term for the n-term power series is formed by integrating $R_n(t)$ from a to b,

$$\int_a^b R_n(t)\,dt \;=\; \sum_{j=0}^{\min(n+1,M)} \frac{\overline{q}^{(j)}\,\overline{A}_{n+1-j}}{j!}.$$

Finally, using Equation (4.23),

$$|I(a,b) - I_n(a,b)| \le \left|\frac{b^{n+2} - a^{n+2}}{n+2}\right| \sum_{j=0}^{\min(n+1,M)} \frac{\overline{q}^{(j)}\,\overline{A}_{n+1-j}}{j!}.$$

Pseudocode to compute $|I(a,b) - I_n(a,b)|$ is given in Figure 4.8.

The form of the expression above does not provide a means of computing *a priori* the number of terms necessary to compute $I(a,b)$ to attain a given accuracy δ. Therefore, to use the power series, we accumulate terms while simultaneously evaluating the remainder term. The loop exits when the desired tolerance is achieved. Termination is guaranteed by the convergence proof in Appendix B. The loop may be coded efficiently by precomputing bounds on the derivatives of p and q.

<u>*Compute_Remainder*</u>(p, q, a, b, n)

[*Given polynomials p and q, compute a remainder term, bounding* $|I(a,b) - I_n(a,b)|$.]

Compute $\overline{p}$, bounds on the derivatives of p [*See Section 4.1.1.*]

Compute $\overline{q}$, bounds on the derivatives of q

$\overline{A}_0 = e^{\overline{p}^{(0)}}$

for $k \leftarrow 1$ **to** $(n+1)$ **do**

$\quad \overline{A}_k \leftarrow$ *Compute_A_Bound*$(\overline{p}, k, \overline{A}_0, \dots, \overline{A}_{k-1})$

$\quad$ **end for**

$$r \leftarrow \left|\frac{b^{n+2} - a^{n+2}}{n+2}\right| \sum_{j=0}^{\min(n+1,M)} \frac{\overline{q}^{(j)}\,\overline{A}_{n+1-j}}{j!}$$

return r

<u>*Compute_A_Bound*</u>$(\overline{p}, k, \overline{A}_0, \dots \overline{A}_{k-1})$

[*Given bounds on polynomial p and terms* $\overline{A}_0 \dots \overline{A}_{k-1}$, *compute* $\overline{A}_k$ *as in Equation (4.33).*]

$$\overline{A}_k \leftarrow \frac{1}{k} \sum_{j=1}^{\min(k,N)} \frac{\overline{p}_j\,\overline{A}_{k-j}}{(k-1)!}$$

return $\overline{A}_k$

Figure 4.8: Pseudo-code for computing a remainder term.

$\underline{PowerSeries}(p, q, a, b, \delta)$
[*Given polynomials p and q, compute the volume rendering integral from a to b with tolerance δ using a power series.*]
Compute $\overline{p}$, bounds on the derivatives of p [*See Section 4.1.1.*]
Compute $\overline{q}$, bounds on the derivatives of q
$A_0 \leftarrow e^{p_0}$
$\overline{A}_0 = e^{\overline{p}^{(0)}}$
$I_n \leftarrow q_0\, A_0\, (b - a)$
$k \leftarrow 1$
repeat
  $A_k \leftarrow \mathit{Compute\_A}(p, k, A_0, \ldots, A_{k-1})$
  $\overline{A}_k \leftarrow \mathit{Compute\_A\_Bound}(\overline{p}, k, \overline{A}_0, \ldots, \overline{A}_{k-1})$
  $I_n \leftarrow I_n + \sum_{j=0}^{\min(k,M)} q_j A_{k-j} \left(b^{k+1} - a^{k+1}\right) / k + 1$
  $r \leftarrow \left| \frac{b^{n+2} - a^{n+2}}{n+2} \right| \sum_{j=0}^{\min(n+1,M)} \frac{\overline{q}^{(j)}\, \overline{A}_{n+1-j}}{j!}$
until $r < \delta$
return I

Figure 4.9: Pseudo-code for the complete power series.

Iterative techniques are then used to evaluate $A_n(t)$ and $\overline{A}_n$, as shown in pseudo code in Figure 4.9.

4.1.5 Discussion and Results

In order to evaluate the relative performance of the controlled-precision integration methods, each of the approaches were tested using a random distribution of rays passing through a single voxel whose emission and absorption values at each corner were randomly distributed over the interval $[0, 1]$. The time to compute 10,000 such integrals to a specified precision was measured. The results are summarized in a Figure 4.10, where computation time is plotted against precision. Note that precision is measured in bits, so the horizontal scale is logarithmic.

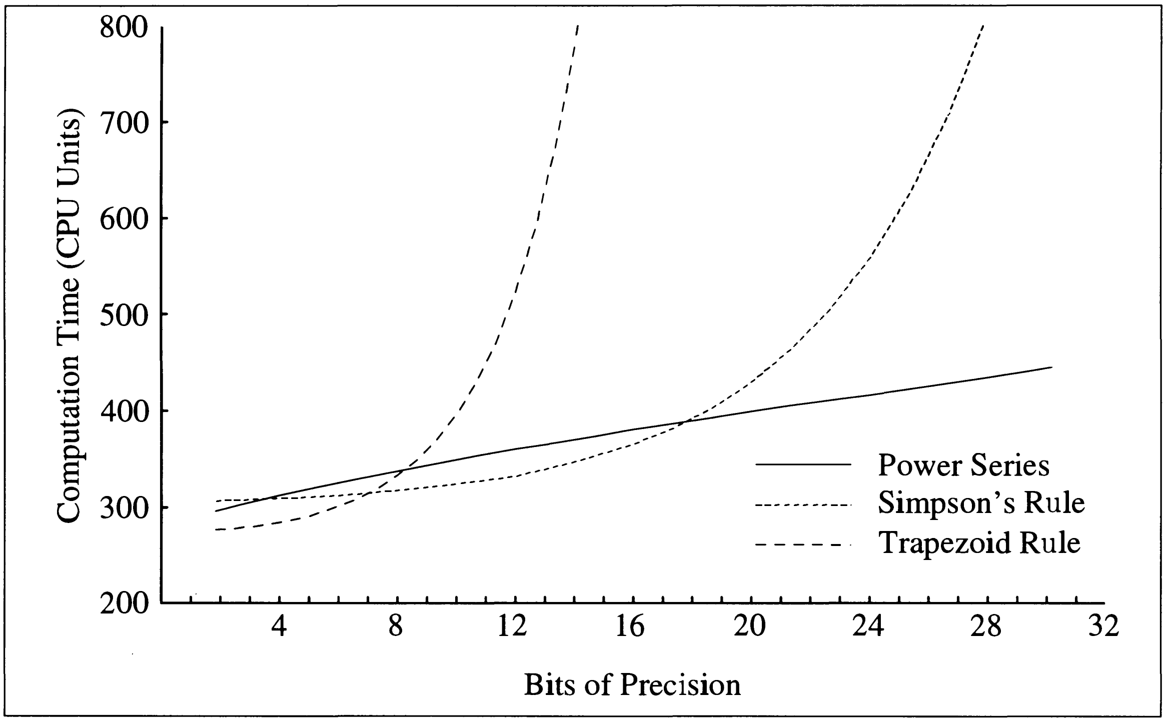


Figure 4.10: Performance on 10000 test integrals.

Figure 4.10 shows that the three quadrature methods exhibit very different convergence properties, and that each method has a range where its performance is superior. For low- and medium-precision results, trapezoid and Simpson's rule are faster than the power series, although the relative difference in computation time is small. Trapezoid rule convergence is only quadratic, while Simpson's rule is fourth order. However, the performance of the trapezoid rule is superior at low precision because it needs to bound only the second derivative of the integrand, in contrast to Simpson's, which requires bounds on the fourth derivative. The power series requires several terms before beginning to converge, making it less efficient for lower precisions. Convergence is exponential, however, which enables it to compute very high precision solutions that would not be practical with the other methods.

Since most computer graphics displays have an 8-bit dynamic range, one might conclude from the graph in Figure 4.10 that Simpson's rule is the method of choice. This is not always the case, however. First, the precision reported in Figure 4.10 is absolute, whereas most volume-rendered images are scaled relative to the brightest pixel value in the image. Relative scaling will shift the curves along the precision axis, thus changing the location of the crossover points. Secondly, the precision reported in Figure 4.10 is for a single voxel. As described in the next section, when the precision along an entire ray is considered, the errors from each voxel accumulate; thus, the tolerance for a single voxel may have to be considerably tighter than the tolerance for the ray itself.

The methods described in this section compute guaranteed bounds on the error, but not necessarily the tightest bounds possible. When a given solution is significantly more accurate than can be proven with these tools, computation time is wasted in refining the results until they are provably below tolerance. There is a tradeoff in utilizing tighter bounds: they are more expensive to compute, but they may allow the quadrature algorithm to compute a result to the same precision with less work. Investigation of other methods of bounding these derivatives is warranted.

Further work is needed on the applicability of other quadrature rules, such as

Table 4.1: Quadrature rules and their information requirements. The function $f(t)$ is the volume rendering integrand, $e^{-\int_a^t \alpha(v)\,dv}\,\epsilon(t)$.

Conditions	Piecewise Constant Bounds	Trapezoid, Simpson's Rules	Power Series Method
Bounds on $\alpha(v), \epsilon(t)$	X		
$f(t)$ Computable		X	
$f^{(n)}(0)$ Computable			X
Bounds on $f^{(n)}(t)$		X	X

Gaussian quadrature, to the volume rendering integral. The power series could also be converted to a continued-fractions format, in hopes of achieving better convergence properties [PFTV88].

While the derivations we have presented are specific to polynomial α and ϵ, the underlying quadrature rules are more general. For example, Newton-Cotes methods require only evaluation of the integrand at points and bounds on derivatives in the domain of integration. A summary of the requirements for each algorithm is given in Table 4.1. This work should be extended to exploring more general forms of α and ϵ.

4.2 Controlled-Precision Ray Casting

Since we are assuming polynomial spline interpolation, the controlled-precision quadrature techniques described above can only be used for computing values of $I(a, b)$ within a single cell. The trapezoid rule, Simpson's rule and the power series method require bounds on derivatives to compute remainder terms. These bounds are not available across cell boundaries because the interpolated function is discontinuous. In this section, we consider how these quadrature rules can be used in a simple ray-casting volume rendering algorithm to compute $I(a, b)$ across multiple cells to any specified precision. In Chapter 5 we will develop a more efficient hierarchical algorithm.

We begin by partitioning the ray along its intersections with the voxel grid. Where these intersections correspond to discontinuities of α and ϵ, this strategy is akin to the discontinuity meshing algorithms for radiosity [LTG92]. The algorithm we use for partitioning the ray is the one developed by Snyder and Barr for ray-casting an axis-aligned rectilinear grid [SB87]. The grid spacing must remain constant along each axis, but may vary between the axes. Their algorithm is based on the following observations:

1. All intersection points occur on an x, y, or z plane of the volume grid.

2. A ray intersects each of these planes at regular intervals.

Let t be the distance parameter along the ray. During the initialization phase, the t values of the first intersection of the ray with the bounding x, y and z planes of the data are computed and stored as t_x, t_y, t_z. The values Δ_x, Δ_y and Δ_z are set to the distance along the ray between data slices along each of the axes. The algorithm then iterates by choosing the smallest of $\{t_x, t_y, t_z\}$ as the next intersection with grid. That value is then incremented by the appropriate Δ value. The algorithm therefore identifies all voxels in a ray's path and the distances along the ray to each of these intersections. The algorithm requires no multiplications or divisions in the inner loop. An iteration of the this algorithm is shown in Figure 4.11.

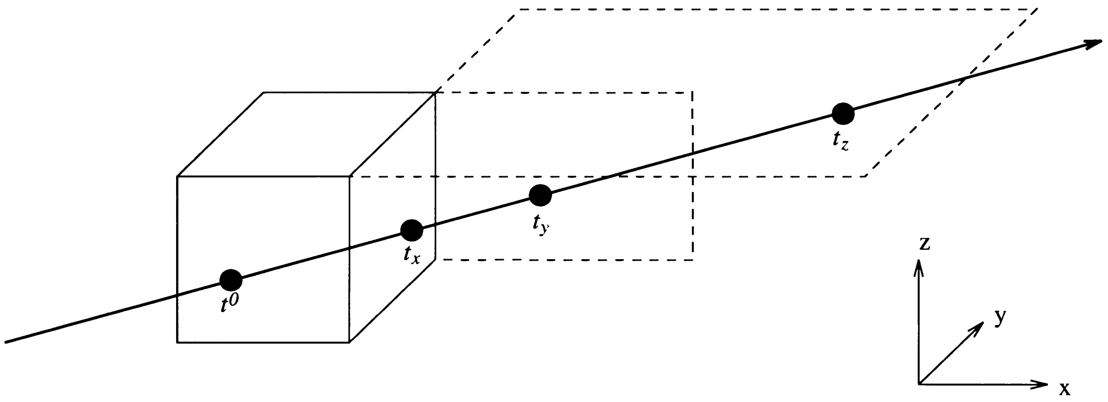


Figure 4.11: Tracing a ray through a grid using the Snyder and Barr algorithm [SB87]. The ray enters the cell at t^0. The next intersection points with a grid plane along each of the axes are t_x, t_y and t_z. The smallest of these values, in this case t_x, is the ray's next grid intersection. The algorithm proceeds to the next cell and increments t_x by Δ_x.

Recall from Section 4.1.2 that if intervals of intensity, $\boldsymbol{I}_k$, and transparency, $\boldsymbol{T}_k$, are given for each segment k, then bounds on $I(a, b)$ can be computed using interval arithmetic,

$$\boldsymbol{I}(a, b) = \sum_{i=1}^{n} \boldsymbol{I}_i \prod_{j=1}^{i-1} \boldsymbol{T}_j. \tag{4.34}$$

We can obtain $\boldsymbol{I}_k$ using the quadrature techniques from the previous section by adding and subtracting the remainder terms from the estimates. Since absorption is polynomial, the transparency integral reduces to a simple arithmetic expression. Neglecting roundoff error, we can assume that $\boldsymbol{T}_k$ can be computed exactly. Therefore we write it as an ordinary scalar, T_k. If the user wishes to consider roundoff error, the more sophisticated approach detailed in Chapter 5 can be used to properly handle intervals of $\boldsymbol{T}_k$.

To compute a ray with an error tolerance of δ, we may distribute the error evenly across all the segments allocating a tolerance

$$\delta_i = \frac{\delta}{n}$$

to each of the additive terms in Equation (4.34),

$$\boldsymbol{I}_i \prod_{j=1}^{i-1} T_j.$$

Since each of the T_k's are in the range [0,1], the value of the product will be a fraction between zero and one. Thus, the error tolerance for the i^{th} term may be met by computing I_k with a tolerance of

$$\frac{\delta_i}{T_1 T_2 \ldots T_{i-1}}$$

which will be larger than or equal to δ_i. A front-to-back algorithm that implements this strategy is shown in Figure 4.12. Line (1) is an application of a quadrature rule such as the ones introduced in Section 4.1. The choice of quadrature rule can be based on the value of δ_i to obtain the fastest convergence for the given tolerance.

The algorithm developed in this section was used to compute the 256 $\times$ 256 image of the pelvis dataset shown in Figure 4.13a. Each ray was computed to the nearest grey level. For comparison, Figure 4.13b shows the same scene rendered

```
Cast_Ray(r⃗, a, b, δ)
[Compute I(a,b) along a ray r⃗ to within ±δ.]
T = 1 [Initialize transparency]
I = 0 [Initialize intensity]
Partition r⃗ into N ray–voxel intersections
δ_i = δ/N
for each ray–voxel intersection i from front to back do
    Compute T_i      [Equation (3.2)]
    Compute I_i to error tolerance δ_i      [Section (4.1)]
    I ← I + I_i × T
    T ← T × T_i
    δ_i ← δ_i/T_i
end for
return I
```

Figure 4.12: Pseudo-code for controlled-precision ray casting.

using the piecewise constant approximation. Piecewise constant regions were equal in length to half the voxel grid spacing. The images appear quite similar. Figure 4.13c is a subtraction and rescaling of the Figures 4.13a and 4.13b. Neutral grey indicates an exact match to 8-bit precision. The brightest and darkest areas indicate a difference of 20 grey levels – a significant disparity. When the step size of the piecewise constant algorithm was set to one-tenth of the grid spacing, the two images agreed to 8 bits of precision. A radial beating artifact in the output of the piecewise constant algorithm is enhanced in Figure 4.13c. This is aliasing caused by undersampling along the ray axis. The coherence of the artifact arises from sample points that are placed at uniform distances from the eyepoint.

4.3 Integration over pixel area

The approach described in the previous section correctly computes the value of $I(a, b)$ along individual rays to any specified precision. Recall from Section 3.3.1 that ray casting point-samples the illumination function across the image plane. By itself, guaranteeing the accuracy of each ray does not guarantee the accuracy of the final image. This is due to the fact that we have no bound on the variation of the illumination between the sample points. Error due to point-sampling of the image plane can be severe. If an object falls between rays, it will not contribute to the final image at all.

Our preliminary investigations into bounding screen-space error concentrated on obtaining bounds on the illumination function across screen-space disks of a given radius r. A practical solution to this problem enables a controlled-precision

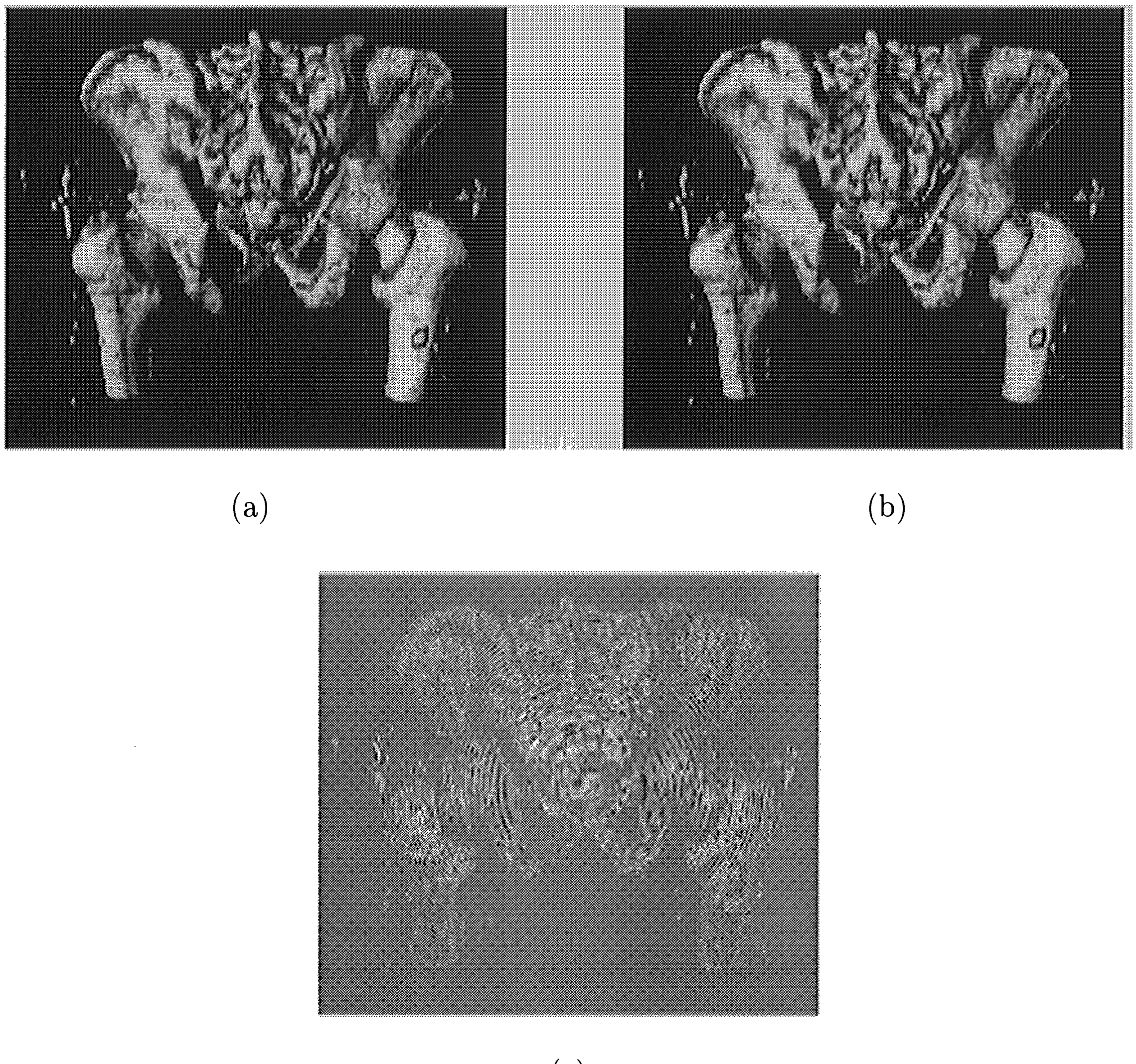


Figure 4.13: Controlled-precision ray casting. Image (a) was computed using controlled-precision techniques to within one grey level at each ray. Image (b) was computed using the piecewise constant approximation with a step size of one-half the maximum grid spacing. Image (c) is a subtraction of images (a) and (b), highlighting the errors in image (b).

algorithm, since a screen-space region could be approximated with smaller and smaller disks until a specified error tolerance is reached. Our initial approach operated by sampling absorption and emission at the center of the disk and considering the maximum variation of the functions across the requisite volume. The resulting algorithm is impractical, since the bounds computed in this manner are far too coarse.

Obtaining tight bounds for the variation of the illumination function is difficult because there are no guarantees that the function will be slow-varying. For example, two rays in any finite neighborhood may intersect a completely different set of voxels. A tempting approach would be to consider all possible discontinuities of the illumination function by projecting the voxel grid onto the screen. However, this technique would be impractical because it would subdivide the screen into $O(n^3)$ individual regions, where n is the linear resolution of the data.

4.4 Summary and Future Directions

The work presented in this chapter represents a step toward more rigorous error analysis of volume rendering techniques. We first presented controlled-precision techniques for computing the emission-absorption model along rays that pass through a single interpolation cell. These results were extended to form an algorithm that computes illumination along rays passing through collections of interpolation cells. A theoretically valid algorithm has also been presented for controlled-precision calculation of pixel values.

There remains much work to be done in this area. A practical algorithm for bounding error in screen-space is an important next step. There are also other sources of error in volume-rendered images that we have not touched upon. These include reconstruction error and uncertainty in measured data values. Errors introduced by approximations such as fixed-precision arithmetic and splatting have yet to be explored.

Throughout this chapter, we have been assuming an objective error metric, one that measures deviation from an ideal image. While these metrics are easy to fit into a mathematical framework, their correspondence to how we perceive error in images is unknown. Study of perceptual effects of rendering error on the interpretation of volume-rendered images is warranted.

The abstract nature of volume rendering poses some difficulty in performing any type of error analysis. One abstraction is the process of mapping raw data values to functions of emission and absorption. This process is currently under user control. Clearly errors can occur during this assignment phase. But since we have no ideal model of how mapping should take place, we are currently unable to quantify errors introduced at this stage.

Chapter 5

Efficiency Issues

Volume rendering is often used in time-critical applications, such as medical imaging, that require fast imaging of data. Even in situations that don't require immediate access to final results, fast-feedback images are often helpful when testing rendering parameters such as camera position, lighting and color. While improvements in processor speed and memory chip capacity have resulted in a continual increase in the power of graphics workstations, a simultaneous explosion in dataset size has more than kept pace. Advances in scanning technology result in an $O(n^3)$ increase in data size for every $O(n)$ improvement in linear resolution. Algorithmic improvement is necessary to continue to provide users with reasonable rates of feedback.

While we wish to provide fast feedback, we don't want to compromise on our goals of presenting information clearly and accurately. Feedback that confuses or misleads the user is certainly not an asset. In this chapter, we consider issues regarding efficient computation of volume-rendered images. In the first section, we address techniques for fast computation of perspective projections of volume data. This is important since many of the faster algorithms presented in Section 3.3 could only compute orthographic projections. In the second section we introduce an adaptive error bracketing scheme that attempts to compute ray values to any specified precision using minimum computation.

5.1 Efficient Perspective Projection

The majority of images produced using volume rendering have employed orthographic projection. While quite effective in many circumstances, orthographic views have, in a sense, held scientists at an infinite distance from their data. Orthographic projections often result in depth ambiguities and reversals of apparent depth as is shown in the surface rendering of Figure 5.1. Since our visual system is tuned to comprehend the perspective views projected onto our retinas, scene geometry and depth are more readily apparent in simulated perspective views. In addition, data flythroughs and lens effects become available to aid in the comprehension of spatial relationships within the data when perspective is used.

The widespread use of perspective projection has been prevented by a lack of efficient algorithms. The fastest rendering algorithms such as splatting and coherent projection exploit the coherence of parallel viewing rays. Frequency-domain rendering techniques are currently constrained to the parallel projection model used by the Fourier slice theorem. Ray-casting approaches have been hampered by the need for random access to large datasets and by the nonuniform sampling

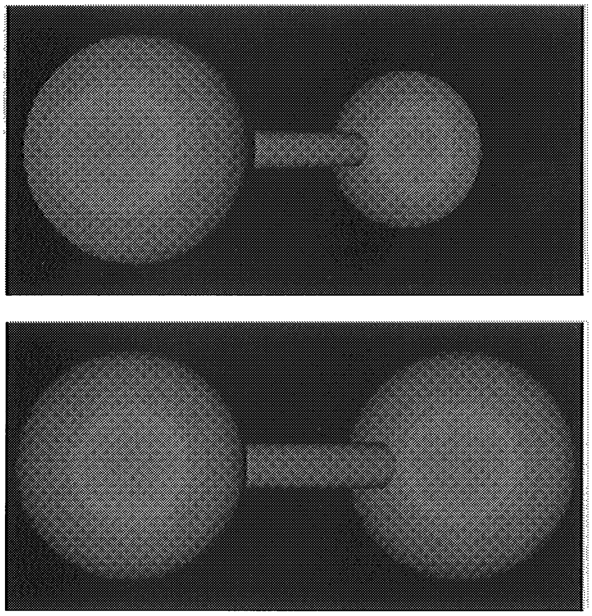


Figure 5.1: A comparison of a perspective (top) and an orthographic (bottom) view of two equal-sized spheres connected with a cylinder. In the orthographic view, lack of perspective depth cuing causes size and orientation ambiguity.

rate inherent in perspective ray divergence. In this section, we present an optimal paging strategy and an adaptive supersampling technique that address each of these problems. Used in concert, they result in an algorithm that computes perspective projections with modest requirements in terms of both computation time and resident memory.

Several researchers have explored the use of perspective in volume rendering. However, to date none have described efficient, single pass slice-by-slice processing techniques. Most ray casting implementations have assumed random access to the volume dataset [SSW86,UK88,Sab88,Web90]. If the volume data has n voxels on a side, this approach requires $O(n^3)$ storage space, which quickly outstrips RAM capacity for high resolution arrays. Random access to large virtual memory spaces

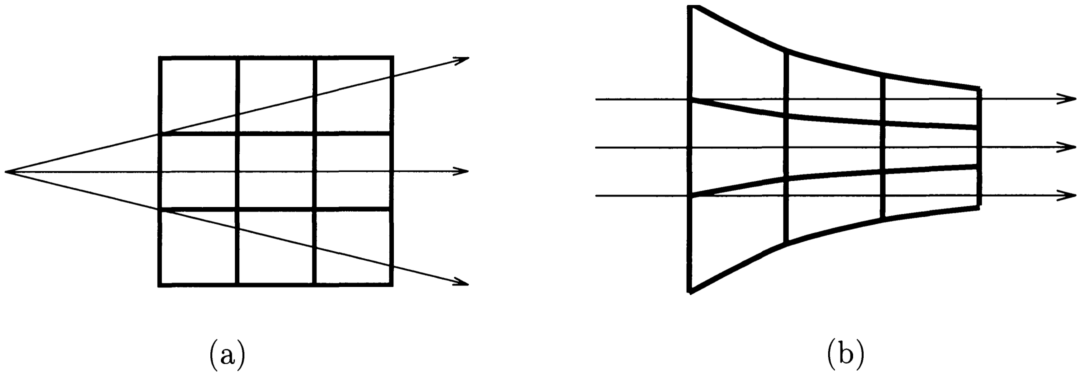


Figure 5.2: Computing perspective by predistorting the volume. The perspective rays, (a), can be computed by firing parallel rays, (b), into an appropriately distorted version of the dataset.

causes an unacceptable amount of disk paging. Another approach, Upson and Keeler's cell-by-cell processing method, assumes that the data can be accessed a slice-at-a-time along any of the three coordinate axes [UK88]. This presents a difficulty because volume data is usually stored so that slice data may be efficiently accessed on only one axis.

Perspective views may also be realized by casting parallel rays into an appropriately transformed copy of the volume grid [DCH88,HBP+89,Han90]. This approach is illustrated in Figure 5.2. Aside from being expensive, the resampling step is prone to error. In fact, even under ideal conditions, where the original volume is sampled above the Nyquist frequency, error can be introduced using this method. This is because the data is resampled onto a grid with the same resolution as original grid. The shrinking of parts of the volume that are distant from the eyepoint, an effect that can be observed in Figure 5.2, will cause an increase in the frequency of the 3D signal. This in turn raises the signal's Nyquist frequency, possibly rendering the original sampling rate inadequate [Han90].

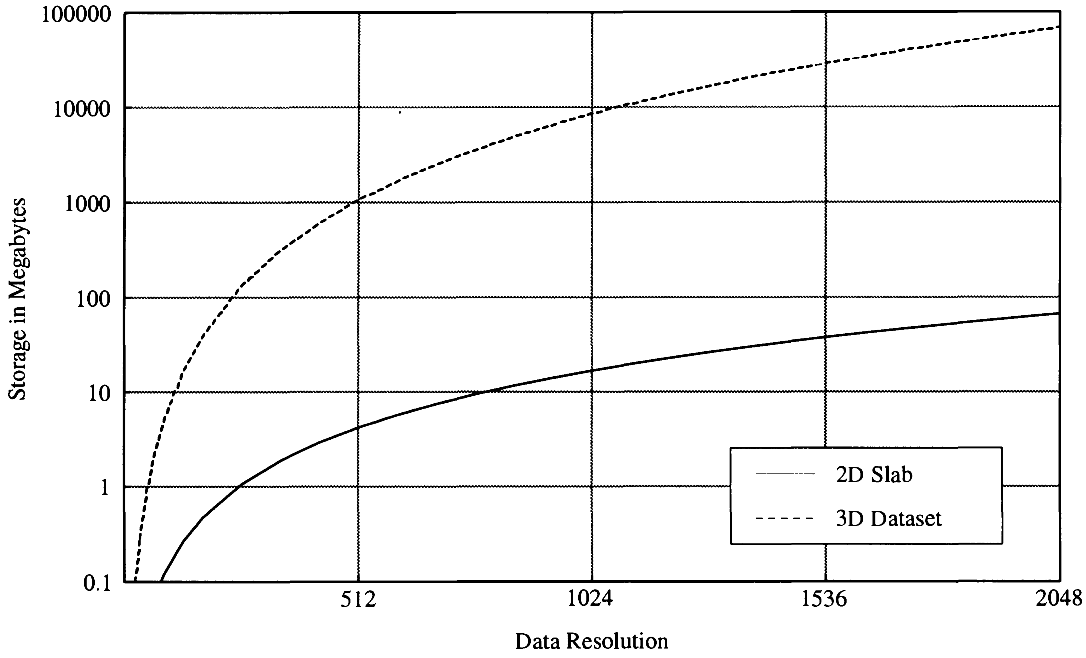
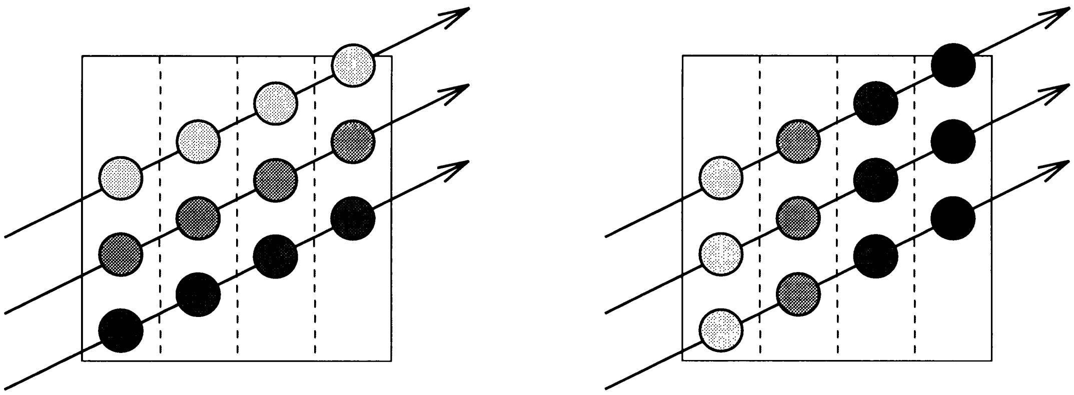


Figure 5.3: Storage required for a 3D volume compared to storage required for two 2D slices. Even at the highest data resolutions, the 2D slices will fit into RAM memory on current workstations. A 4-byte floating-point representation for each sample of α and ϵ is assumed. Note that the vertical axis is logarithmic.

5.1.1 An Optimal Paging Strategy

In this section, we present an algorithm that reduces the resident memory requirements of a ray-casting volume renderer from $O(n^3)$ to $O(n^2)$. The increase in efficiency is due to processing two 2D slices of data at a time, rather than the entire dataset. The significance of this improvement may be observed in Figure 5.3 which compares an $O(n^3)$ growth curve to an $O(n^2)$ growth curve. Assuming 32-bit floating point representations for each sample of emission and absorption, a $2048 \times 2048 \times 2048$ dataset requires 68 *giga*bytes of data. Two 2D slices of this same dataset will require only 67 *mega*bytes of data – storage that is easily found on modern workstations. We present the new algorithm in two parts. First, a basic scheme to allow ray tracing on a slice-by-slice basis is introduced. Next, we describe a processing order that requires that each slice of the dataset is loaded into memory only once.



(a) (b)

Figure 5.4: Ray-oriented versus data-oriented processing. In the diagrams above, ray segments with the same shade dot are processed during the same iteration. (a) Using ray-oriented processing, all segments on each ray are processed at the same time. (b) Using data-oriented processing, all segments in the same region are processed at the same time.

Slice by Slice Ray Tracing

A typical ray tracer has an outer loop that iterates once for each ray. Each ray is traced until it terminates, yielding a final pixel value. Processing then concentrates on the next ray. This scheme, illustrated in Figure 5.4a, is extremely inefficient with regard to memory access because each ray requires access to sample points from across the entire dataset. If the data is too large to fit into RAM, the processing of the segments that comprise a ray is likely to cause virtual memory page faults. The faults will be repeated for each ray, and each part of the dataset may be loaded and re-loaded into RAM thousands of times.

To increase efficiency the algorithm is inverted, so that it is driven by the data currently in memory. This technique, illustrated in Figure 5.4b, performs as many of the necessary operations on the data currently loaded into memory before swapping it out of RAM. We decided to base our paging strategy on two-dimensional slices of data. As we have already established, the growth rate of n^2 is slow enough that slices can be accommodated in typical memory configurations. At the same time, slices are large enough so that the overhead involved in reading a slice of data from a disk is relatively insignificant. Because we rely on trilinear interpolation at sample points, at least two slices of data are required in memory at one time. We call the volume bounded by a pair of adjacent slices a *slab*. Higher-order interpolation can be used at the cost of maintaining thicker slabs.

<u>*Initialize_SlabQueue*</u>(r, V, s)
[*Generate an initial slab queue, given a set of rays, r, a volume V,* and a set of slabs, s]
SlabQueue $\leftarrow$ empty slab queue
for each ray r_i **in** r **do**
    $s_i \leftarrow$ slab that contains the first intersection of r_i with V.
    Enqueue ray r_i onto $SlabQeueue[s_i]$
end for
return *SlabQueue*

Figure 5.5: Pseudo-code for initializing the slab queue.

In order to implement the data-oriented strategy, we developed a data structure called a *slab queue*. A slab queue has an entry for each slab of data. Each entry is a queue of rays. These are rays that require the data in the slab for the processing of their next segment. The slab queue structure is initialized by tracing each ray to its first intersection with the volume array. The slab that contains this intersection point is determined, and each ray is placed on the appropriate queue. This algorithm is shown in pseudo-code in Figure 5.5. The existence of the slab queue enables the following data-oriented processing strategy, given in pseudo-code in Figure 5.6. Whenever a slab is loaded into memory, it activates each ray on its queue in turn. The ray processes the segments in its path until the next segment lies in another slab, or until it exits the volume. In the former case, the slab containing the next segment is determined, and the ray is placed on the proper slab queue.

This algorithm requires the ability to stop and restart the tracing of a ray. We therefore introduce another data structure that contains all information pertinent to a ray's processing. A pointer to this data structure is what actually is stored in the slab queue. At a minimum, this structure must contain:

- For each wavelength of interest, the intensity accumulated along the ray up to the current segment;
- the transparency accumulated along the ray up to the current segment;
- the direction of the ray (the origin is simply the eyepoint).
- a pointer to the next ray structure on the slab queue, or an indicator that this ray is the last on the queue;

```
Process_SlabQueue(SlabQueue)
[Given an initialized slab queue, compute a volume rendering]
while SlabQueue is not empty do
    Choose a slab s_i to load into memory
    for each ray r_i in SlabQueue[s_i] do
        repeat
            Evaluate illumination for current ray segment [Section 4.1]
            s_j ← slab containing r_i's next segment
        until s_j ≠ s_i
        if s_j is within the volume then
            Enqueue r_i onto SlabQueue[s_j]
    end for
end while
```

Figure 5.6: Pseudo-code for processing the slab queue.

Additional fields are useful for storing information that speed ray operations. For a piecewise constant algorithm that takes evenly-spaced sample points along the ray, such as the one presented by Levoy [Lev88], retaining the current sample point reduces ray reactivation overhead. Storing the last sample point that is in the volume speeds testing of the termination condition.

In order to apply the quadrature rules given in Chapter 4, the exact boundaries of the ray segments must be computed. For this purpose we use the ray/grid intersection of Snyder and Barr that was described in Section 4.2 [SB87]. For this algorithm, it is helpful to store distance, t, from ray origin to the start of the current segment and the t values of the next data slice along each axis: t_x, t_y, t_z.

The exact amount of storage will depend on the bits of precision used to store numeric values and on the optional fields that are used. The current implementation for computing single-precision grayscale images uses 40 bytes for evenly-spaced sample points and 56 bytes for exact ray/grid intersection. Total storage cost for the ray data structures and one slab of emission and absorption values is $56m^2+8n^2$ bytes of storage, where m is the screen resolution and n is the data resolution. This compares to $8n^3$ bytes of storage when the entire dataset is stored in memory. If we assume screen resolution equal to data resolution, the method presented here is more memory efficient for $n > 8$.

The slab queues are maintained as linked lists through the pointer field in the ray data structure. The ray queuing and dequeueing operations require only two pointer reassignments each, since objects are added and removed from the front of the queue. Thus, the overhead for maintaining the queues is negligible.

Slab Processing Order

Now that we have established a method for data-driven evaluation of ray segments, we wish to determine a slab processing order that allows all rays to be computed while loading each slab into memory only once. We derive the processing order for front-to-back processing of ray segments. The reverse order is optimal for back-to-front processing. The order we define takes advantage of the fact that all viewing rays share a common origin – the eyepoint. Assume without loss of generality that the data may be accessed as slices perpendicular to the z axis. Let k be the number of slabs, and let the slabs be indexed by increasing z value, $s_1, s_2, \ldots, s_k$. We denote the slab that contains the z coordinate of the eyepoint s_z. If no slab contains the z coordinate of the eyepoint, s_z is the slab whose z range is closest the the z coordinate of the eyepoint. Regardless of view direction, all rays emanating from the eyepoint must fall in to one of three categories:

1. The ray may stay in the s_z slab.

2. The ray may diverge in the $+z$ direction from the eyepoint. In this case it requires access to the slabs $s_z, s_{z+1}, \ldots s_k$ in increasing order.

3. The ray may diverge in the $-z$ direction from the eyepoint. In this case it requires access to the slabs $s_z, s_{z-1}, \ldots s_1$ in decreasing order.

Rays in any of the categories may therefore be processed during a single pass through the dataset. We first load slabs s_z through s_k in increasing order and then slabs s_{z-1} through s_1 in decreasing order. Alternatively the slabs may be loaded starting with s_z and decreasing first. In either case, the order is optimal in terms of paging since each data point is loaded only once. Several examples using different eyepoints and processing orders are shown in Figure 5.7.

Combining the data-driven processing technique with the slab processing order, we obtain a complete algorithm for ray-tracing volume datasets using perspective projection that requires only $O(m^2)$ storage, makes a single pass through the dataset, and requires only a negligible amount of computation time over that required by algorithms requiring $O(n^3)$ storage. Adaptive refinement may be achieved by displaying the image after each slab is processed. The effectiveness of the intermediate results depends on the orientation of the slabs relative to the image plane.

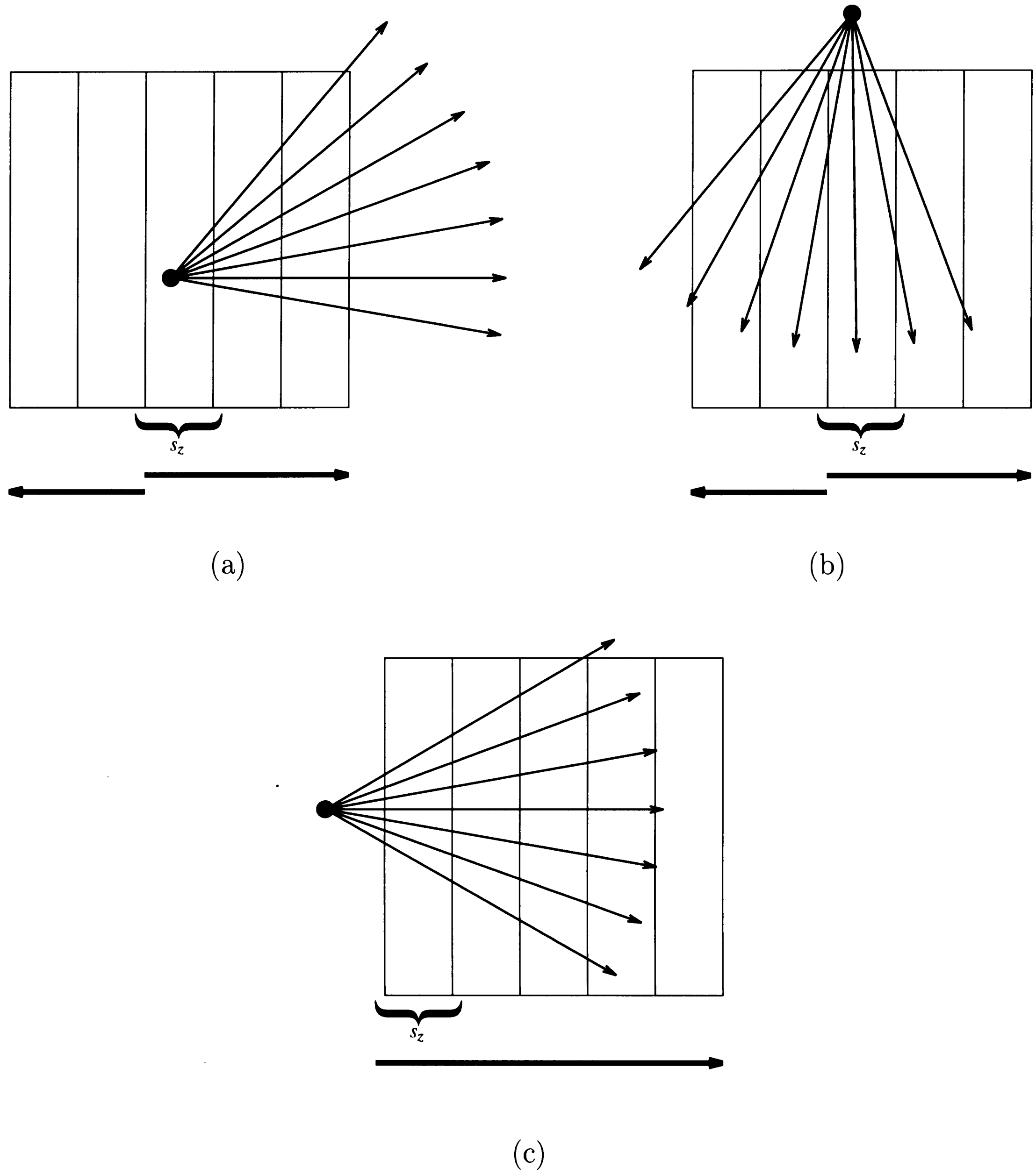


Figure 5.7: Slabs are processed starting from slab s_z, the slab containing the z-coordinate of the eyepoint, and continuing outward. This order is indicated by the arrows at the bottom of each diagram. In (a) and (b), the z-coordinate of the eyepoint is the same. This leads to identical processing orders even though the view directions are different. In (c), the eyepoint lies outside the z extent of the slabs. The slab closest to the eyepoint becomes s_z.

5.1.2 Sampling Issues

We now turn our attention to sampling problems inherent in perspective projection. There are two issues:

- Using perspective projection, objects that are more distant from the eyepoint occupy less screen space than similar-sized objects that are close to the eyepoint. Therefore, there is a loss of object-space resolution toward the back of the scene being viewed.

- Perspective rays diverge, causing the 3D sampling rate to change with depth. Undersampling of the volume data is likely to occur.

The first issue is inherent in a perspective projection. This effect is familiar in conventional rendering and photography as well as in our own visual system. We do not consider this a serious problem. If increased resolution is necessary for objects near the back of the dataset, the user may move the eyepoint closer to the object of interest, rotate the eyepoint to look at the back of the dataset, or simply increase image resolution.

Of greater concern is the potential for undersampling the volume data due to perspective ray divergence. This can result in severe aliasing, which is visually unappealing, and more seriously, the erroneous omission of small objects from the image entirely. To better understand this latter case, consider a 60-degree field of view looking directly onto a face of the volume bounding box. One ray is shot through the center of each pixel on a screen equal in resolution to a slice of the dataset. The first slice of the dataset will be sampled at the same rate as the screen. However, this sampling rate will decrease for each successive slice. By the time the last slice is reached, the sampling rate is half the screen sampling rate. This example is illustrated in 2D in Figure 5.8.

One way to control undersampling is to increase the number of rays that are fired per pixel. Such supersampling was the basis for the controlled-precision image rendering algorithm presented in Section 4.3. With appropriate error tolerances and sufficient computational resources, this technique can compute perspective images with no perceptible rendering artifact. Uniform supersampling of screen space is not, however, an efficient method for increasing the accuracy of perspective images. As shown in Figure 5.9, when we fire enough rays to adequately sample the regions of the dataset that are distant from the eyepoint, the regions close to the eyepoint are *over*sampled, thus wasting computation time.

An alternative approach to maintaining adequate sampling rates with perspective projection would be to use multiresolution filterings of the data as in the 3D-mipmap algorithms [Wil83,Lev90a]. In these schema, instead of firing additional rays to obtain the average value across a larger region, samples are taken from successively blurred versions of the volume data. This has the positive effect of reducing the number of samples taken during ray tracing. In particular, it reduces computation effort in distant areas which do not contribute much to the final image. However, it adds an extra level of interpolation between the raw data and the ray sampling process that is likely to introduce error. It also adds some complexity to the ray tracer which will have to pass through several sets of slabs.

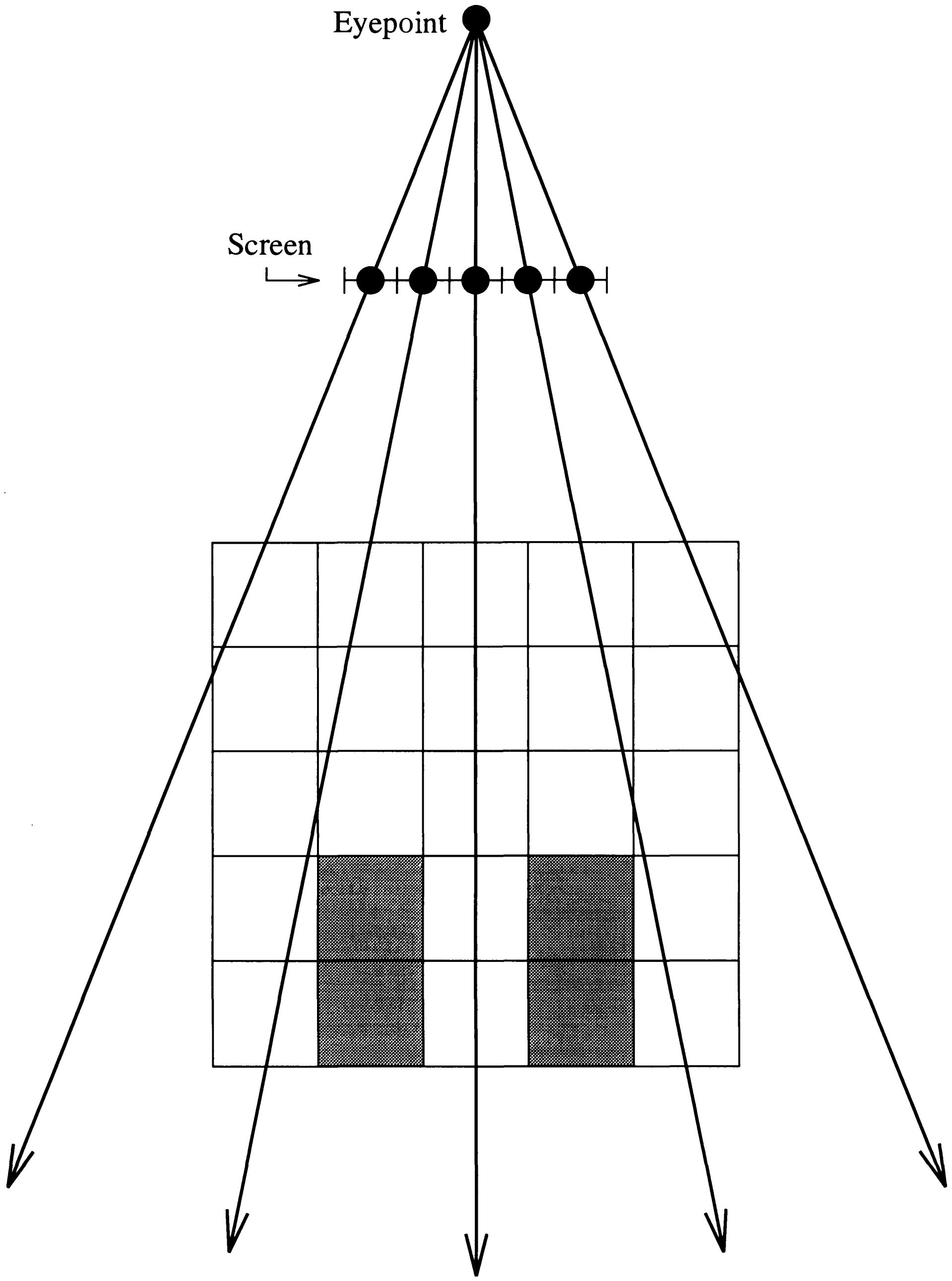


Figure 5.8: Divergence of perspective rays causes nonuniform sampling of the volume data. In this case, although the first slice of data is sampled once per cell, the last slice is sampled only every two voxels. The shaded cells are not encountered by any rays. Any data contained in these cells will not contribute to the final image.

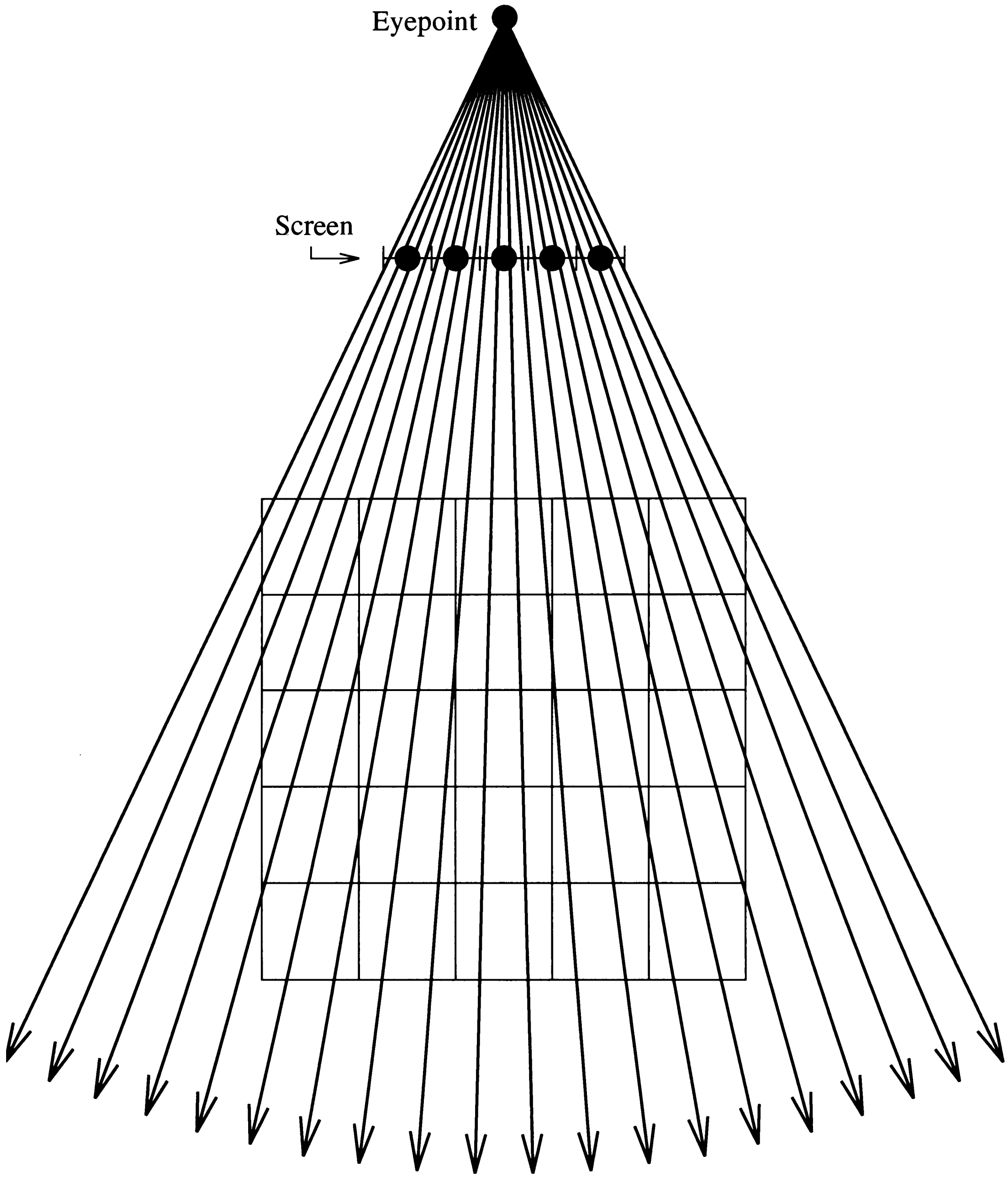
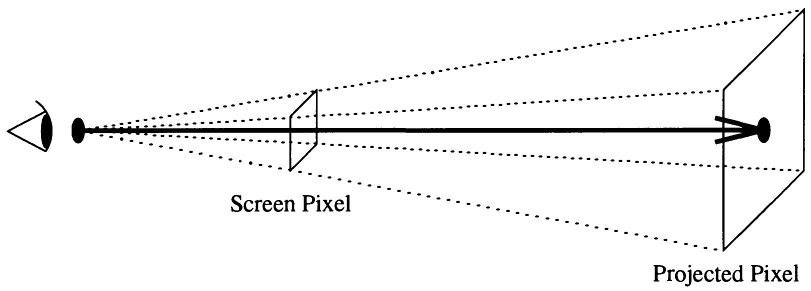


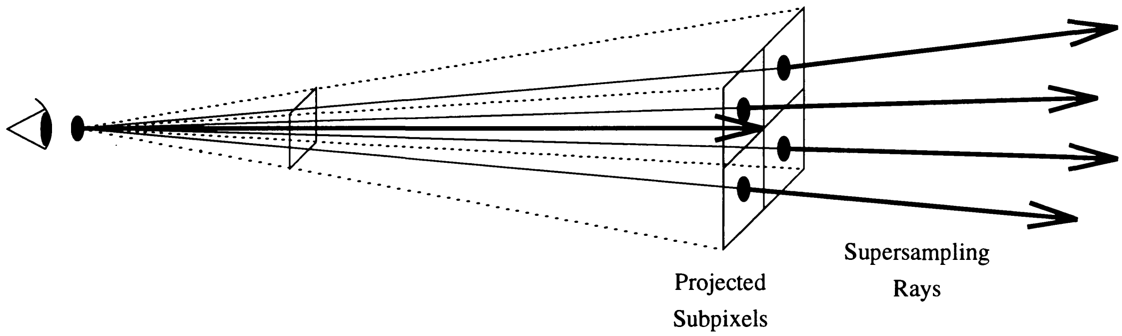
Figure 5.9: A simple solution to the problem illustrated in Figure 5.8 is to use pixel supersampling. Enough rays are fired through each pixel so that the most distant voxels are sampled adequately. This results in wasted computation as voxels close to the eyepoint are oversampled.

We have developed a supersampling approach that does not waste processing time by oversampling regions near the eyepoint. The method originates supersampling rays within the volume, rather than at the eye point. The basic scheme is illustrated in Figure 5.10. A ray is traced through the center of a screen pixel, and into the volume. Processing continues until the ray reaches a point where the projected area of the pixel reaches a user-specified threshold. The ray is then split into four supersampling rays that originate at the centers of subpixels, extrapolated to the distance of the split point. The splitting continues recursively for the supersampling rays as the projected area of their subpixels reaches the threshold. The process stops when a ray exits the volume. The algorithm results in a tree of rays spawned from each initial viewing ray. The overall effect is shown in 2D in Figure 5.11.

For each ray, values of intensity, $I(a, b)$, and transparency, $T(a, b)$ are computed using the quadrature rules from Section 4.1. After the splitting is complete, a pixel value must be determined based on the supersampling ray values, which are stored in the tree of rays. At each level of the tree, the four child rays are composited to yield a single value of intensity, I, and transparency, T. These composite values will be combined with those obtained along the parent ray using the methods from Section 3.3.2. Composition of the ray values of the children is analogous to the process of obtaining a single pixel value from ordinary supersampling. In the current implementation, we use a box filter, however any filter could be used in its place.



(a)



(b)

Figure 5.10: The ray splitting algorithm. (a) A ray is traced until the projected area of the screen pixel reaches a user-specified threshold. (b) At this point, the ray is halted. Further processing is done along four supersampling rays. This prevents the sampling rate from dropping too low.

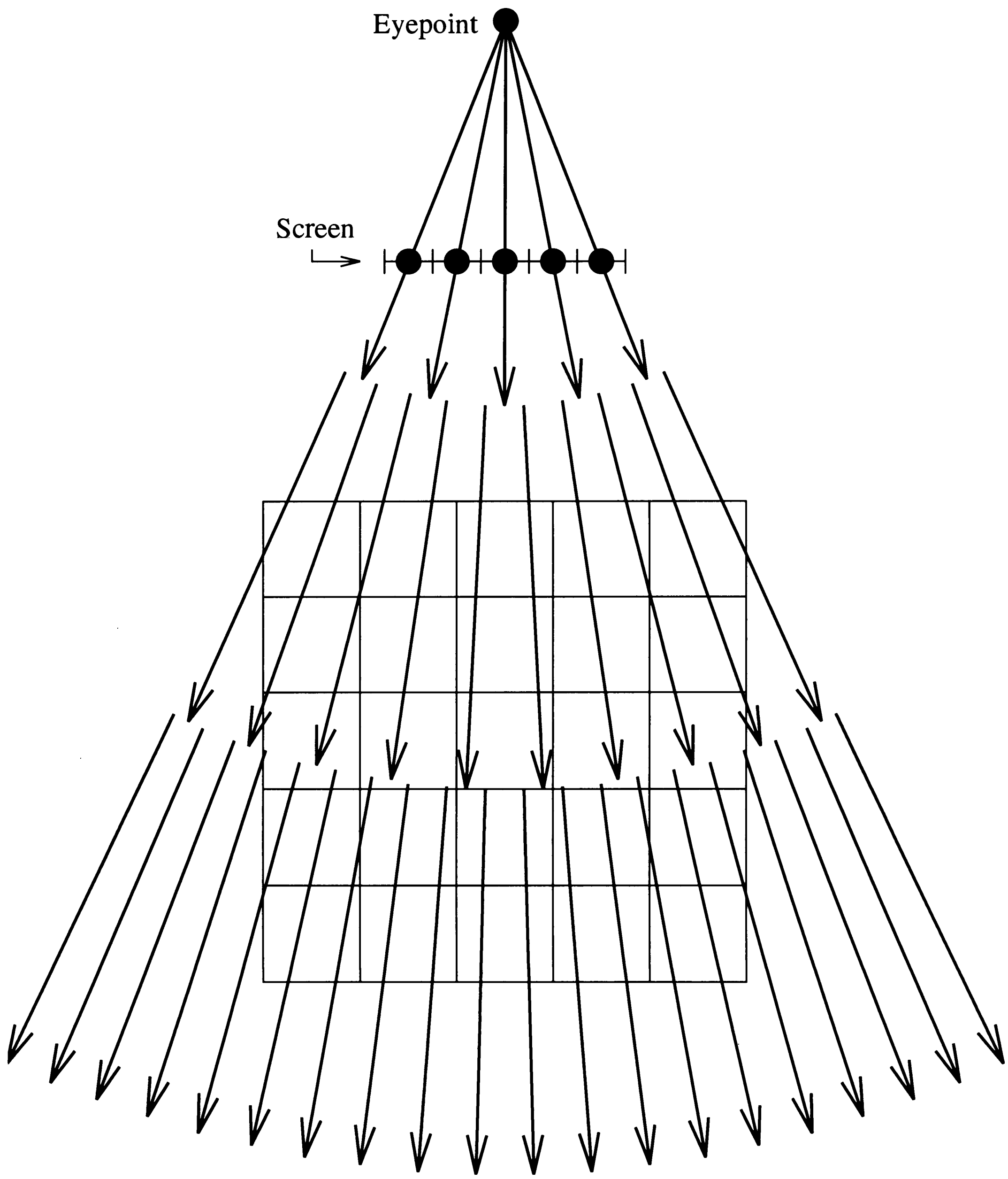


Figure 5.11: Using adaptive supersampling, each ray is split when divergence causes sampling below a threshold value. The resulting tree of rays are then composited to yield final pixel values. Using the method, the benefits of supersampling distant regions can be attained without the cost of oversampling nearby regions.

The size of the projected pixel that will induce splitting is determined by considering the smallest-sized object that the end-user wishes to be considered in the final image. By setting the threshold projected pixel area to be less than the square of the smallest voxel grid spacing, all voxels within the viewing pyramid can be guaranteed to contribute to the final image. Reducing the threshold value will result in increased accuracy at the cost of more supersampling rays. Unfortunately, it is not currently possible to determine in advance the splitting threshold that is necessary to ensure that final pixel values are computed to within a certain error tolerance.

This adaptive supersampling scheme works well in conjunction with the slice-by-slice processing algorithm presented above. The ray split points do not depend on the data values, so the start and end points of each ray in the tree may be computed as a preprocess. Each of these rays is then queued separately during initialization. The slice processing order remains unchanged.

The computation of pixel values from the ray trees is performed during a post-process. In order to implement this phase of the algorithm, it is necessary to amend the ray data structure described in Section 5.1.1 to include pointers to the ray's children. If the child rays are allocated as a contiguous block of memory, only one pointer needs to be stored.

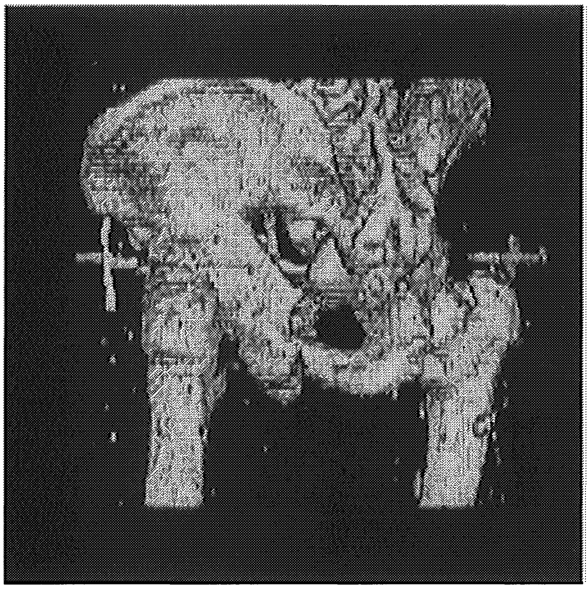
Our adaptive supersampling method is attractive because it automatically computes the number of child rays necessary to maintain a given sampling rate in a manner that is independent of screen resolution. Through ray splitting, computational effort is not wasted by oversampling the region close to the eyepoint. While supersampling is not always necessary for good image quality, this technique can be implemented quite easily as an option to any ray tracer, thus providing a guarantee that all voxels are taken into account when such precision is absolutely necessary.

5.1.3 Results

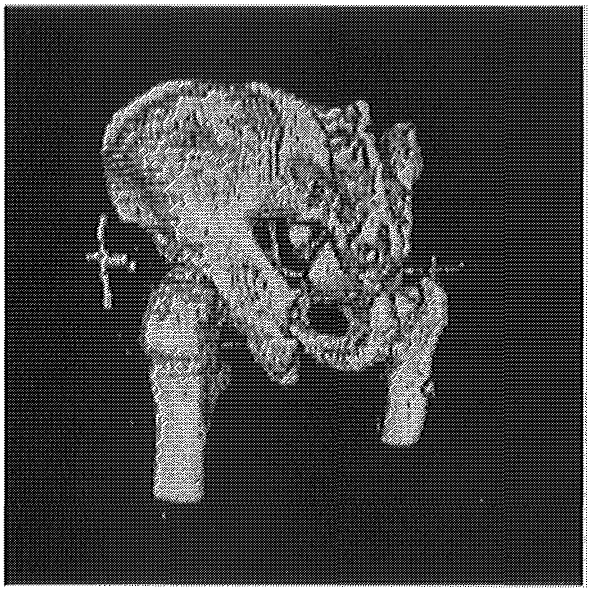
Figures 5.12 through 5.16 show 256×256 renderings of a $119 \times 112 \times 73$ CT study of a human pelvis. The viewpoint is on the left side and slightly toward the back of the body. Image 5.12 compares a perspective view with an orthographic view. In both cases, one ray was cast per screen pixel. Note that in the orthographic view, the size of the femurs (leg bones) do not vary with distance, so the orientation of the pelvis is difficult to discern. The perspective view was rendered using the slice-by-slice processing technique presented here. It took 18 seconds to render on a HP750 workstation. Profiling showed that 2.5 per cent of the total time was spent performing the ray enqueue and dequeue operations that were described in Section 5.1.1. This percentage could be substantially reduced if the calls were implemented as macros rather than as procedure calls.

Figure 5.13 compares uniform supersampling to the ray-splitting adaptive supersampling technique. Figure 5.13a was computed using uniform supersampling to ensure that at least two rays pass through each voxel. Approximately 12 million samples of emission and absorption were taken inside the volume array. Figure 5.13b was computed using adaptive supersampling to ensure the same minimum sampling rate while taking only about 6 million samples of emission and absorption inside the volume array. Figure 5.13c is a subtraction of images a and b.

Perspective projection also enabled us to experiment with alternate forms of presenting the volume data. One approach was to attain data fly-throughs by moving the eyepoint within the volume. Three frames of a fly-through animation

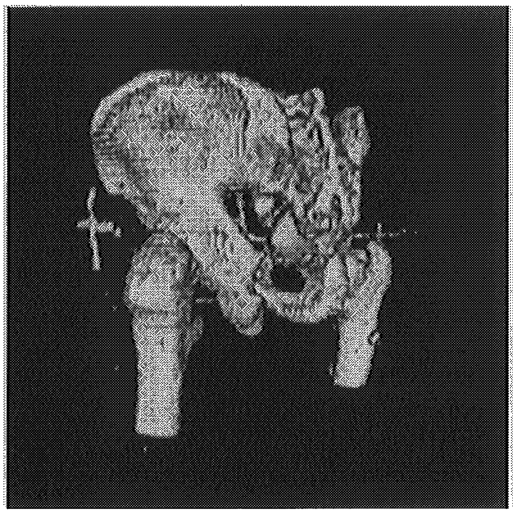


(a)

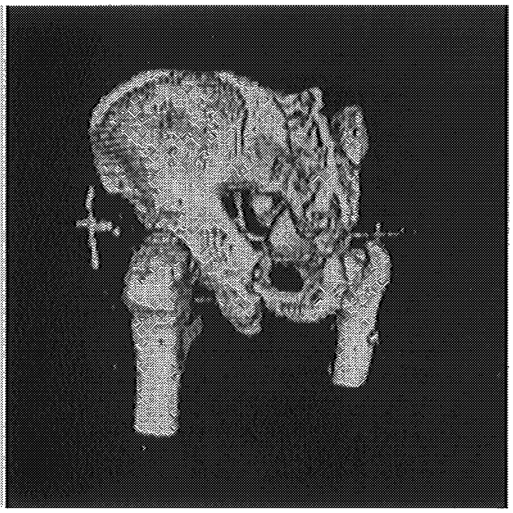


(c)

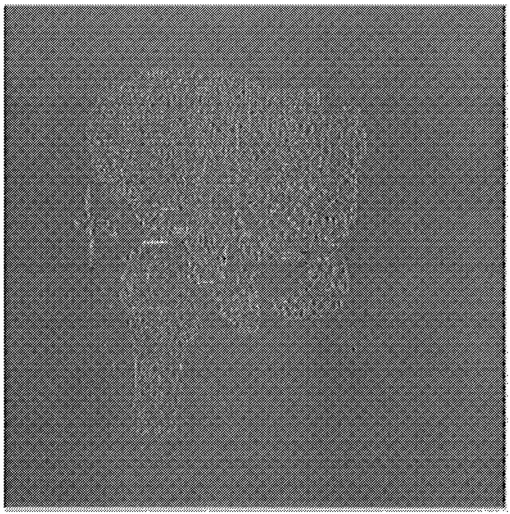
Figure 5.12: (a) A 256×256 orthographic view of a $192 \times 113 \times 73$ CT study of a human pelvis. Note that the orientation of the pelvis is difficult to discern. (b) A 256×256 perspective view of the same dataset.



(a)



(b)



(c)

Figure 5.13: Uniform versus adaptive supersampling. (a) Uniformly supersampled image that ensures at least 2 rays pass through each voxel. (b) Adaptively supersampled image that ensures at least two rays pass through each voxel. Computation time was approximately halved by using the adaptive ray-splitting technique. (c) Subtraction of the two images.

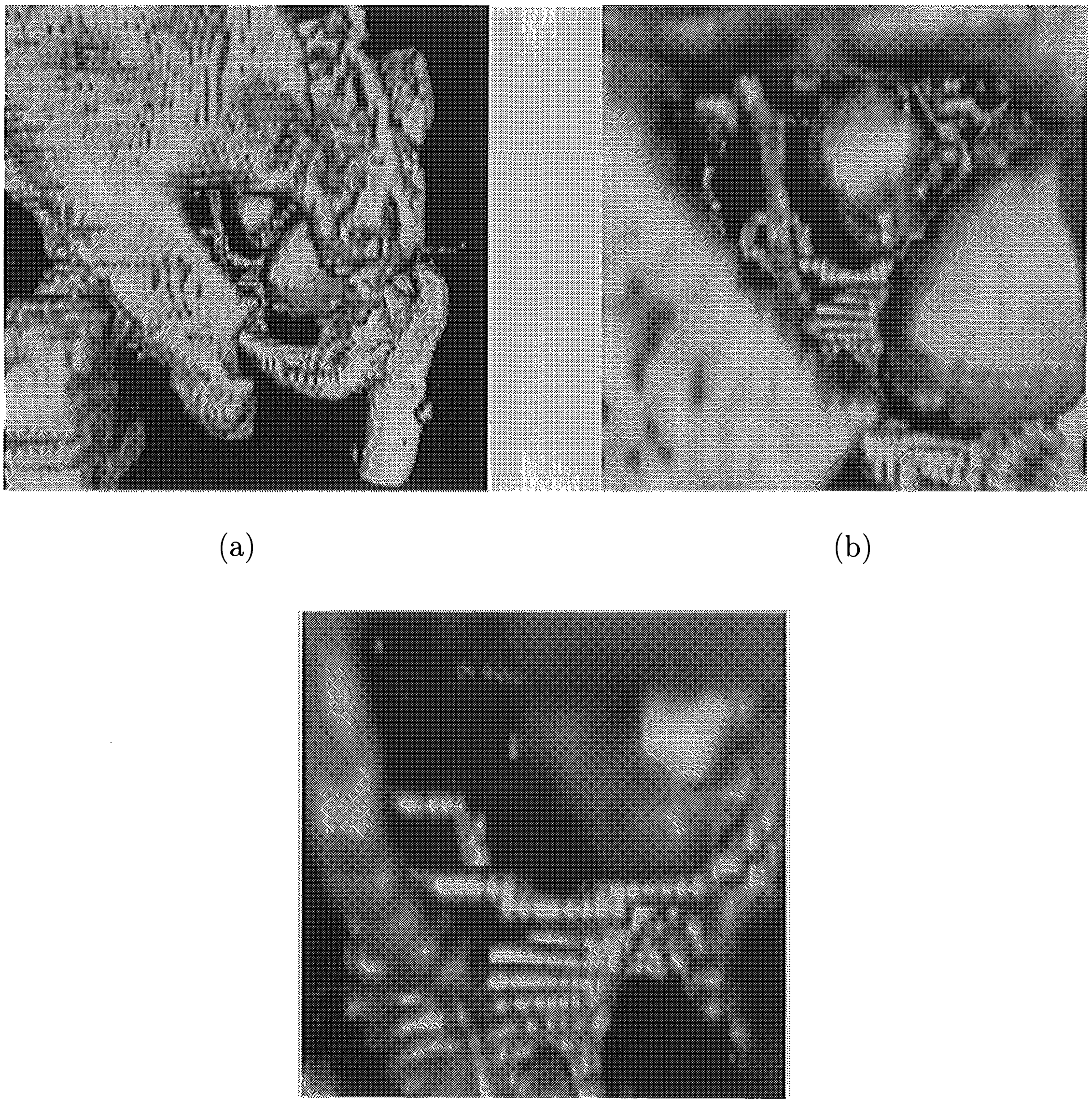
sequence are presented in Figure 5.14.

We also experimented with depth of field effects [PC82,CPC84]. If the user defines an area of interest within the volume data, we found that one may improve image comprehension by focusing a virtual camera on that region. The result is better depth perception because objects are blurred as distance from the focal region increases. The blurred remnants of distant objects help to maintain global perspective, but do not interfere with the comprehension of objects in the focal plane. One method for computing depth of field effects is is to distribute the rays over a finite aperture to blur the image outside the focal region [CPC84]. Because this technique effectively involves the jittering of the eyepoint, the slice processing order must be modified to take into account the minimum and maximum z extents of the eyepoint. This necessitates making two passes through the slices within this extent, as illustrated in Figure 5.15. Three images with different aperture sizes are presented in Figure 5.16. The images are focused on the base of the spine using a simulated normal lens at $f/2$, $f/5.6$ and $f/11$.

5.1.4 Discussion

There are several aspects of our research in efficient perspective projection that warrant further research:

Alternate processing orders. The slice processing order presented here is optimized so that each slice of memory is loaded only once. One could imagine alternative slab processing orders that would provide more useful intermediate images for adaptive refinement. This technique could be useful in time-critical



(a) (b)

(c)

Figure 5.14: Using perspective projection, the eyepoint may be moved inside the volume. The images (a), (b), and (c) show successive frames of an animated fly-through of the CT dataset.

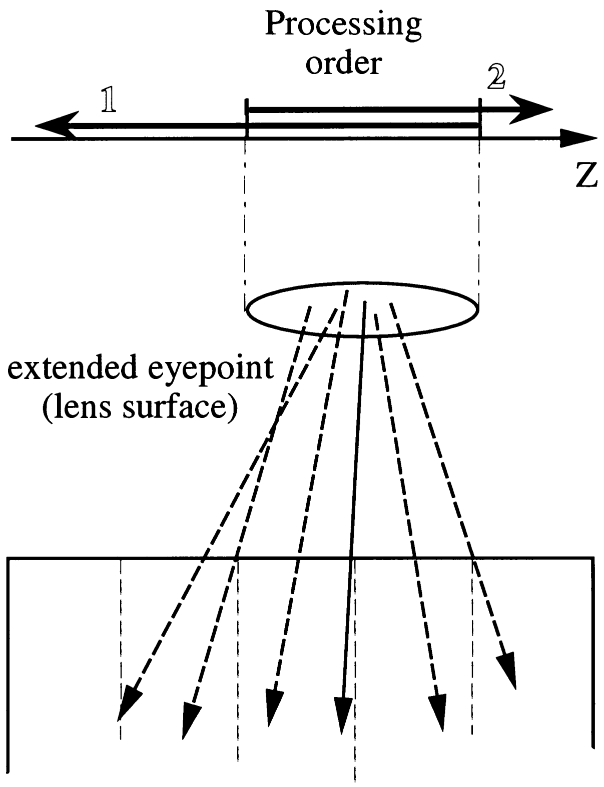
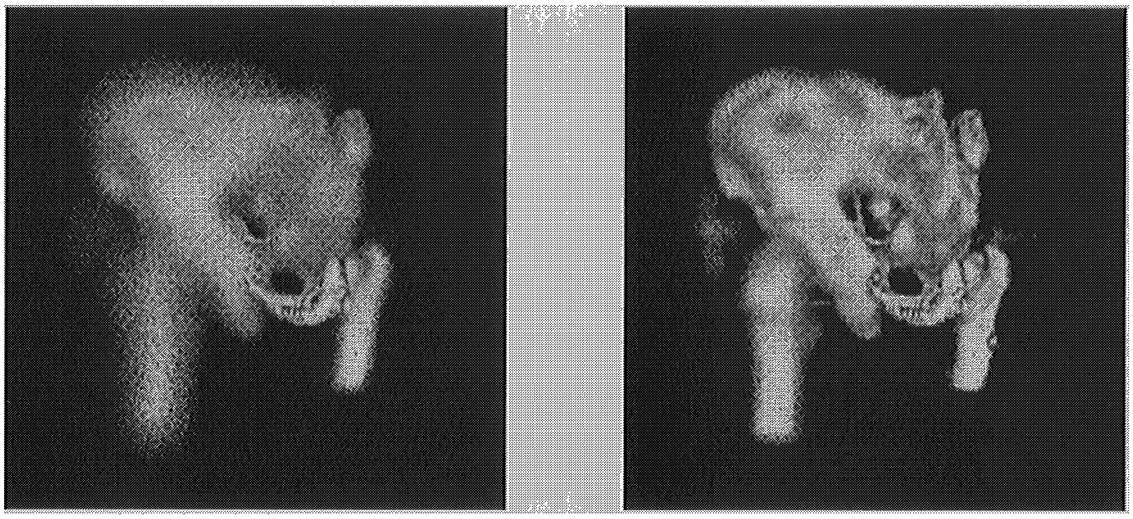


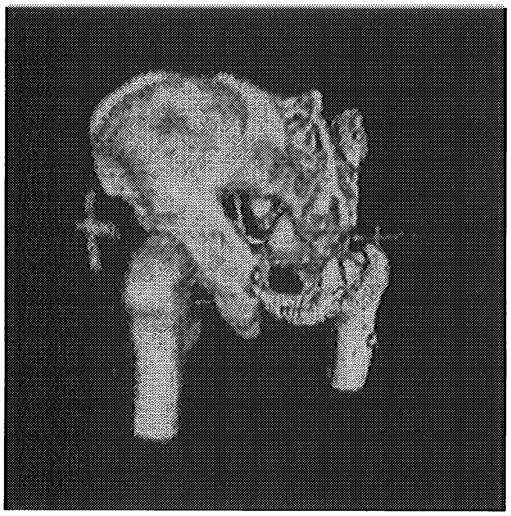
Figure 5.15: When jittering samples over a lens surface, the processing order must be modified to account for the shifting ray origins. Slabs between the minimum and maximum z extents of the lens must be processed twice.

computing applications. If each ray were given an importance weight, processing could be directed towards the slab with the most weight in its queue. A ray would be assigned a high weight if, for example, it is known to pass through a region of interest.

Parallelism. The slice-by-slice processing technique could be modified to take advantage of shared memory coarse-grained parallelism by distributing the rays on a given slab queue to several processors. If memory is not shared, all processors must load the current slab into memory. Thus parallelism should only be used if



$f/2$ $f/5.6$



$f/11$

Figure 5.16: Images computed by focusing a virtual camera on the base of the spine and using different f settings.

the slab queue is longer than a certain threshold.

Alternate data topologies. The ray tracing algorithm assumes that the data is stored as parallel rectilinear slabs. Other regular data structures should be amenable to the paging technique. Such data structures include octrees and data sampled in cylindrical and spherical coordinate grids.

Controlled-precision techniques. The ray-splitting algorithm should be extended so that the ray split points can be automatically determined, given an error tolerance for the final pixel value.

5.2 Efficient Controlled-Precision Ray Casting

We now consider the efficient computation of rays to any specified precision. Recall that the technique presented in Section 4.2 had three major shortcomings:

1. The error tolerance for the ray was distributed evenly across all segments, regardless of the ease of approximating that segment.
2. All voxels along a ray's path had to be visited.
3. The algorithm assumed that transparency could be computed exactly, and that no error was introduced due to floating-point arithmetic.

Other researchers in volume rendering have investigated methods for trading speed for accuracy in the computation of ray intensities. We described the most popular of these approaches, early ray termination, in Section 3.3.3. Early ray termination avoids computation in areas of the volume that are largely occluded [Lev90a,DH92]. Danskin and Hanrahan introduce a more sophisticated approach whereby coarse approximations to the volume rendering integral are employed in segments where such approximations will introduce little overall error [DH92]. Such segments include those that are mostly occluded; those that are empty or nearly empty; and those that are largely homogeneous. The method employs an octree organization of the data. At each node, the range of values encountered among all child nodes is stored, as well as the average value. The range information is used in finding near-empty and homogeneous regions. The average values are used in the computation of coarse approximations to the integral across a segment.

This previous work is lacking in that the approximations that are introduced do not include error bounds. Thus, although speed can be traded for accuracy, there is no indication of how accurate the solution is at any given stage. Furthermore, these techniques are limited to a strict front-to-back traversal of the dataset. The method we develop here, which we call *adaptive error bracketing* addresses both these concerns.

5.2.1 Overview

As in the method described by Danskin and Hanrahan, [DH92], we assume a hierarchical organization of the data. In our implementation, we use a BSP tree [FKN80] instead of an octree. The hierarchical structure is view-independent, and can be built once for each dataset and stored as part of the data.

Our method is based on the method of piecewise constant bounds, thus we use concepts and notation from Section 4.1.2. Associated with each segment k are two intervals $\boldsymbol{I}_k$ and $\boldsymbol{T}_k$, which respectively bracket the intensity and transparency of the segment. We actually store amended intervals that are three-tuples consisting of *minimum*, *estimated*, and *maximum* contributions. We denote the estimated value of an amended interval $\boldsymbol{A}$ by $\widetilde{A}$. When doing arithmetic, the minimum and maximum values are treated as ordinary intervals [Moo66], whereas the estimated values are treated as scalars. For example, given two amended intervals $\boldsymbol{A} = [A^-, \widetilde{A}, A^+]$ and $\boldsymbol{B} = [B^-, \widetilde{B}, B^+]$, we have $\boldsymbol{A} - \boldsymbol{B} = [\, A^- - B^+,\ \widetilde{A} - \widetilde{B},\ A^+ - B^- \,]$.

For each ray, the algorithm maintains a set $\mathcal{S}$ of contiguous segments $\{[d_{i-1}, d_i]\}$ along the ray. Initially, the set contains just a single segment whose endpoints give the intersection of the ray with the entire volume. The algorithm proceeds iteratively, at each step *selecting* the segment from $\mathcal{S}$ that is estimated to contribute the most error, then *refining* that segment—either by splitting it in two, or else by computing its contribution more precisely—and, finally, *updating* the approximation to $I(a, b)$ based on the new $\mathcal{S}$ and associated bounds. The process is repeated until the approximation to $I(a, b)$ is computed to within a given error tolerance, δ. This solution strategy is shown in pseudo code in Figure 5.17, where V is the given volume dataset. In the following sections, we look at each these steps—select, refine, and update—in more detail.

5.2.2 Selection

A good selection strategy would be to choose the segment of $\mathcal{S}$ that contributes the most error to the overall ray intensity I. Unfortunately, this information is not normally available. Note that finding the exact error contribution of each segment is equivalent to the problem of computing the ray intensity with no error.

<u>*RenderVolume*</u>(V, δ)
[*Render volume* V *to error tolerance* δ]
for each ray R **do**
  let d_0, d_1 be the entry and exit points of R through V.
  initialize $\mathcal{S} \leftarrow \{[d_0, d_1]\}$.
  initialize $\boldsymbol{I}$ using $d_1 - d_0$ and global bounds on α and ϵ.
  while $\| \boldsymbol{I} \| > \delta$ **do**
    1. [*Select.*] Choose a segment $[d_{k-1}, d_k]$ from $\mathcal{S}$ to refine.
    2. [*Refine.*] Improve the bounds $\boldsymbol{I}_k$ and $\boldsymbol{T}_k$ for $[d_{k-1}, d_k]$, either by applying a quadrature rule or by subdividing and bounding the smaller segments.
    3. [*Update.*] Compute the new value of $\boldsymbol{I}$.
  end while
  output $\widetilde{I}$ for this ray.
end for

Figure 5.17: The main loop of the adaptive error bracketing algorithm.

Our selection strategy is based on an error *estimate* that provides a heuristic for choosing the best segment to refine. It is important to note that the quality of the error estimate will only effect the selection process. The algorithm is guaranteed to maintain reliable bounds on $I(a,b)$ at all times.

In principle, the error contributed by a single segment k to the overall ray intensity can be described as the sum of two components. The first is error contributed directly due to inaccuracy in the estimated intensity contribution of the segment, $\widetilde{I}_k$. The second is error contributed indirectly due to inaccuracy in the estimated transparency of the segment, $\widetilde{T}_k$. This latter error contribution is indirect because the error is only introduced when the transparency estimate is multiplied by the intensity of the combined segments behind it. The sum of these two components may be written as

$$E_k \;=\; T_{\text{front}}(k)\,\Big(\,\Big|\,\widetilde{I}_k - I_k\,\Big| \;+\; \Big|\,\widetilde{T}_k - T_k\,\Big|\;I_{\text{back}}(k)\,\Big)\,,$$

where $T_{\text{front}}(k)$ is the accumulated transparency of the segments in front of k, and $I_{\text{back}}(k)$ is the intensity of the combined segments behind k. We do not know either of these quantities exactly, but from Equation (3.4) we can estimate them as follows:

$$\widetilde{T}_{\text{front}}(k) \;=\; \prod_{i=1}^{k-1} \widetilde{T}_i\,, \qquad \widetilde{I}_{\text{back}}(k) \;=\; \sum_{i=k+1}^{n} \widetilde{I}_i \prod_{j=k+1}^{i-1} \widetilde{T}_j\,.$$

Finally, we can bound the quantities $|\,\widetilde{I}_k - I_k\,|$ and $|\,\widetilde{T}_k - T_k\,|$ to arrive at a practical estimate $\widetilde{E}_k$ for the error contribution of segment k:

$$\widetilde{E}_k \;=\; \widetilde{T}_{\text{front}}(k)\,\Big(\,\|\,\boldsymbol{I}_k\,\| \;+\; \|\,\boldsymbol{T}_k\,\|\;\widetilde{I}_{\text{back}}(k)\Big)\,. \tag{5.1}$$

Computing and updating this error estimate efficiently for each segment k is complicated by the fact that refining a single segment changes the error estimate for all the others. We therefore employ a binary tree structure, which we call a *span tree*, to aid in propagating these updates efficiently. Each leaf of the tree represents a single segment k of the ray. The intervals $\boldsymbol{I}_k$ and $\boldsymbol{T}_k$ are at the leaves. Each internal node represents the union of all the segments below it, and stores an intensity and transparency interval for this union. Note that the actual intensity $I(a,b)$ of the entire ray is contained in the $\boldsymbol{I}$ interval of the span tree's root.

Using this structure, error estimates for any segment k can be computed while traversing the tree from the root to k. Initially, the quantities $T_{\text{front}}(k)$ and $I_{\text{back}}(k)$ are set to complete transparency and zero intensity, respectively. At each node, the error formula from Equation 5.1 is evaluated for the front and back child segments. The algorithm will descend either toward the front or toward the back of the ray, depending on which segment contains the higher error estimate. If descending towards the front, $I_{\text{back}}(k)$ is updated to include the intensity of the back child. If descending towards the back, $T_{\text{front}}(k)$ is updated to include the transparency of the front child. This algorithm is illustrated in Figure 5.18 and shown in pseudo code in Figure 5.19. The procedure selects a "promising" leaf segment in $O(\log n)$ time. Note that the search described here does not guarantee that the leaf segment with the highest estimated error will be chosen. However, a more careful selection would require visiting all the leaf nodes. In practice, we have found that the more careful $O(n)$ search is not generally worth the extra selection cost.

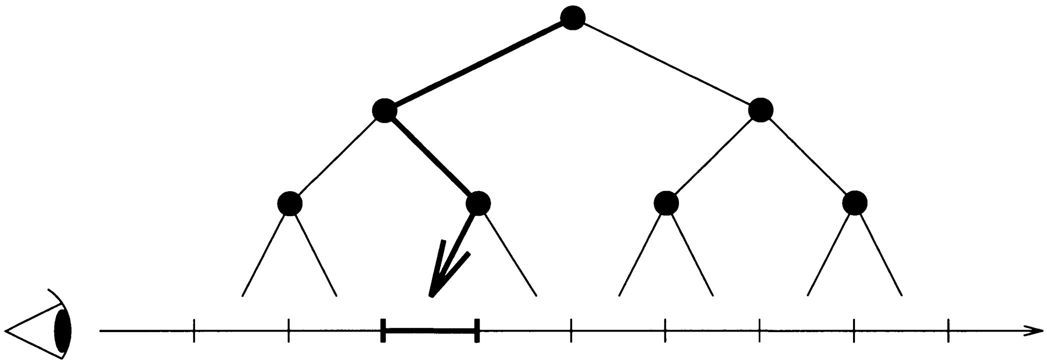


Figure 5.18: In order to avoid a sequential search for the segment with the highest estimated error, a binary tree structure called a *span tree* is employed. Amended intervals for intensity and transparency are stored at each node. Selection starts by examining the root node. At each stage, processing is directed towards the child node with the higher error, until a segment is located. This process is not guaranteed to find the segment with the highest estimated error, however it runs in only $O(\log n)$ time.

5.2.3 Refinement

The purpose of refinement is to obtain tighter bounds on the intensity and transparency within the selected segment, thereby improving the approximation of the overall intensity.

Refinement may be performed in many different ways. We chose to employ a two-tier strategy. To refine ray segments that span multiple voxels, we split the segment in two. After splitting, we obtain new bounds on the functions of absorption and emission across each of the new segments. Since the new segments are smaller, we expect the bounds to be tighter, thus resulting in a better approximation. This strategy, which corresponds to the method of piecewise constant bounds, could be applied below the voxel level at the cost of slow convergence. We chose instead to introduce a second strategy that we apply when a segment spans

<u>*Select*</u>$(root)$

[*Select a segment to refine, given the span tree root*]

return *RecursiveSelect*$(root, 1, 0)$

<u>*RecursiveSelect*</u>$(s, \widetilde{T}_{\text{front}}, \widetilde{I}_{\text{back}})$

if s is a leaf **then**

    return s

else

    [*Descend tree via front child or back child*]

    $f \leftarrow$ front child of s

    $b \leftarrow$ back child of s

    $\widetilde{I}_{\text{mid}} \leftarrow \widetilde{I}_b + \widetilde{T}_b \widetilde{I}_{\text{back}}$

    $\widetilde{T}_{\text{mid}} \leftarrow \widetilde{T}_{\text{front}} \widetilde{T}_f$

    [*Evaluate Equation (5.1) for each child*]

    $\widetilde{E}_f \leftarrow \widetilde{T}_{\text{front}}(\ \|\boldsymbol{I}_f\| + \|\boldsymbol{T}_f\|\ \widetilde{I}_{\text{mid}})$

    $\widetilde{E}_b \leftarrow \widetilde{T}_{\text{mid}}(\ \|\boldsymbol{I}_b\| + \|\boldsymbol{T}_b\|\ \widetilde{I}_{\text{back}})$

    if $\widetilde{E}_f > \widetilde{E}_b$ **then**

        return *RecursiveSelect*$(f, \widetilde{T}_{\text{front}}, \widetilde{I}_{\text{mid}})$

    else

        return *RecursiveSelect*$(b, \widetilde{T}_{\text{mid}}, \widetilde{I}_{\text{back}})$

    end if

end if

Figure 5.19: The selection process for the adaptive error bracketing algorithm.

a single voxel. This strategy is to compute the integral along the segment directly using one of the other controlled-precision quadrature rules from Chapter 4.

If refinement is done by subdivision, a splitting point must be chosen. In our implementation this point is chosen according to the splitting planes of the BSP tree. If refinement is done by application of a quadrature rule, a specific rule and error tolerance must be chosen. We used the power series to compute the integral to the highest-possible precision. This way, we know that we will not have to revisit the node for further refinement.

5.2.4 Updating

Once we have refined a segment, we need to update the values of $\boldsymbol{I}$ and $\boldsymbol{T}$ in the span tree. The update carries a dual purpose. First, it computes a new value of $I(a, b)$ that can be used to check if the current solution is within tolerance. Second, it updates values that will be used in computing error estimates for the next invocation of the selection code. The update is accomplished by a single bottom-up traversal of the tree beginning with the refined span. Each time we move to a parent node, we recompute $\boldsymbol{I}$ and $\boldsymbol{T}$ at that node by compositing the intensity and transparency information from the children. The complexity of this process is $O(\log n)$, where n is the number of spans. The algorithm for update is illustrated in Figure 5.20 and given in pseudo code in Figure 5.21.

5.2.5 Initializing the BSP tree

Our implementation uses an axis-aligned BSP tree as a simple hierarchical organization of the voxel data. This data structure can be built once and stored

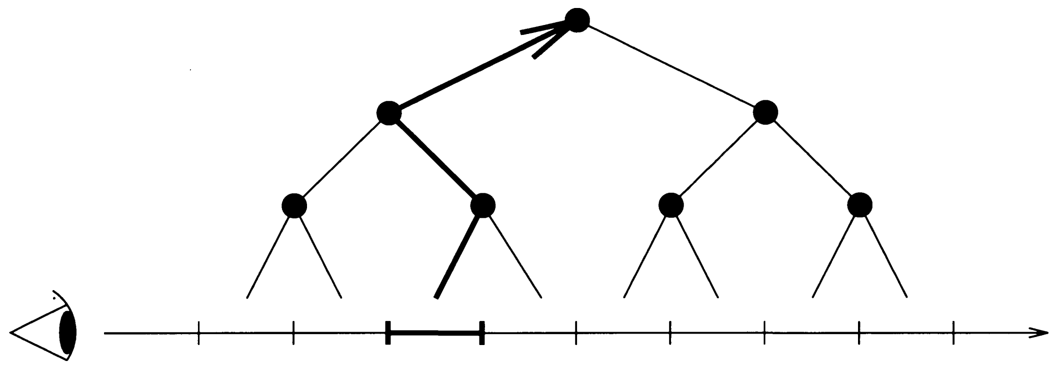


Figure 5.20: Because intermediate results are stored at the span tree nodes, updates may be processed efficiently. During a single leaf-to-root traversal, revised estimates and error bounds are propagated using pairwise compositing.

along with the data. Each node of the BSP tree stores amended intervals $\boldsymbol{\alpha}$ and $\boldsymbol{\epsilon}$ that bracket the absorption and emission coefficients for all the voxels represented by that node. The estimated values, $\tilde{\alpha}$ and $\tilde{\epsilon}$, are computed by averaging the respective coefficients. The bounds and estimates may be propagated during a bottom-up traversal of the hierarchy [LH91]. If the original dataset has N voxels, the resulting BSP tree will have $2N-1$ nodes. Each node requires a fixed amount of storage, so the BSP tree requires only a constant factor more storage than the original dataset. Unfortunately, this constant factor can become quite large. Our current implementation stores 36 bytes per node, thus a dataset with $128 \times 128 \times 128$ voxels requires 144 megabytes of storage. Although we have not yet implemented this strategy, the BSP tree planes could also be used to subdivide screen space. This information could be used to determine the parts of the data that are required for the processing of an individual ray. This would reduce resident memory requirements.

<u>*Update*</u>$(s, \boldsymbol{T}_{\text{new}}, \boldsymbol{I}_{\text{new}})$

[*Update the bounds associated with leaf node s.*]

$\boldsymbol{T}_s \leftarrow \boldsymbol{T}_{\text{new}}$

$\boldsymbol{I}_s \leftarrow \boldsymbol{I}_{\text{new}}$

[*Propagate the change up to the root.*]

while s has a parent **do**

    $p \leftarrow$ parent of s

    $f \leftarrow$ front child of p

    $b \leftarrow$ back child of p

    [*Compose* $\boldsymbol{T}$ *and* $\boldsymbol{I}$ *of p's children*]

    $\boldsymbol{T}_p \leftarrow \boldsymbol{T}_f \boldsymbol{T}_b$

    $\boldsymbol{I}_p \leftarrow \boldsymbol{I}_f + \boldsymbol{T}_f \boldsymbol{I}_b$

    $s \leftarrow p$

end while

Figure 5.21: The update process for the adaptive error bracketing algorithm.

Since we split the ray along BSP tree planes, a segment always corresponds directly to a ray intersection with a single BSP tree node. Therefore, the intervals $\boldsymbol{T}_k$ and $\boldsymbol{I}_k$ for a new segment k can be computed from the $\boldsymbol{\alpha}$ and $\boldsymbol{\epsilon}$ values stored in the BSP tree, according to equations (4.10) and (4.11). Note that splitting along BSP tree planes does not restrict ray direction. However, it does mean that segments are not necessarily split directly in half during refinement.

We apply a quadrature rule when the ray segment to be refined passes though a cell in the lowest level of the BSP tree. In this case, the BSP tree is queried for the values of α and ϵ at the corners of the cell. These values are used in determining the coefficients of the emission and absorption polynomials, $\alpha(t)$ and $\epsilon(t)$, along the ray segment.

5.2.6 Results

The adaptive error bracketing scheme attempts to compute each ray to a specified accuracy using the smallest number of refinement operations. The cost of the strategy comes in the selection and refinement operations that enable the determination of the most promising segment to refine. Selection and update each require a traversal of the span tree, imposing an $O(\log n)$ cost at each stage, where n is the number of segments under consideration. This computational cost is only worthwhile if it pays for itself in terms of a sufficient reduction in the overall number of refinement operations.

To evaluate this tradeoff, we implemented another controlled-precision ray evaluation algorithm that implements both early termination and machinery to skip

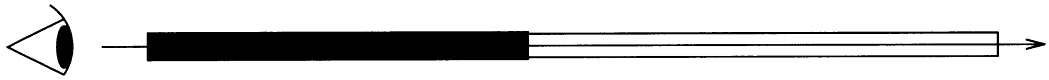


Figure 5.22: Using the front-to-back algorithm, the frontmost part of the ray, shown in black, is evaluated to the highest possible precision. Empty regions are identified by the BSP tree and skipped, since they don't contribute to the final result. While processing front-to-back, a coarse bound on the contribution of the back part of the ray is maintained. Processing halts when the maximum contribution of the back part of the ray falls below the user-specified error tolerance.

over large empty regions. In this manner, it is similar to the scheme presented by Levoy [Lev90a], however our scheme allows refinement to a precise error bound. The ray is refined front-to-back; thus, selection and update take constant time. The frontmost region is subdivided until it is empty, or until it is the size of a single voxel, in which case a quadrature rule is applied. Since we do not have the luxury of revisiting a segment using a front-to-back scheme, the error tolerance is set to machine precision. In order to maintain an error bound on $I(a, b)$ during each stage of the algorithm, we compute the maximum possible emission along the back part of the ray that has not been evaluated. This is illustrated in Figure 5.22. Termination occurs when the product of the accumulated transparency and this maximum remaining emission falls below the error tolerance.

We tested the adaptive error bracketing scheme (AEB) described in this section against the front-to-back method (FTB) outlined in the preceding paragraph using three different datasets and a range of error tolerances. We measured performance in terms of both total computation time and total number of quadrature rules employed. We ran all our tests on an HP 750, a 70 MIPS machine.

The results are shown in Figures 5.24, 5.25, and 5.26. For each dataset, we present a four-by-six array of images. The top three rows show the components of the amended intensity interval for the image. From top to bottom, they are the minimum, estimated, and maximum intensity respectively. The bottom row shows the difference of maximum and minimum intensities for each pixel. The six columns show the images produced for increasingly tight error bounds. These bounds are listed below each column, both in terms of absolute error in ray intensity (δ) and relative error. Relative error is the ratio $\delta/I_{\max}$, where $I_{\max}$ is the intensity of the brightest pixel in the highest-quality image. We also show graphs of total CPU time, and total number of quadratures below the images for each of the two methods AEB and FTB.

The first example, shown in Figure 5.24, is a $64 \times 64 \times 64$ dataset depicting a uniformly emissive grid behind a cloud of absorptive material that decreases linearly in density from left to right. The geometry of the dataset is shown in Figure 5.23. The slow variation of cloud absorption allows for accurate bracketing by the AEB algorithm without many applications of the quadrature rule. Since the cloud is not completely opaque, the FTB algorithm cannot employ early termination, thus making it 125% to 2090% slower, as the relative error ranges from 0.2% to 170%.

The second example, shown in Figure 5.25, is representative of isosurface data. It is a $192 \times 112 \times 73$ CT dataset of a human pelvis. In this example, transparency drops sharply upon intersection with the bone surface, reaching negligible values after penetrating only a few voxels. In this domain, the FTB algorithm quickly locates the first intersection with the bone surface and applies the quadrature rule

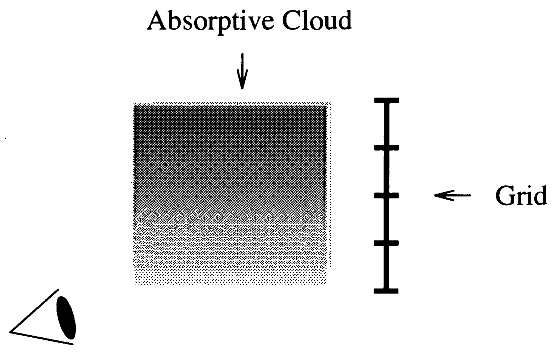


Figure 5.23: The geometry of the dataset in Figure 5.24 (plan view). The grid, indicated at right, is partially obscured by an absorptive cloud whose density increases linearly as indicated with shading. The location of the view position is shown at bottom left.

where it is most effective. The AEB algorithm, on the other hand, tries to find regions where the bracketed error is acceptable, but ultimately applies quadrature rules to many of the same voxels on the bone surface. On this high-gradient dataset, the extra cost of deciding where to refine causes AEB to run between 85% and 98% slower, as the relative error ranges from 0.5% to 30%.

The third example, shown in Figure 5.26, is a $97 \times 97 \times 116$ electron density map of an SOD enzyme. Electron density is mapped directly to absorption α, and emission ϵ is obtained via gradient shading. Absorption is low enough that rays penetrate deep into the volume, but high enough that few rays have any transparency left when they exit the dataset. For low accuracy results (within 134%), the FTB algorithm is slower than AEB. However, the benefit of using AEB gradually decreases as accuracy increases, and for the highest-precision images (within 0.5%), AEB is 104% slower even though it uses only 77% of the quadratures.

The examples suggest that adaptive bracketing is very effective in reducing the number of quadratures necessary to compute an accurate image. However, fewer quadratures does not always translate into an overall speed-up, since the extra complexity in choosing where to refine imposes additional cost. In the limit, when the highest accuracy rendering is required, there is no advantage in trying to be careful about where to refine, since any method would inevitably have to perform all of the same quadratures along the entire ray. Thus, the adaptive bracketing method pays off most when a lower-accuracy rendering is desired.

As demonstrated by the first example, the adaptive bracketing scheme has a great advantage over the front-to-back approach for "cloudy" or low-gradient data.

5.2.7 Discussion

There are several aspects of this research that warrant further research:

Other refinement strategies. The select–refine–update scheme, which is central to adaptive error bracketing, can make use of any tools that refine error estimates within a segment. Currently, we have implemented only a simple two-tier scheme: segments are split down to the voxel level, after which highly accurate quadrature is applied. Possibilities for improvement include limiting the depth of the span tree, thus applying quadrature over larger regions. The addition of other refinement tools with intermediate costs and benefits might also be useful. For example, a cheaper, lower-accuracy quadrature formula like the trapezoid rule might be one such tool. There is a tradeoff since overall cost would be higher if the same segment must be refined multiple times. We would also like to explore incorporating some notion of the relative costs and benefits of the various refinement tools into the selection strategy.

Appropriate error tolerances. The error bounds provided by adaptive bracketing are absolute. This type of error bound is useful if results are desired in precise units that are well-defined in advance. Unfortunately, this is rarely the case in volume rendering. Instead, once an image is created it is typically scaled to make effective use of the dynamic range of the display. This scaling also affects the absolute error. We would therefore like to investigate an iterative strategy whereby the absolute error tolerance is refined along with the image in order to achieve a prescribed "relative" error bound.

Bounding other errors. There are many sources of error that are currently not addressed but that may in fact be well-suited to the interval arithmetic approach described here. For example, if bounds could be placed on the data acquisition and reconstruction errors, this information could be maintained and propagated quite naturally in our scheme. The roundoff error introduced due to floating-point or fixed-precision arithmetic could also be accounted for in this way.

Irregular data. So long as we can bracket the contribution of any given sample to a segment along a ray, we can apply our technique to achieve any given accuracy. Furthermore, the BSP tree is a general-purpose data structure that does not require rectilinear cells. Thus, irregularly-spaced data could be handled directly.

Exploiting Coherence. The current algorithm starts with a new span tree for each ray. It should be possible to get a rough idea of where refinement will have to occur by using information gathered while processing nearby rays. One approach would be to use the structure of a neighboring span tree as an initial guess. This type of approach was employed successfully for maintaining active edge lists for polygon scan-conversion by Watkins [Wat70].

Progressive refinement. Ideally, the effort invested in making a fast but low-quality image could be put to good use when making a highly-accurate rendition of the same image. The idea of progressive refinement is to generate approximate images as intermediate results on the way to the final image [BFG86]. The progressive technique of Levoy [Lev90b] refines solutions by continually adding rays and interpolating the regions between them. An approach based on adaptive error bracketing would instead refine all rays simultaneously. However, such a scheme would require storing a span tree for every ray, which would increase the overall storage requirement to size $O(m^3 + mp^2)$, where m is the linear resolution of the volume data, and p is the linear resolution of the display. This increased storage requirement may be prohibitive for very high-resolution data or displays.

Interactive exploration. The span trees store intermediate results at each node, thus enabling fast updates if the emission and absorption functions were modified. If the span tree were stored for every ray, an interactive exploration tool could be developed that would allow real-time manipulation of these data parameters.

Adding color. Color is extremely valuable in visualizing precisely those datasets for which adaptive bracketing is most effective. Color can be added in a straightforward way to our algorithm without increasing storage requirements since the BSP nodes need only bound either the greatest contribution in any one channel, or else a single luminance contribution.

Loose bounds. As we discussed in Chapter 4, one problem with the current controlled-precision techniques is that the actual error is often substantially smaller than the computed bounds indicate. For example, in Figure 5.25, all the estimated images appear identical, even though the pixel values of the leftmost image are not guaranteed to be within 170% of the correct values. Since adaptive error bracketing is based on the principle of doing the least work in refining a solution as possible, its effectiveness is diminished by a failure to recognize the true quality of solution at any given stage.

5.3 Summary and Future Directions

The algorithms presented in this chapter aim towards efficient solutions to the problems of intuitive display and accurate imaging for volume rendering. We have presented an algorithm for perspective projection that does not require random access to the dataset and that avoids both undersampling and oversampling the data. We also presented an adaptive error bracketing algorithm that attempts to compute guaranteed-accuracy solutions with the fewest refinement operations possible. It is more efficient than a conventional front-to-back algorithm for low-precision solutions and for low-gradient data.

These techniques have been successful in that their computation and storage requirements are easily met by today's workstation-level machines. However, neither of the approaches is currently valid for rendering at interactive rates. The ability to interactively explore the dataset is important for improving comprehension. Even when computing still images, it is often helpful to generate fast preview images to test rendering parameters. Currently, users seeking interactive feedback must abandon precise error bounds and accurate perspective projections. An important future research topic is to lift these restrictions.

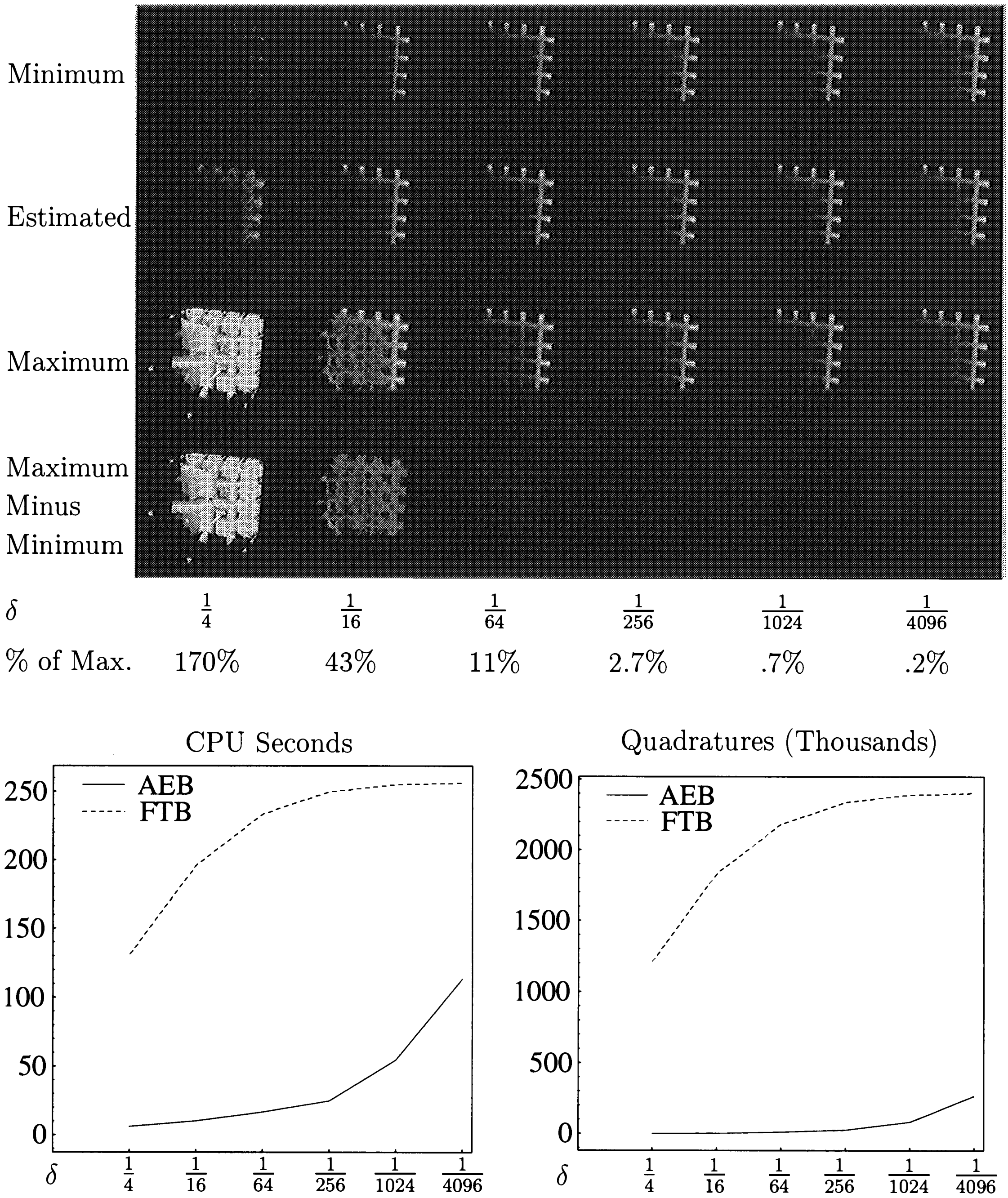


Figure 5.24: Adaptive error bracketing. Results using the grid dataset

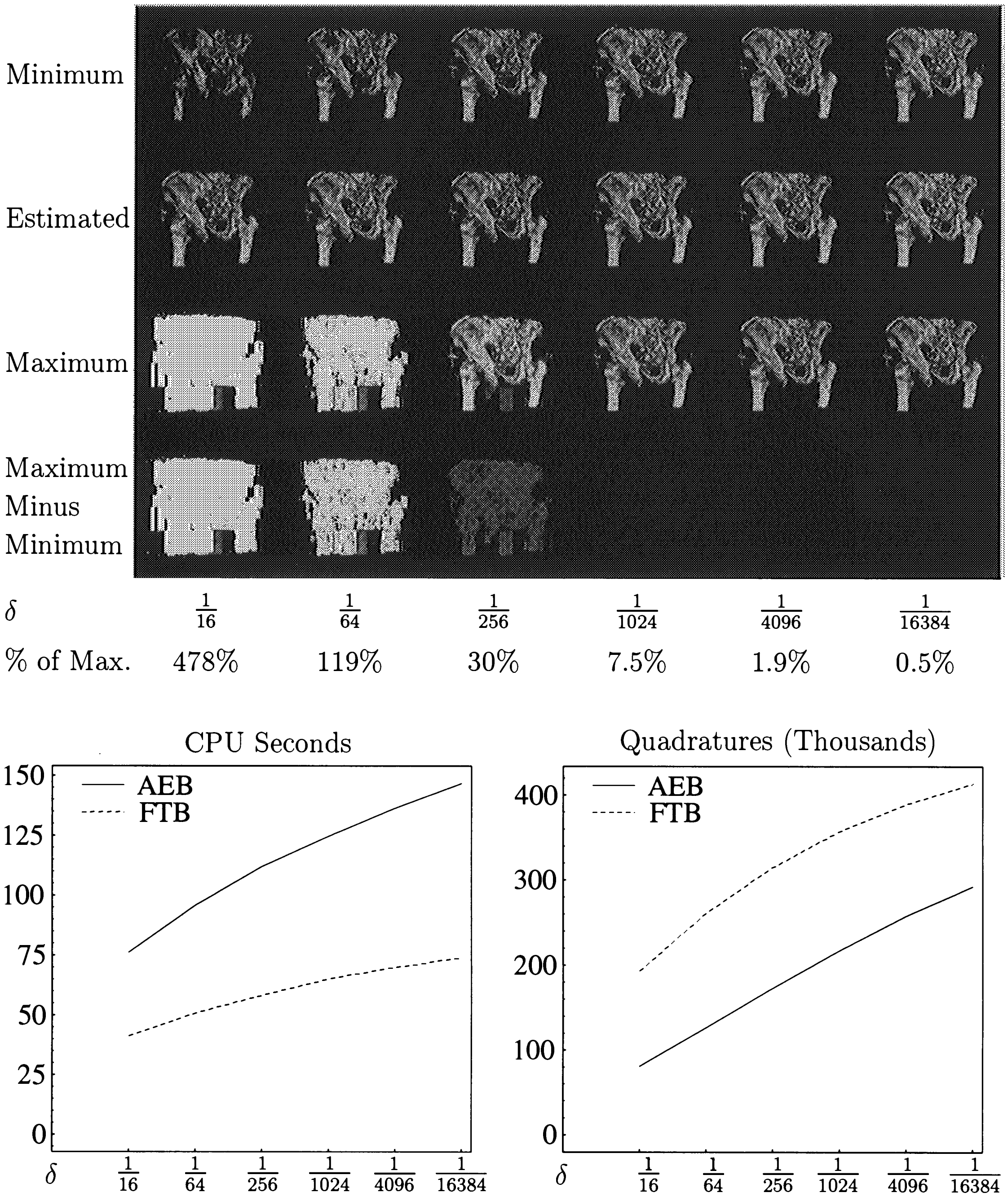


Figure 5.25: Adaptive error bracketing. Results using the CT pelvis dataset

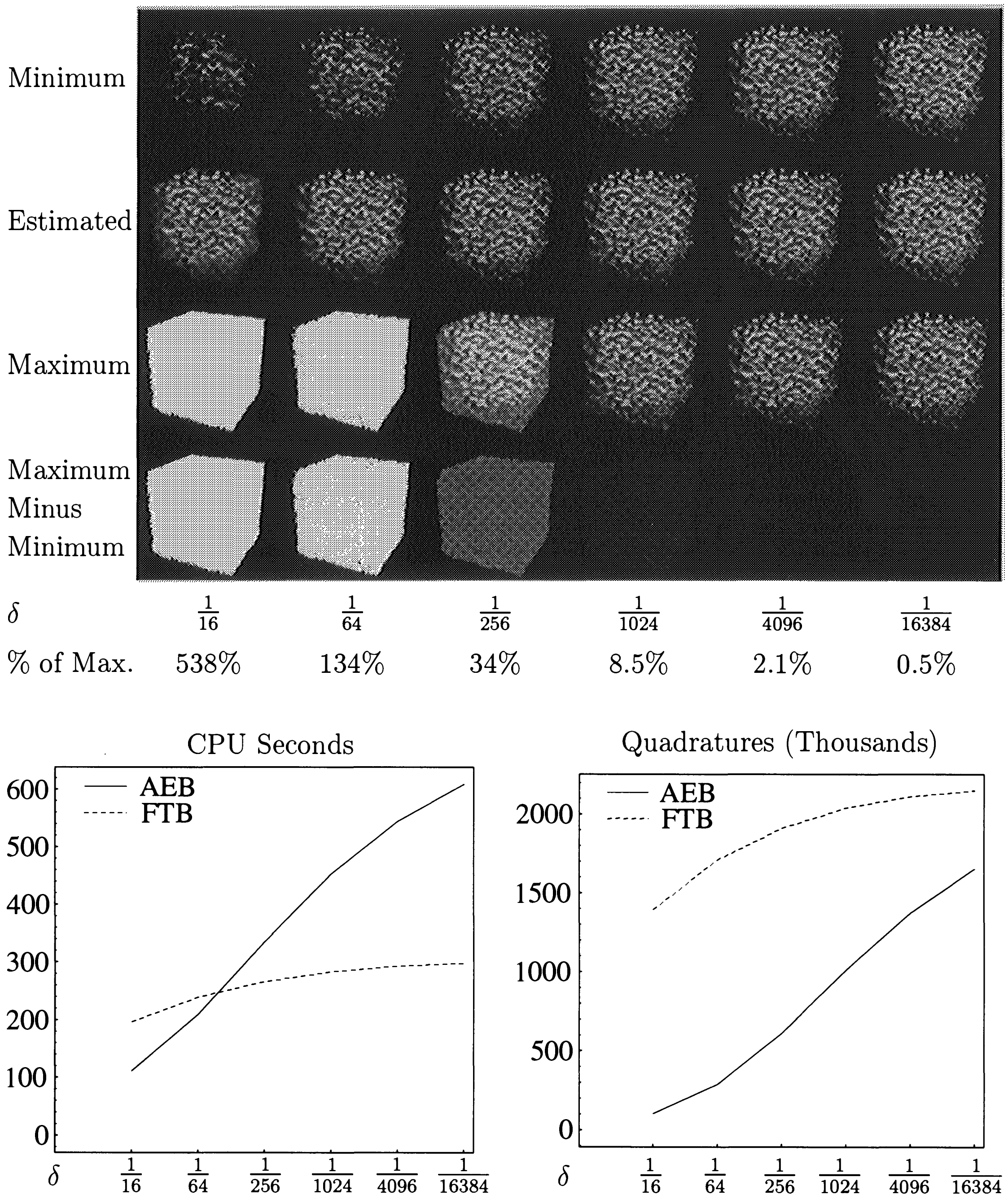


Figure 5.26: Adaptive error bracketing. Results using the SOD enzyme dataset

Chapter 6

Imaging for Cardiology

In this chapter, we present the preliminary results of our ongoing project in rendering cardiac ultrasound data. This case-study will illustrate the difficulties encountered in providing reliable imaging of the types of datasets that are generated by modern scientific scanning equipment.

6.1 The goals of the project

Our continuing project is oriented towards the development of a low-cost, dynamic, four-dimensional imaging system for the human heart *in vivo*. We are interested in developing non-invasive diagnostic tools that are practical for use in a physician's office, rather than at a special-purpose imaging center. The project is performed in collaboration with researchers at the Cornell Medical Center, Hewlett-Packard, The University of Firenze, and ESAOTE Biomedica. Our portion of the project is limited to the display of the 3D data.

The overall medical goals of the project include the display of:

- muscle wall motion, useful for locating "dead" muscle tissue;
- valve action, for early detection of diseases such as Marfan's syndrome; and
- blood-flow patterns, in order to detect malfunctioning valves.

Without the capability for displaying dynamic, three-dimensional projections of the heart, many of these goals are not realizable. For example, it is practically impossible to understand the twisting motion of an abnormal valve flap by examining two-dimensional slices.

The project goals determined both the data acquisition method and the visualization platform that were used. Current clinical choices for gathering 3D cardiac data include x-ray computed tomography (CT), magnetic resonance imaging (MRI), and ultrasound. X-rays result in harmful ionizing radiation, thus CT is considered an invasive technique. MRI systems are non-invasive and have recently been overcoming early problems in providing the spatial and temporal resolution necessary for diagnostic cardiology. However, these systems are currently too large and too expensive for use as a casual diagnostic tool. We chose to base our work on ultrasonic imaging. Ultrasound devices are small and inexpensive. Patients are scanned using an external hand-held probe. The ultrasonic waves that are used in normal diagnostic practice are not considered harmful.

For visualization, we required a mass-produced system with high-end graphics capabilities. Although supercomputers are attractive for their computational power and large RAM caches, we found them to be impractical for our purposes.

Personal computers are widely available, but currently lack both high-end graphics and the resources necessary for handling large datasets. Today's graphics workstations represent a good tradeoff between the two extremes of supercomputers and personal computers, and we decided to design our software for these machines. Workstations are small enough and inexpensive enough to meet our clinical goals. Since the power of personal computers continues to grow exponentially, we expect that our software will be feasible for execution on personal computers in just a few years, thus further reducing the cost of the total system.

6.2 Data Collection

We begin by reviewing the data-collection process. Ultrasonic imaging works in a manner similar to that of radar. An ultrasonic transducer converts electrical energy into ultrasonic energy. It produces an ultrasonic beam that passes through the body. At each interface between structures with differing acoustical impedances, some of the energy is reflected back towards the transducer in the form of an echo. Upon receiving the echo, the transducer converts the acoustical signal into electrical pulses. The time between emission of the beam and reception of the echo is used to infer the depth of the interface. More detailed information concerning the physics of ultrasound may be found in review articles, such as the one by Havlice [HT79].

The process is extended to two dimensions by sequentially reorienting the ultrasound beams at regular angular intervals across a thirty- to ninety-degree arc. The sweep can be performed either by mechanically rocking the transducer or by electronically "beam-steering" the waveform generated by a phased array of transducers. In either case, the system must pause to gather echoes before emitting a beam at the next angular position. It is desirable that the entire sweep be performed in a thirtieth of a second, in order to allow real-time video display of the fan-shaped 2D slice. The speed of ultrasound in the body is approximately 1480 m/s. Assuming instantaneous reorientation of the transducer, up to 308 angular samples are possible across a ninety-degree arc scanned to a depth of 16 cm. Currently available hardware attains 120 angular samples. This corresponds to sampling the plane along scanlines separated by three-quarters of a degree increments. The process of gathering this fan-shaped slice of data is illustrated in Figure 6.1.

The simple extension of this scheme to three dimensions is not feasible since the time required to sweep across even a small-sized volume is too large relative to the speed of motion of the heart. Current technology does not allow us to gather data from multiple beams simultaneously due to the interference that would result from multiple ultrasonic waves. Thus we cannot reconstruct 3D static images by taking a snapshot at a single point in time. Instead, we rely on the fact that heart motion is cyclic. We reconstruct static images of the heart by combining information obtained over several cardiac cycles. First, we take a planar snapshot from a known position and orientation. Then we wait for the heart to return to the same position and take another snapshot from a different orientation. This process is repeated until enough planes of data are available. Then the planes are merged to form a single 3D dataset. The timing of the snapshots is determined by electrical

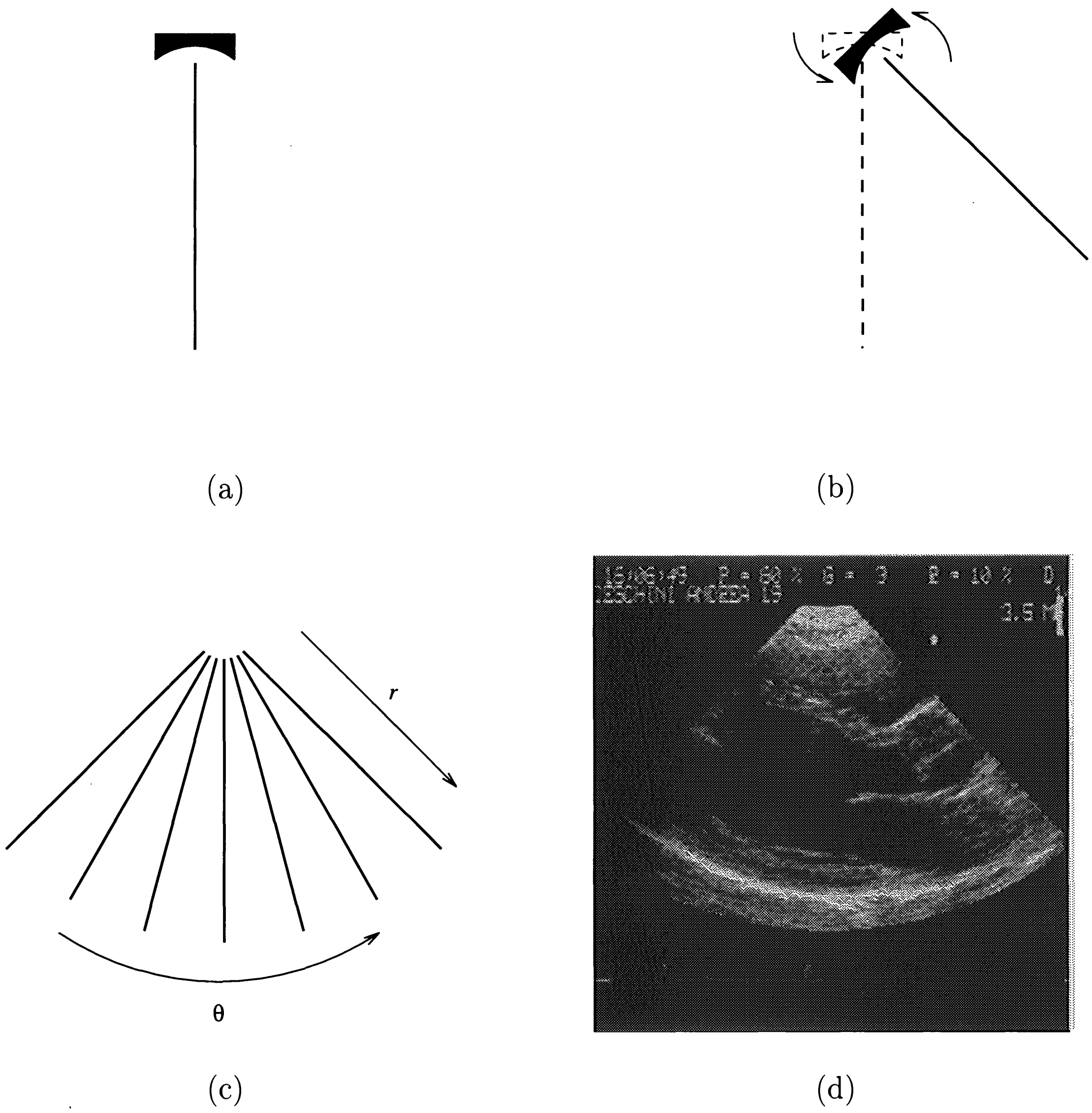


Figure 6.1: Gathering 2D ultrasound data. (a) Inferring depth from the time between emission and echo, an ultrasound transducer is able to gather data along a line that extends into the body. (b) This line may be moved by mechanically rocking the transducer (as shown) or by beam-steering a phased-array of transducers. (c) The transducer is swept over a 90 degree arc, gathering data in acquisition lines from across a 2D plane. Data samples are indexed in polar coordinates. The sweep occurs within 1/30$^{\text{th}}$ sec. (d) A 2D cardiac ultrasound image of a normal volunteer.

signals from the autonomic nervous system that control the heart. These may be measured outside the body by an electrocardiogram (EKG) device. We use the EKG impulse as a reference frame for each heart cycle, and take our snapshots at regular intervals after the pulse. The gating is performed electronically. Note that this technique is not useful for diagnosing diseases that result in acyclic irregularity of cardiac function.

The choice of planes that are used to gather three-dimensional data is limited by the fact that ultrasound waves are attenuated sharply by bone, shadowing any structures at greater depths. Thus, the rib cage presents difficulties for the acquisition of ultrasound data about the human heart. Our colleagues at the University of Firenze, ESAOTE Biomedica and New York Hospital developed an ultrasound probe with a rotating transducer for the acquisition of 3D data [PMM+90, PCM+92]. The new probe can be held fixed while internally, the transducer rotates about its central axis. By sweeping the fan-shaped scan profile across a full arc, we are able to gather data within a cone-shaped region. The radius of the cone is small enough in the near field so that the ultrasound probe may be manually aligned with a gap between two ribs with no intersection of the scanning region with the rib cage. A photograph of the ultrasound system during data acquisition is given in Figure 6.2.

Since we are interested in studying the dynamic aspects of heart function, we collect data at several points in the heart cycle. Because rotation of the transducer is not instantaneous, it is more efficient to gather data from all time steps at a single orientation before rotating the transducer. Data is collected during every

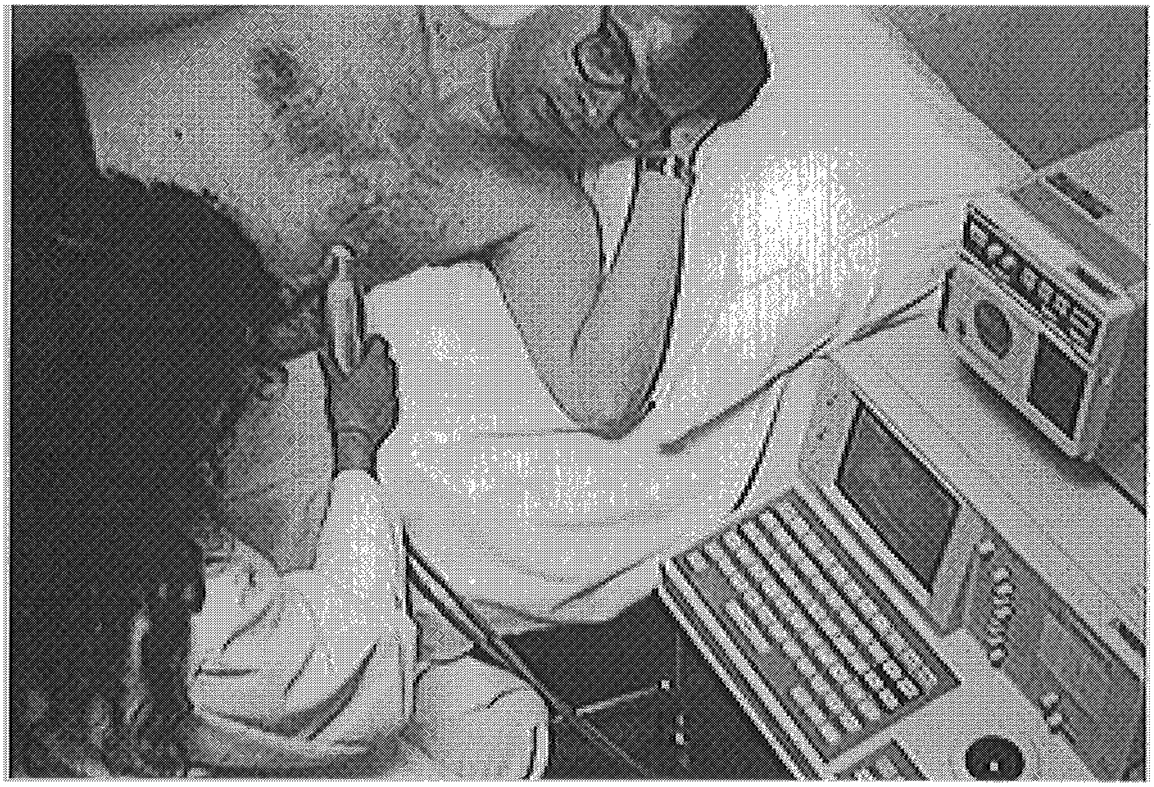
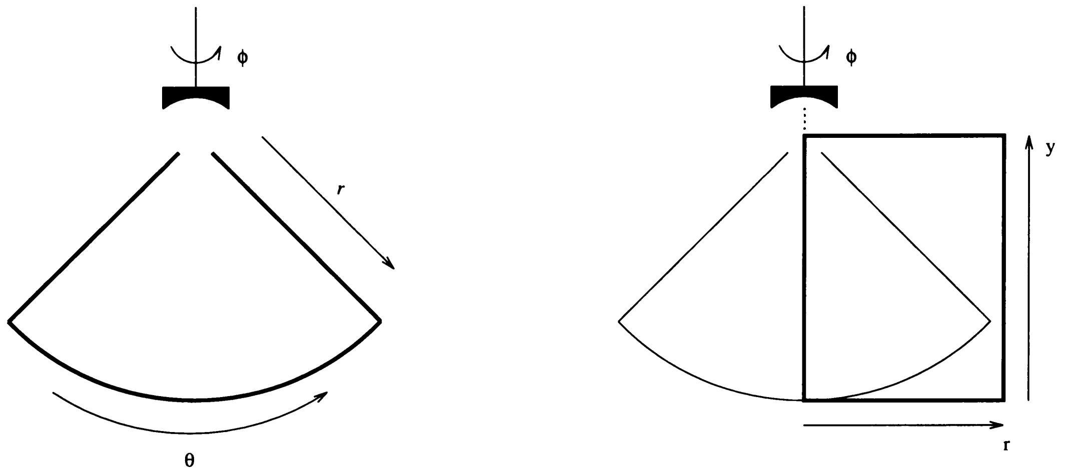


Figure 6.2: The ultrasound system during data collection. The hand-held probe contains a rotating transducer that is controlled by the apparatus at center right. The probe connects to a standard 2D ultrasound device at lower right.



(a) (b)

Figure 6.3: Coordinate axes relative to the transducer for spherical and cylindrical ultrasound data. (a) Spherical coordinate (r, θ, ϕ) axes. (b) Cylindrical coordinate (r, y, ϕ) axes.

other heart cycle; the intervening cycles are used for rotating the transducer. In order to obtain 62 radial slices for every time-step, 124 heart cycles are necessary, requiring 75 to 123 seconds. After the scan, the imaging proceeds without further participation by the patient.

With the addition of the transducer rotation, data is gathered at regularly-sampled points in a spherical coordinate system. The full dataset is indexed by (r, θ, ϕ, t), where r is distance along the acquisition line, θ is the angle of the rocking of the transducer and ϕ is the angle of rotation of the transducer. These axes are shown in Figure 6.3a. The time-step within the heart cycle is given by t.

6.3 Current Limitations

Unfortunately, at the present time, it is not possible for us to obtain digital data in the native spherical coordinate system. The bandwidth and fast RAM requirements for moving and storing the mass of data accumulated by the transducer over short time intervals are too high for practical implementation. While we await developments in this area, we have been able to gather 2D slices of data after they have been scan-converted for display by the ultrasound device. Conversion of the data to video standard requires a resampling onto a 2D rectangular grid. This step is performed by hardware inside the ultrasound device that employs a bilinear interpolation algorithm [LL80]. The scan-converted data is recorded on video tape, which is later digitized for further processing. With scan-conversion of each 2D slice, the data arrives as regular samples in a cylindrical (r, y, ϕ) coordinate system, with the axes defined as in Figure 6.3b. The resolution of the dataset is as follows. Each 2D slice is made up of 512×512 8-bit pixels. Each 3D dataset contains 62 slices. The heart cycle is sampled at 15 time steps. Thus, the total size of the dataset is

$$512 \times 512 \times 62 \times 15 = 243,793,920 \text{ bytes} \approx 232\text{MB}.$$

A great deal of information is lost using the current acquisition system. Consider the many resampling steps that exist in the pipeline:

- Analog ultrasound signals are converted to digital signals.

- The digital signals are resampled onto a rectangular grid.

- The gridded data is converted to analog form for 2D display.

- The 2D display is recorded on video tape.

- The tape is played back into a frame grabber, where the video frames are captured and converted to digital data.

The sum of these steps causes a significant amount of degradation that cannot be accounted for during subsequent processing.

6.4 Visualization of the data

Once the raw data has been digitized, the individual slices must be sorted according to time index. For each time index, each slice is placed in its proper spatial orientation, resulting in a single cylindrical (r, y, ϕ) dataset per time frame. The process of collecting the 3D data is illustrated in Figure 6.4.

Before projection, it is necessary to convert the raw data to absorption and emission functions. As described in Section 3.1.1, the first step is to identify the objects of interest in the raw data. In this case the objects of interest are the muscle walls and valve structures that make up the heart. The internal textures of these tissues result in echoes throughout their mass, not just at the interfaces. Since we are using parasternal views, the heart is the only internal organ present in the scans. Classification is largely a matter of filtering out weak echoes, caused by ultrasound interactions with blood cells, and strong "speckle" artifacts caused by anisotropic reflection of ultrasound waves by structures such as the cartilaginous fibers in the ventricle. We chose to use isosurface techniques to extract the

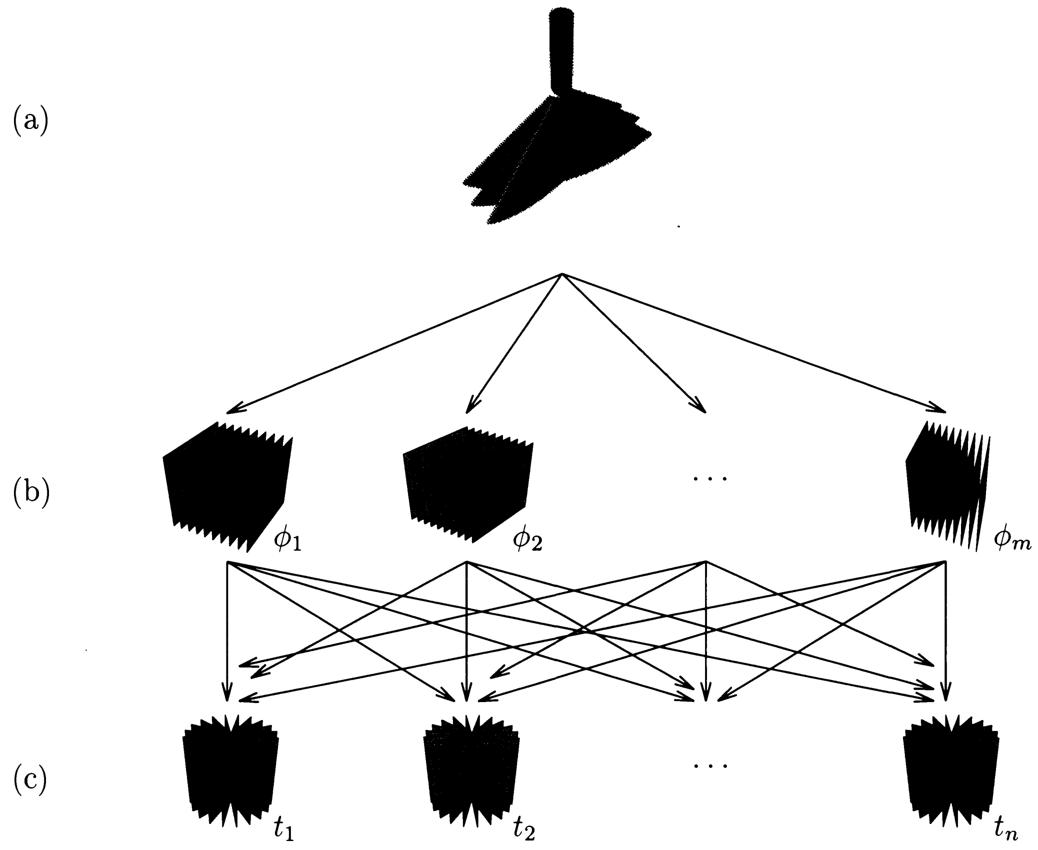


Figure 6.4: Gathering 3D ultrasound data. The transducer gathers data at m angular positions. Illustration (a) shows the transducer and the fan-shaped slices resulting from three transducer orientations. At each orientation, data is gathered for n time intervals. After scan-conversion, the dataset is recorded onto videotape as m orientations, $\phi_1, \ldots, \phi_m$, with n rectilinear slices per orientation, as shown in illustration (b). These slices are digitized and then sorted according to time index. The result is n cylindrical 3D datasets, $t_1, \ldots, t_n$, as shown in illustration (c).

midrange of echo intensities. The algorithm that we used was a direct implementation of Levoy's isosurface extraction algorithm [Lev88] that was described in Section 3.1.1. Our results were acceptable as a first approach, even though noise and speckle artifact were not completely eliminated. Once classification was completed, absorption parameters were assigned so that the muscle wall appeared as a solid mass. Emission characteristics were determined by computing the gradient of the data and applying the Phong shading model, as described in Section 3.1.3.

The output of the above process is a dataset with two scalar quantities at each sample point: an absorption coefficient and an emission coefficient. We have investigated three approaches to projecting this data that arrives as regular samples across a cylindrical grid.

The first approach was to resample the data onto a regular rectilinear grid. Resampling enables the use of the wide range of projection algorithms that are currently available for this data topology, including the controlled-precision techniques presented in this thesis. We chose to use a ray-casting algorithm and a perspective camera model. The expensive conversion of sample points from cylindrical to rectilinear coordinates is performed only once. Problems arise using this method because the original dataset samples Cartesian space nonuniformly. If the rectilinear grid samples the core of the dataset adequately, it oversamples the outer regions, imposing high storage costs. Conversely, if the outer regions are sampled adequately the core is undersampled, potentially resulting in the omission of small details. This problem is presented in Figure 6.5. An additional drawback to this approach is that the original data is not sampled above the Nyquist frequency, therefore resampling is guaranteed to introduce error. We are currently unable to quantify the magnitude of this error.

A second approach was based on the splatting technique that we described in Section 3.3.4. We developed a new splatting algorithm that operates in cylindrical coordinates. Splatting is difficult to apply to cylindrical data since the shape of the filter kernels must vary over space to account to the varying sampling rate. Ideally the spherical filter kernels that are used for rectilinear data need to be stretched and bent along the curvature of the cylindrical grid. We chose to approximate this transformation by merely stretching the kernels along the axis perpendicular to each radial line. This results in kernels with the same ellipsoid shape that has been studied in the splatting literature [Wes90,LH91]. An illustration comparing the coverage of a 2D polar-coordinate plane using uniform circular filter kernels and elliptical filter kernels is illustrated in Figure 6.6. Once the proper orientation and stretching of a kernel is determined, it may be projected using the hierarchical splatting technique and a hardware z-buffer [LH91]. This technique is an improvement over the resampling method in that it operates in the native coordinate system of the data. However, it spends most of its computational energy computing the splats for the multitude of sample points at the core. This is unfortunate since the screen-space contribution of these sample points is minimal. The largest drawback of the approach is the cascade of approximations that are employed. In raw form, splatting introduces error due to the improper treatment of filter kernel overlap. Hierarchical splatting approximates the raw splats by Gouraud-shaded polygons and approximates image compositing operations with

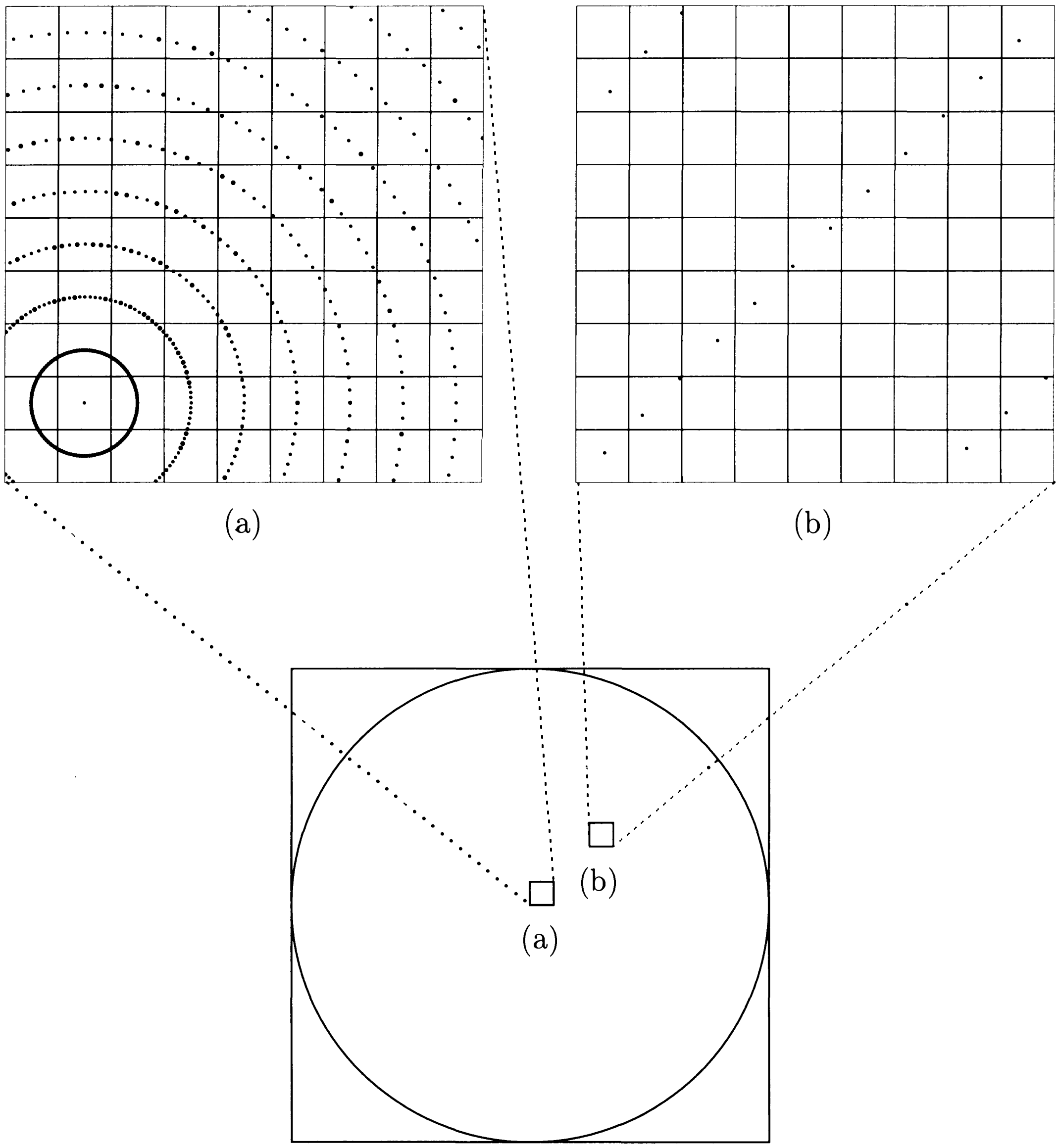


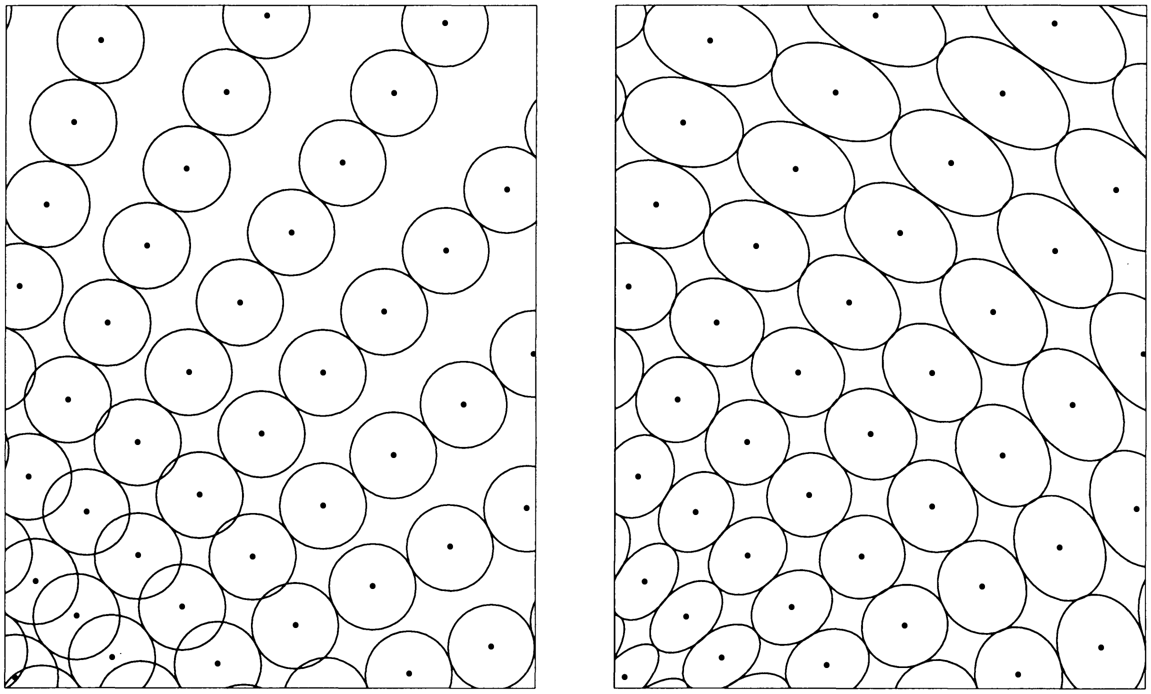
Figure 6.5: Resampling a cylindrical grid onto a rectilinear grid. In this example the cylindrical grid has 128 radial lines, each containing 256 samples. The spacing between rectilinear grid lines was chosen to be equal to the spacing between samples along a single radial line. Two portions of a horizontal planar slice of the dataset are expanded. (a) Note that near the core, many cylindrical samples fall within each rectilinear cell. (b) A short distance away, most cells contain no cylindrical samples.

fixed-precision arithmetic. Our modification of the technique adds further inaccuracy due to the approximation of the ideal cylindrical splat shape.

Our third approach was to use a ray-casting algorithm that was adapted to operate in cylindrical coordinates. We took evenly-spaced sample points along the ray and computed the volume rendering integral using the piecewise-constant approximation. Thus the algorithm was similar in spirit to the one presented by Levoy [Lev88]. Each sample point along the ray was converted from (x, y, z) coordinates to (r, y, ϕ). The sample points at the corners of the enclosing cell were located, and trilinear interpolation along each of the cylindrical-coordinate axes was used. The drawback of this approach is that choosing an acceptable uniform sampling rate along the ray is difficult. The situation is analogous to that encountered when resampling the data onto a regular grid, as shown in Figure 6.7. In this instance, because of concerns about computational expense, we chose to undersample the core. While it is desirable to spend less effort computing the contribution of the core, this method ignores large numbers of samples entirely, thus introducing the possibility of gross inaccuracy.

6.5 Discussion

Our studies with these three algorithms revealed the many difficulties in applying current-technology volume rendering algorithms solving to real-world scientific visualization problems. The techniques produced images that, when viewed individually, appeared plausible. However, they differed markedly from one another. Images produced by each of the methods with the same dataset is given in Fig-



(a) (b)

Figure 6.6: Comparison of uniform circular splats to nonuinform elliptical splats in polar coordinates. (a) Uniform splats do not account for the varying spatial sampling rate. Filters close to the coordinate system origin overlap while gaps appear between filters in more distant regions. (b) The circular splats are stretched along the axis perpendicular to the radial lines. A more uniform coverage of the plane is attained even though ideally, the splats would be bent to conform to curvature of the radial axis as well. For the purpose of illustration, we chose to stretch the filters until they "just touch" one another. In practice, a uniform fraction of overlap is maintained.

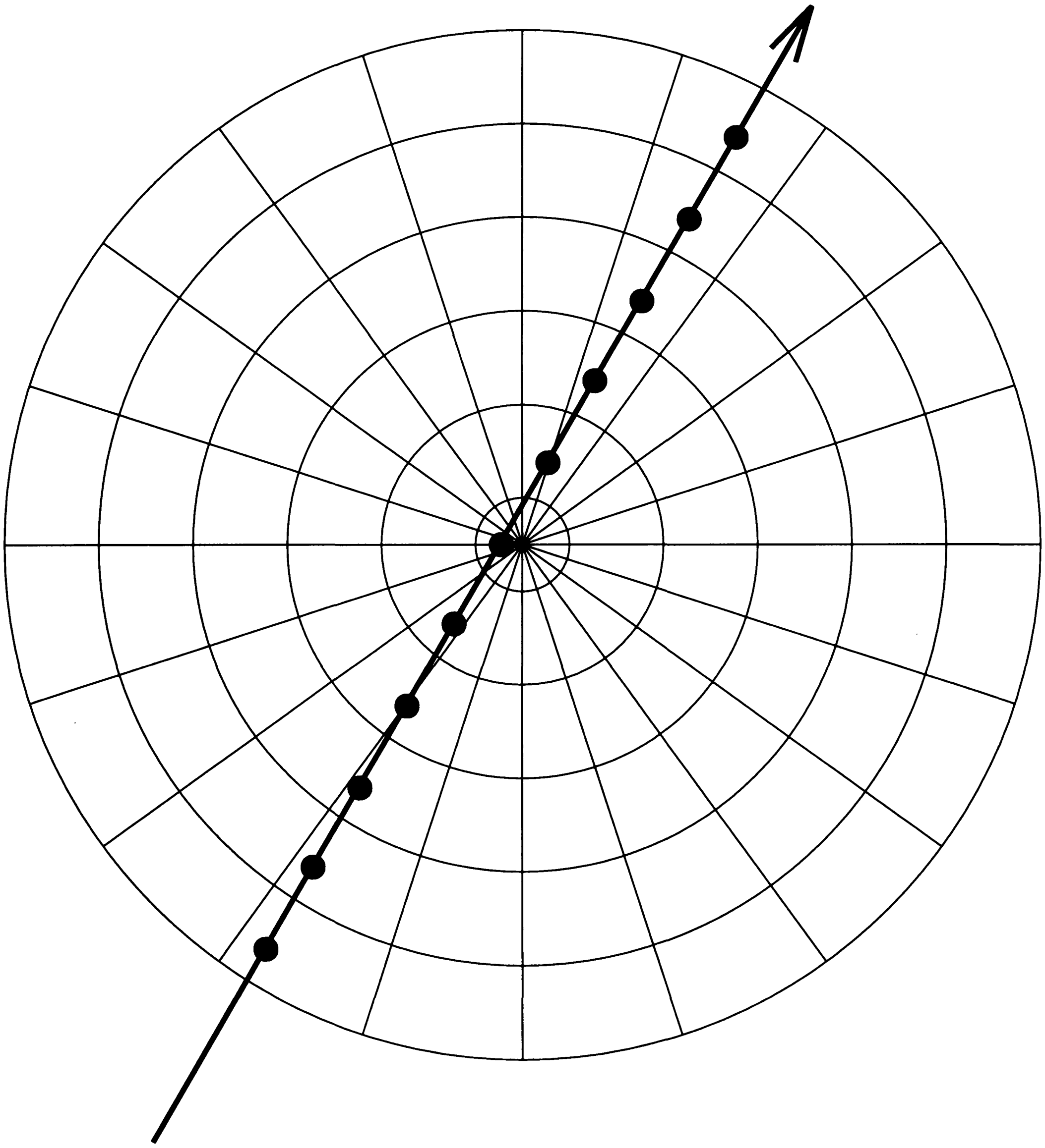


Figure 6.7: Ray-casting onto a cylindrical-coordinate grid. When evenly-spaced sample points are taken, some voxels at the core may not be sampled at all, as shown in this diagram. The cost of using a regular sampling rate that ensures that all voxels in the ray's path are considered is prohibitively high.

ure 6.8. It was the disparity that is evident in side-by-side comparisons that led us to begin an error analysis of the volume rendering process.

There are many sources of error in the process used in our cardiac imaging system. They include:

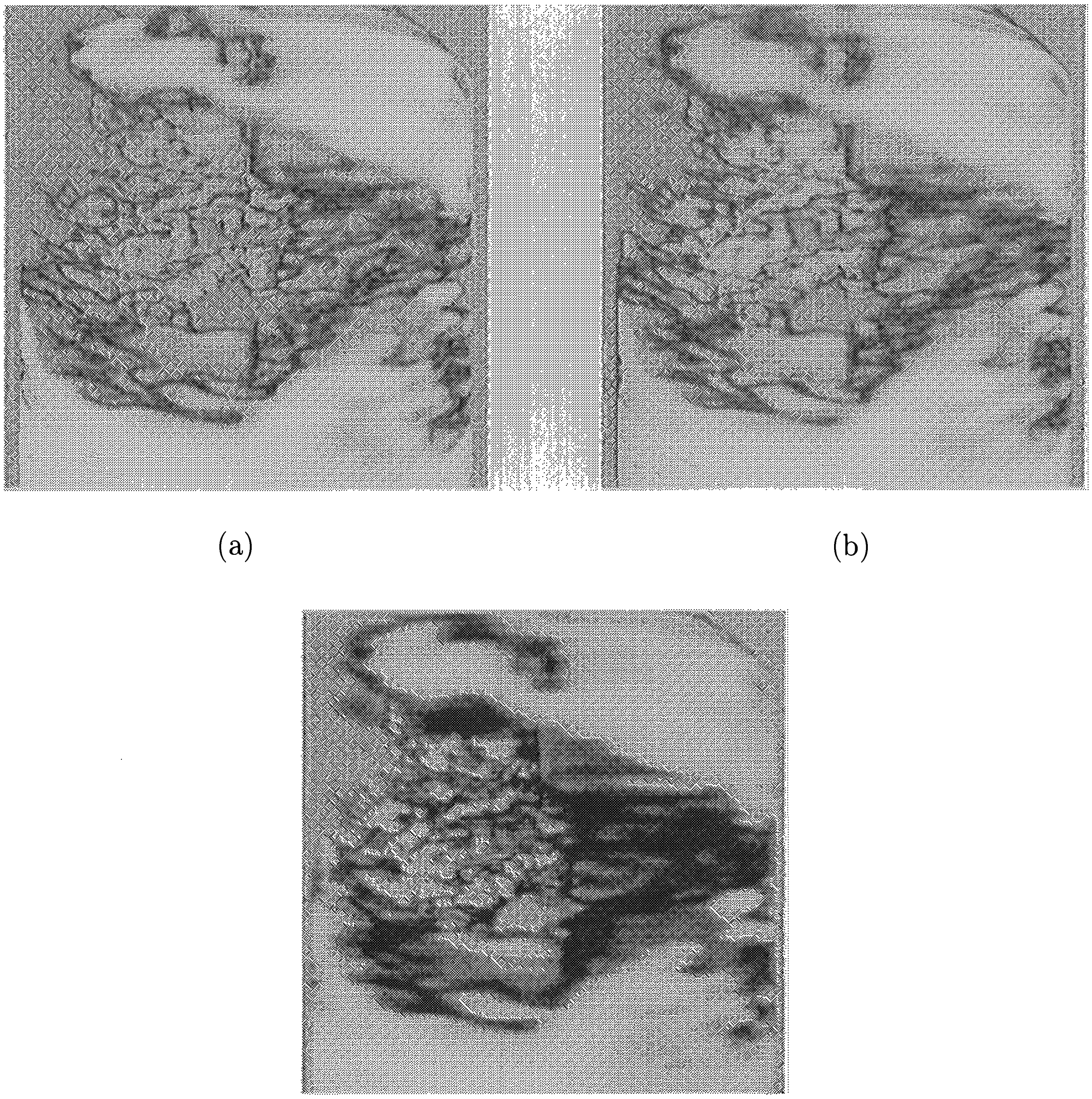
Acquisition Error. The data that arrives for rendering contains inaccuracies, including limited spatial and temporal resolution, noise, inaccurate positioning of the transducer, and lossy scan-conversion.

Reconstruction Error. The volume rendering algorithms operate on 3D scalar fields. Thus, a reconstruction filter must be applied to the discrete data samples that are produced during acquisition. Due to practical considerations, ideal filters cannot be used, thus error is introduced during this phase. The process is complicated by the fact that data does not arrive as samples on a rectilinear grid.

Classification error. The raw data does not correspond to the parameters of emission and absorption that are required by the rendering model. The raw data must be interpreted to produce these values. Incorrect classification will result in misleading images.

Rendering Error. Once the functions of emission and absorption are determined, the volume rendering model must be evaluated. Since analytic expressions for the resulting integrals are not known, approximation error is introduced.

The controlled-precision methods that were introduced in this thesis address the problems of rendering error. We presented techniques for computing individual rays to any specified precision, given polynomial or piecewise-polynomial functions of absorption and emission. We also developed an algorithm that, in principle, will



(a) (b)

(c)

Figure 6.8: Comparison of three methods for rendering cylindrical-coordinate data: (a) Ray casting with the original grid; (b) Ray casting after resampling onto a regular grid; (c) Cylindrical splatting. The variation in appearance is due to the fact that each method employs a different set of approximations.

compute accurate reconstructions of screen space.

This work needs to be extended so that the controlled-precision algorithms can operate directly on cylindrical and spherical coordinate data. This will eliminate a source of resampling error. In order to apply the techniques, it will be necessary to derive analytic expressions for the interpolated functions of emission and absorption as they pass through the cells of a curvilinear-coordinate grid. While the evaluation of the resulting expressions is likely to be expensive, the images that are produced can be presented with known accuracy. Using a cylindrical-coordinate version of the adaptive error bracketing algorithm might help to alleviate some of the problems of computational expense. We expect that such an algorithm would be able to use coarse approximations within the core while maintaining reliable error bounds for the final values. Thus, an efficiency that was not attainable with any of our existing rendering techniques would become available.

As the above listing implies, there are many more sources of error that need to be investigated. The logical next step would be to quantify the error due to non-ideal reconstruction. This analysis should be performed so that it applies to data that is not regularly sampled in any coordinate system. This will be useful, for example, since we know that the hand-held transducer does not remain perfectly fixed in space during a two-minute scanning session. Variation in the position and orientation of the transducer will result in data that arrives on arbitrarily positioned planes. Precise position and orientation information will be necessary before we can begin to consider this effect in our analysis.

Although more research is needed in all aspects of our cardiac imaging project, we are encouraged by our results thus far. Our basic goal of a low-cost, non-invasive diagnostic imaging tool still appears to be attainable. Although many of the difficulties we face are unique to our project, we find that similar obstacles arise when imaging any real-world volume data. Our results thus far will help scientists by providing guarantees of accuracy of the computed projections. Our results will also provide a starting point for further error analysis of the volume rendering process.

Chapter 7

Conclusion and Future Directions

Three major contributions to the field of volume rendering have been presented:

Controlled-precision volume projection. We developed techniques for computing the intensity accumulated along individual viewing rays to any specified precision. We presented four approaches for solving the emission-absorption integral across individual ray-voxel intersections; piecewise-constant bounds, trapezoid rule, Simpson's rule, and a power series method that is practical for computing high-precision solutions. These results were extended to compute the emission-absorption integral to any specified precision for rays that pass through multiple voxels. Our approaches represent a departure from existing volume rendering algorithms in that they allow quantitative assessment and user control over the quality of solutions to the line integral that are produced.

Adaptive error bracketing. Adaptive error bracketing addresses the problem of efficiently computing point-samples of screen-space to any specified precision.

Using this method, computation is directed toward large sources of error, and easy-to-compute approximations are used whenever possible. The approach allows continual refinement of the solution until a user-specified error tolerance is met, thus avoiding overcomputation while maintaining guaranteed error bounds. The algorithm allows processing of the data without imposing a strict front-to-back or back-to-front evaluation order. Our studies have shown that adaptive error bracketing saves significant computational effort when working with smooth datasets or when computing low-accuracy images.

Efficient perspective projection. We also presented efficient volume rendering algorithms for computing perspective projections. An optimal paging strategy was introduced that makes ray-casting practical in situations where the dataset is too large to fit into RAM memory. A ray-splitting approach to adaptive supersampling was also presented. This technique ensures that all data features are accounted for in the final image, while at the same time avoiding overcomputation in regions of the dataset that are near the eyepoint.

In the introduction to this thesis, we outlined several shortcomings of the state-of-the art in volume rendering. While the above contributions represent an improvement in terms of both accuracy and efficiency, there remains much work to be done:

Accuracy. This thesis provides tools for controlling one source of error in the volume rendering process: approximating the value of the illumination function at point-samples. An important next step is to extend these results to include controlling projection error across entire areas of screen-space. In addition, there are many other sources of error in volume rendered images that need to be addressed. These include non-ideal reconstruction of functions in object-space and in screen-space, and the propagation of errors that are introduced during data acquisition.

Efficiency. The techniques presented in this thesis can reduce computational and storage requirements so that volume-rendered images may be produced in several minutes on workstation-level machines. An important next step is to provide methods providing scientists with interactive feedback. Since the adaptive error bracketing algorithm allows progressive refinement along individual rays, we believe it will serve as a good starting point for this type of research.

Arbitrary data formats. The main research contributions of this thesis all assume a restricted data format; only scalar data organized on a regular rectilinear grid was considered. While the theoretical work in controlled-precision projection remains valid for any scalar field, there remain many unresolved implementation issues. These include proper reconstruction of non-rectilinear data and the intersection of a ray with irregularly-shaped voxels. Generalization of the algorithms to the visualization of multiple scalar and vector fields await further research in techniques for the effective display of such information.

Clearly, significant research is necessary to transform volume rendering into a reliable general-purpose visualization tool. We hope that the contributions of this thesis represent a step in the direction of solving this larger problem.

Appendix A

Nomenclature

This chapter is a summary of the nomenclature that is defined in the thesis. It is included only for reference.

Symbol	Definition	Page Introduced
α	Absoprtion	7
ϵ	Emission	8
$I(a, b)$	Intensity along a ray between points a and b	7
$\kappa(t)$	Integrating Factor	8
ρ	Density	11

Symbol	Definition	Page Introduced
$V(v)$	Optical distance from a point on a ray to the light source	14
$\Phi(u)$	Reflectance at a point on a ray	14
ξ	Raw data value	24
$T(a,b)$	Transparency along a ray between points a and b	29
$f(t)$	The volume rendering integrand	62
$P_n(t)$	Power series approximant	75
$R_n(t)$	Power series remainder	75

Interval Arithmetic

As used in this thesis, intervals are contigous regions of the real axis. The interval bounded by scalars a and b is denoted by the bracketed pair $[a, b]$. Variables representing intervals are denoted by variables in bold letters, *i.e.* $\boldsymbol{\alpha}$, $\boldsymbol{\epsilon}$. Additional notation regarding intervals is given in Table A.1.

Table A.1: Notation for interval arithmetic. The definitions below apply to the interval $\boldsymbol{x} = [a, b]$

Notation	Meaning	Definition
$\boldsymbol{x}^-$	Lower bound on $\boldsymbol{x}$	a
$\boldsymbol{x}^+$	Upper bound on $\boldsymbol{x}$	b
$\overline{\boldsymbol{x}}$	Midpoint of $\boldsymbol{x}$	$\frac{1}{2}(a+b)$
$\\| \boldsymbol{x} \\|$	Width of $\boldsymbol{x}$	$b - a$

Appendix B

Power Series Convergence Proof

In this section we prove that $\lim_{n\to\infty} R_n(t) = 0$ for all t. Equation (4.31) expresses $R_n(t)$ as the sum of a finite number of terms T_{n-1} where

$$T_{n-1}(t) = t^n \frac{\overline{q}^{(j)}\,\overline{A}_{n-j}}{j!}.$$

Since j is fixed, these terms are of the form

$$K_1\, t^n\, \overline{A}_{n-j}$$

where K_1 is the constant $\overline{q}^{(j)}/j!$. It will be sufficient to show that

$$\lim_{n\to\infty} T_n = 0.$$

We first show that $\{T_1(t), T_2(t), \ldots\}$ is bounded for all t, from which convergence to zero will follow easily.

Substituting for $\overline{A}$ from Equation (4.33) yields

$$T_{n-1} = t^n \frac{K}{n-j} \sum_{k=1}^{N} \frac{\overline{p}^{(k)}\overline{A}_{n-j-k}}{(k-1)!}$$

for $n-1 \geq N+j$. Letting

$$K_2 = K_1 \max_{k=1}^{N} \frac{\overline{p}^{(k)}}{(k-1)!},$$

we have

$$\begin{aligned} T_{n-1} &\leq t^n \frac{K_2}{n-j} \sum_{k=1}^{N} \overline{A}_{n-j-k} && \text{(B.1)} \\ &= t^n \frac{K_2}{n-j} \sum_{k=1}^{N} \frac{T_{n-k}}{t^{n-k}} && \text{(B.2)} \\ &= \frac{K_2}{n-j} \sum_{k=1}^{N} T_{n-k} t^k. && \text{(B.3)} \end{aligned}$$

Now letting

$$K_3 = K_2 \max_{k=1}^{N} |t^k|,$$

we have

$$T_{n-1} \leq \frac{K_3}{n-j} \sum_{k=1}^{N} T_{n-k}. \qquad \text{(B.4)}$$

As $n \to \infty$, $T_{n-1}(t)$ cannot grow without bound, since when

$$\frac{K_2}{n-j} < \frac{1}{N},$$

$T_{n-1}(t)$ is strictly less than the average of its preceding N values, and hence strictly less than the maximum of these values. Thus after a finite number of terms $T_n(t)$ is a decreasing sequence. Therefore we may bound the sequence

$$T_n \leq \max\{T_1(t), T_2(t), \ldots\} = B < \infty.$$

Finally, from Equation (B.4) it follows that

$$T_{n-1} \leq \frac{K_3 \, N \, B}{n} = \frac{K_4}{n}$$

and

$$\lim_{n\to\infty} \frac{K_4}{n} = 0.$$

This completes the theorem. □

Note that T_n, and hence R_n is bounded over all j and all $t \in [a, b]$. This bound allows us to interchange limiting operations and integration,

$$\int_a^b \lim_{n\to\infty} R_n = \lim_{n\to\infty} \int_a^b R_n$$

which we use implicitly in Chapter 4.

BIBLIOGRAPHY

[Act70] Forman S. Acton. *Numerical Methods That Work*. Harper & Row, New York, 1970.

[ASK92] Ricardo S. Avila, Lisa M. Sobierajski, and Arie E. Kaufman. Towards a comprehensive volume visualization system. In *Proceedings: Visualization '92*, pages 13–20. IEEE Computer Society, 1992.

[BFG86] Larry Bergman, Henry Fuchs, and Eric Grant. Image rendering by adaptive refinement. *Computer Graphics*, 20(4):29–38, August 1986.

[Bli82] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics*, 16(3):21–29, July 1982.

[BT75] Phong Bui-Tong. Illumination for computer-generated pictures. *Communications of the ACM*, pages 311–317, June 1975.

[Cha60] S. Chandrasekar. *Radiative Transfer*. Dover Publications, New York, 1960.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, July 1984.

[CT82] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions of Graphics*, 1(1):7–24, 1982.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, August 1988.

[DH92] John Danskin and Pat Hanrahan. Fast algorithms for volume ray tracing. In *1992 Boston Workshop on Volume Visualization*, pages 91–98. ACM SIGGRAPH, 1992.

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Nayloy. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, July 1980.

[FvFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice, Second Edition.* Systems Programming Series. Addison-Wesley, Reading, Massachusetts, 1990.

[Gla89] Andrew S. Glassner, editor. *An Introduction to Ray Tracing.* Academic Press, San Diego, California, 1989.

[Han90] Pat Hanrahan. Three-pass affine transforms for volume rendering. *Computer Graphics*, 24(5):71–78, November 1990.

[HB86] Karl Heinz Höhne and Ralph Bernstein. Shading 3D-images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging*, MI-5(1):45–47, March 1986.

[HBP+89] Karl Heinz Höhne, Michael Bomans, Andreas Pommert, Martin Riemer, Carsten Schiers, Ulf Tiede, and Gunnar Wiebecke. 3D visualization of tomographic data using the generalized voxel model. In *Proceedings of the 1989 Chapel Hill Workshop on Volume Visualization*, pages 51–57, May 1989.

[HBP+90] Karl Heinz Höhne, Michael Bomans, Andreas Pommert, Martin Riemer, Carsten Schiers, Ulf Tiede, and Gunnar Wiebecke. 3D visualization of tomographic data using the generalized voxel model. *The Visual Computer*, 6(1):28–36, 1990.

[HT79] James F. Havlice and Jon C. Taenzer. Medical ultrasonic imaging: An overview of principles and instrumentation. *Proceedings of the IEEE*, 67(4):620–641, April 1979.

[HTSG91] Xiao Dong He, Kenneth E. Torrance, François X. Sillion, and Donald P. Greenberg. A comprehensive physical model for light reflection. *Computer Graphics*, 25(4), August 1991.

[KK89] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. *Computer Graphics*, 23(3):271–280, July 1989.

[Kru90] Wolfgang Krueger. Volume rendering and data feature enhancement. *Computer Graphics*, 24(5):21–26, November 1990.

[KS88] Avinash C. Kak and Malcom Slaney. *Principles of Computerized Tomographic Imaging.* IEEE Press, New York, 1988.

[KV84] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. *Computer Graphics*, 18(3):165–173, July 1984.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[Lev89] Marc Levoy. *Display of Surfaces from Volume Data.* Ph.D. dissertation, The University of North Carolina at Chaptel Hill, 1989. Also published as Department of Computer Science Technical Report TR89-022.

[Lev90a] Marc Levoy. Efficient ray tracing of volume data. *Transactions on Graphics*, 9(3):245–261, July 1990.

[Lev90b] Marc Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, February 1990.

[Lev92] Marc Levoy. Volume rendering using the Fourier projection-slice theorem. Technical Report CSL-TR-92-521, Stanford University Computer Systems Laboratory, April 1992.

[LH91] David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics*, 25(4):285–288, July 1991.

[LL80] H. G. Larsen and S. C. Leavitt. An image display algorithm for use in real-time sector scanners with digital scan converters. In B. R. McAvoy, editor, *1980 Ultrasonics Symposium Proceedings*, pages 763–765. IEEE Group on Sonics and Ultrasonics, November 1980.

[LTG92] Dani Lischinski, Fillipo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.

[Luc92] Bruce Lucas. A scientific visualization renderer. In *Proceedings: Visualization '92*, pages 227–233. IEEE Computer Society, 1992.

[Mal90] Tom Malzbender. Fourier volume rendering. In *Proceedings of the 1990 Hewlett-Packard Graphics Symposium*, 1990.

[Mal93] Tom Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(3), July 1993.

[MHC90] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics*, 24(5):27–33, November 1990.

[Moo66] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[NA92] Kevin L. Novins and James R. Arvo. Controlled precision volume integration. In *1992 Boston Workshop on Volume Visualization*, pages 83–89. ACM SIGGRAPH, 1992.

[NAS92] Kevin L. Novins, James R. Arvo, and David H. Salesin. Adaptive error bracketing for controlled-precision volume rendering. Technical Report TR92-1312, Cornell University Department of Computer Science, Ithaca, New York, November 1992.

[NH92] Paul Ning and Lambertus Hesselink. Vector quantization for volume rendering. In *1992 Boston Workshop on Volume Visualization*, pages 69–74. ACM SIGGRAPH, 1992.

[NSG90] Kevin L. Novins, François X. Sillion, and Donald P. Greenberg. An efficient method for volume rendering using perspective projection. *Computer Graphics*, 24(5):95–102, November 1990.

[OS75] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[PC82] M. Potmesil and I. Chakravarty. Synthetic image generation with a lens and aperture camera model. *ACM Transactions on Graphics*, 1(2):85–108, April 1982.

[PCM+92] R. Pini, M. Costi, G.A. Mensah, L. Masotti, K. L. Novins, D. P. Greenberg, B. Greppi, M. Cerofolini, and R.B. Devereux. Computed tomography of the heart by ultrasound. In *Computers in Cardiology 1992*, pages 17–20. IEEE Computer Society, 1992.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics*, 18(3):253–259, July 1984.

[PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipies in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 1988.

[PMM$^+$90] Riccardo Pini, Elisabetta Monnini, Leonardo Masoti, Kevin L. Novins, Donald P. Greenberg, Barbara Greppi, Marino Cerofolini, and Richard B. Devereux. Echocardiographic three-dimensional visualization of the heart. In Karl Heinz Höhne et al., editor, *3D Imaging in Medicine: Algorithms, Systems, Applications*, pages 197–215. Springer-Verlag, Berlin, 1990. NATO ASI Series, Volume F 60.

[Rat72] Floyd Ratliff. Contour and contrast. *Scientific American*, 226(6):91–101, June 1972.

[RB74] Earl D. Rainville and Phillip E. Bedient. *Elementary Differential Equations.* Macmillan, New York, 1974. Fifth Edition.

[RT87] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. *Computer Graphics*, 21(4):293–302, July 1987.

[Rus88] Holly Edith Rushmeier. *Realistic Image Synthesis for Scenes with Radiatively Participating Media.* Ph.D. dissertation, Cornell University, Ithaca, New York, May 1988.

[Sab88] Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics*, 22(4):51–57, August 1988.

[SB80] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980. Translated by R. Bartels, W. Gautschi, and C. Witzgall.

[SB87] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics*, 21(4):119–128, July 1987.

[SN89] Peter Shirley and Henry Neeman. Volume visualization at the center for supercomputing research and development. In *Proceedings of the 1989 Chapel Hill Workshop on Volume Visualization*, pages 17–20, May 1989.

[SNC90] George W. Sherouse, Kevin L. Novins, and Edward L. Chaney. Computation of digitally reconstructed radiographs for use in radiotherapy treatment design. *International Journal of Radiation Oncology, Biology and Physics*, 18(3):651–658, March 1990.

[Sob63] V. V. Sobolev. *A Treatise on Radiative Transfer.* D. Van Nostrand Company, Princeton, New Jersey, 1963. Translated by S.I. Gaposchkin.

[SSW86] Daniel S. Schlusselberg, Wade K. Smith, and Donald J. Woodward. Three-dimensional display of medical image volumes. In *Proccedings of NCGA '86*, pages 114–123. National Computer Graphics Association, March 1986.

[ST90] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, November 1990.

[TF79] George B. Thomas, Jr. and Ross L. Finney. *Calculus and Analytic Geometry.* Addison-Wesley, New York, fifth edition, 1979.

[TL93] Takashi Totsuka and Marc Levoy. Frequency domain volume rendering. *Computer Graphics*, pages 271–278, August 1993.

[UK88] Craig Upson and Michael Keeler. V-buffer: Visible volume rendering. *Computer Graphics*, 22(4):59–64, August 1988.

[VPR+92] Dirk Vandermeulen, Peter Plets, Steven Ramakers, Paul Suetens, and Guy Marchal. Integrated visualization of brain anatomy and cerebral blood vessels. In *1992 Boston Workshop on Volume Visualization*, pages 39–46. ACM SIGGRAPH, 1992.

[Wat70] G. S. Watkins. A real time visible surface algorithm. Ph.D. Thesis UTEC-CSc-70-101, Computer Science Department, University of Utah, June 1970. NTIS AD-762 004.

[Web90] Robert E. Webber. Ray tracing voxel data via biquadratic local surface interpolation. *The Visual Computer*, 6(1):8–15, 1990.

[Wes89] Lee Westover. Interactive volume rendering. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, May 1989.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[WG91] Jane Wilhelms and Allen Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics*, 25(4):275–284, July 1991.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[Wil83] L. Williams. Pyramidal parametrics. *Computer Graphics*, 17(3):1–11, July 1983.

[WM92] Peter L. Williams and Nelson Max. A volume density optical model. In *1992 Boston Workshop on Volume Visualization*, pages 61–68. ACM SIGGRAPH, 1992.