High Quality Volume Rendering

Joshua John Lawrence

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

March 1999

Abstract

While volume rendering is a well researched topic, many existing renderers sacrifice accuracy for execution speed. The research in this thesis aims for quality over speed. We introduce a ray-casting method that retains fidelity to the emission-absorption volume model, using a high accuracy integration approach. The integration formula handles piecewise linear emission and piecewise constant absorption, and correctly calculates the attenuation of light within each integration step, unlike many common methods.

For efficient rendering, a pointerless octree representation of the volume was used, with each ray performing a leaf by leaf traversal. Rays are able to traverse from leaf to leaf quickly without climbing the tree or performing a search, as in other methods.

We examine the quality of the renderer when used with a number of medical data sets, and discuss the advantages of a high quality rendering for image generation. We find that it is possible to generate high resolution images from reasonably sized data sets using the modest resources of current desktop personal computers. High order methods for integration prove to give greater accuracy for the same calculation time as standard methods.

Preface and Acknowledgements

Volume rendering is an interesting area of research in computer graphics. It is challenging because the phenomena being visualised are removed from our daily experience — we are unable to see through opaque objects. Such an ability is the stuff of fiction, yet it is being made reality by advances in technology. “Being made”, because while CT scanners and other such devices have existed for some years, interpreting and understanding the data they provide is still as much art as science.

I have chosen to study high quality rendering because most research in this area concentrates on speed at the expense of accuracy. A few recent papers reverse this trend, being primarily concerned with removing the simplifying approximations that have become standard elsewhere, and providing clear bounds on any errors that remain. I hope that the research presented here continues this new trend.

This thesis would not be possible without the help of other people. I would particularly like to thank my supervisor, Dr. Richard Lobb, for a great deal of constructive criticism that has improved my ideas immeasurably. I would also like to thank Peter Kulka for many discussions on the topic over the last year. Tania Scott has offered a lot of help and support in the course of my study, I would like to thank her especially for advice on the process of thesis writing and a great deal of food. Lastly, Denise Bateup has improved my knowledge of medical techologies with a number of discussions on radiology.

This document was typeset in 11pt Palatino using the LaTeX 2$_\varepsilon$ document formatting system.

Contents

List of Figures

Chapter 1

Introduction

A number of powerful scanning techniques which produce three dimensional data sets, or volumes, have become available in recent years. Volume visualisation is an important area of research, opening up new ways for scientists to see their data. It requires creativity as well as technical skill to generate tools able to convey meaningful information to users, as the task of visualising volume data is more complex than visualising two dimensional data in an image or even two dimensional surfaces suspended in space. The human visual system is mainly adapted to seeing surfaces — most of the objects encountered in daily life are opaque and we recognise their shapes only by their boundaries.

Volume scanning techniques are common in medical diagnosis. They have the advantage of being non-invasive, meaning that surgery is not required and the risks to the subject are low. This makes these scanning technologies very attractive to medical personnel. However, the data generated must be visualised accurately and effectively for such techniques to be viable — loss of information or inaccuracies in the final image (image artifacts) can have serious consequences.

Recent research in volume visualisation has increased the flexibility of visualisation tools, conveying more information to the end user. However, the large size of data sets has posed problems for volume storage and manipulation. Data sets have grown in pace with the increases in computer speed and technology, so efficient use of computing hardware is an ongoing concern.

1.1 Volume Data Sets

Modern sources of volume data include medical scanning techniques such as computer tomography, magnetic resonance imaging and ultrasound imaging. In many areas of industry, volume data is created by computer tomography for non-destructive testing of machine components or finite element simulations for obtaining stress, temperature or fluid flow data. Typically a scalar field is sampled on a regular rectilinear grid, although vector fields and irregular grids are encountered, for example in finite element simulations. The scalar quantity measured depends on the application, but for the sake of generality will be called *density* in this thesis and given the symbol ρ.

The term density covers a diverse range of quantities. In Computer Tomography (CT), for instance, the scalar field measured corresponds to X-ray absorption. X-rays are passed through the body and measured by a sensor which moves around the subject. The raw data from the device thus consists of a series of measurements of the line integral of X-ray absorption between the X-ray source and the sensor. These line integrals are transformed into a series of samples of absorption at points within the subject.

Magnetic Resonance Imaging (MRI) measures the presence of specific types of molecules by their interaction with radio frequency radiation under a strong magnetic field. Ultrasonic imaging uses the reflection of high frequency sound to map out tissues with varying impedances within the body.

Recently, more exotic techniques such as Positron Emission Tomography (PET) measure the radioactive decay of marker substances injected into the body, reconstructing samples on a grid in a similar fashion to CT scanning. An appropriate choice of marker substance can yield impressive information about metabolic functions such as brain activity.

In this thesis, we shall largely restrict our discussion to medical data, and concentrate on scalar functions sampled on a regular rectilinear grid. This subset of volume visualisation is a diverse area, including the visualisation of CT, MRI and PET data. Other types of data can often be satisfactorily represented by this type of volume: irregularly sampled volumes can often be resampled onto a grid, and scalar functions of vector-valued samples such as the magnitude function can often represent trends in vector data.

We shall use the term *voxel*, by analogy with *pixel*, to denote a density sample in space, and the term *cell* to mean the cuboidal region bounded by eight adjacent voxels.

1.2 Volume Rendering

Volume rendering is the process of translating volume data into an image. This translation requires the concept of an *optical model*, a theoretical justification of an algorithm that models the transmission of light through some medium. Optical models may be based on the physical processes involved in light transmission, in which case they are suitable for photorealistic imaging. However, not all optical models draw their justification from physics; many are used simply for their ability to convey useful information about volume data to the user.

To convey such information meaningfully, one must meet the user's expectations. An argument in favor of physically based models is that these expectations are based in experience of the real world, and unrealistic models are therefore misleading. However, one may note somewhat facetiously that if photorealistic physically based images are superior, photographs are cheaper, more accurate and more convenient than any volume scanning technique. X-ray images have had a huge impact in medical diagnosis despite their lack of photorealism, and their physical basis is far removed from that of human vision.

An optical model gives rise to a set of *optical properties* for each medium assumed to be present in the volume. Such properties usually involve colour, light absorption and reflectivity. The measure of light absorption used in this thesis is called *extinction* or *attenuation*, the terms being used synonymously. The extinction coefficient τ measures the fraction of light absorbed in a unit distance.

Volume rendering falls into two major categories, isosurface rendering and direct volume rendering. Isosurface rendering is a two step process transforming a volume into an intermediate representation, usually a set of polygons, which is then rendered using an ordinary surface renderer. The only optical model supported is that of a surface hanging in a transparent medium that reflects light according to a surface illumination model such as those of Phong or Lambert.

Direct volume rendering, as its name implies, relies on no intermediate representation. It generates images directly from the data, without using surface rendering techniques. Many varieties of optical model are supported, as is discussed in section 2.2.

1.2.1 Isosurfaces

The chief advantage of isosurface rendering is that surface renderers are common and well-understood, and in recent times are often implemented in hardware, so rendering is extremely fast. Real-time animated displays are possible on modest workstations.

The assumption behind isosurface rendering is that the interesting features of a volume can be visualised by displaying one or more isosurfaces from the data. An isosurface consists of the set of points $\{\mathbf{x} \mid \rho(\mathbf{x}) = k,\ \mathbf{x} \in \mathbb{R}^3\}$, where k is a user-defined density level of interest. This set of points is approximated by geometric primitives suitable for surface rendering, such as sets of polygons, and an image is generated. This assumption is not always valid, and the chief criticism of isosurface rendering is that all information within the volume that does not lie at the chosen level is simply discarded. A bad choice, or a noisy data set, yields inadequate results, often without giving clues about what choice of level would improve matters. In most isosurface renderers, only one level can be displayed in an image.

Simple methods of generating the polygonal representation include drawing cells containing the isosurface as cubes, so the polygonal representation consists of the union of all cube faces. This method suffers due to the extremely discontinuous nature of the surface generated. Unsurprisingly, images have a very blocky appearance. In another technique, contours can be constructed for each slice of the volume, a simple image processing problem, and then contours from adjacent slices can be joined by triangles to produce the required representation. While finding contours is easy, robust algorithms for joining them together are difficult to find. Most have problems when the topology of the isosurface changes between slices.

A very popular method for polygonising an isosurface, *Marching Cubes*, was advanced by Lorensen and Cline in [LC87]. In their algorithm, every cell containing the isosurface was analysed and, based on the topology of the isosurface within the cell, a number of triangles were generated. The method was to record an array of eight boolean variables denoting whether each vertex of the cell was above or below the density of the isosurface. This array, stored as a byte, was used to index a lookup table giving the number and location of a set of triangles able to approximate the isosurface. The vertices of each triangle were calculated by linearly interpolating density along the cell edges specified by the lookup table. While one would expect 256 cases were necessary in the lookup table, Lorensen and Cline used rotational and reflectional symmetries to reduce this number to 15. The set of triangles was then shaded and rendered using standard techniques.

As the surfaces thus produced are not smooth, due to the nature of linear interpolation, the shading function did not use the normals of the triangles. Instead, the normalised gradient of the density, $\frac{\nabla\rho}{|\nabla\rho|}$, was used. The density gradient was calculated using central differences (discussed in section 3.5.2). While this method in general does produce realistic images, given the limitations of isosurface rendering in general, it can be shown that the boolean array used is not sufficient information to infer the topology of the isosurface within a cell with certainty. Artifacts will sometimes occur. [vW94]

The last method we will mention generates isosurface images without the intermediate polygonal representation, making it a hybrid with direct volume rendering. Lin and Ching [LC96] have observed that under certain circumstances an isosurface can be rendered analytically, so their renderer does not suffer the effects of the approximation involved in polygonising a surface. They use trilinear interpolation (discussed in section 3.5) to reconstruct a density field from voxel samples. It can be shown that the density field is a piecewise cubic polynomial, and so finding the intersection of a ray with an isosurface reduces to finding the root of a cubic function. A closed form solution to this problem exists. They use this fact to write an isosurface extracting ray-tracer, which is accelerated with an octree representation of the data (octrees are discussed in section 2.5.3).

1.2.2 Direct volume rendering

Rendering images from the volume directly is more appropriate than isosurface rendering for many purposes. Isosurface rendering, while fast, discards much of the information present in the volume. Interesting and relevant features can thus be missed. Isosurface rendering is very dependent on the choice of density level to examine.

Direct volume rendering (DVR) uses the entire data set for its calculations without intermediate steps such as the computation of a polygonal representation. While a 2D image can never display all the detail of a complex 3D structure, DVR is a more robust approach to visualisation.

One key feature in this robustness is the ability to avoid binary choices in parameters. For instance, instead of choosing a single density value to inspect, we can choose a range with a smooth drop off. Noise is suppressed, and a less than optimal choice will still display relevant features, if less effectively.

Two generally accepted categories of DVR are *image order methods* and *object order methods*. In the first, images are generated pixel by pixel, with the complete calculation required to render a pixel being performed before moving on to the next. The most popular image order method is *ray casting*, where rays are cast through the volume and an optical model evaluated. Ray casting is accurate and flexible, but requires a great deal of computing resources. In an object order method, cells or voxels are processed in order, building up an image gradually. The most common technique is *splatting*, where each voxel is blurred and composited on the screen. Splatting is fast and can be implemented at interactive speeds using graphics hardware, but lacks accuracy.

Direct volume rendering is discussed in depth in the next chapter, where both techniques and various optimisations for them are surveyed.

1.3 Goals

This thesis will explore the high quality volume rendering of medical volume data. While the chief priority will be how to display relevant information to the user with as much accuracy as possible, program efficiency will be a secondary goal. A renderer that implements the ideas explored will be advanced, and its performance evaluated.

The remainder of this thesis will be as follows. The second chapter gives background information in the area of direct volume rendering, surveying major techniques and algorithm optimisations. The third chapter will describe the design of a ray casting renderer. The fourth chapter will evaluate the performance of the ray caster, comparing it with a standard ray caster. Lastly, we will gives the conclusions of the thesis, and outline future areas of research.

Throughout the discussion, a number of volumes will be used as examples. We shall describe them briefly here, a fuller discussion is deferred until the results in section 4.1. The first data set is a Gaussian sampled sphere of an opaque material at a resolution of $120 \times 120 \times 120$. The next data set is a $32 \times 32 \times 32$ sampled version of a function with the appearance of high frequency ripples. The third volume is a CT scan of a male pelvis, consisting of $256 \times 256 \times 111$ eight bit samples. The last was extracted from the Visible Woman archive. It is a $256 \times 256 \times 256$ volume of the subject's head and neck.

Chapter 2

Direct Volume Rendering

The best features of direct volume rendering (DVR) are that it can implement powerful optical models, and that it is possible to avoid binary decisions for viewing parameters. Levoy [Lev88, p. 136] states:

> The key improvement offered by [direct] volume rendering is that it creates a mechanism for displaying weak or fuzzy surfaces. This capability allows us to relax the requirement inherent when using geometric representations, that a surface be either present or absent at a given location. This in turn frees us from the necessity of making binary classification decisions.

We shall first discuss the material classification schemes allowed by DVR. We shall then explore in some depth the wide range of optical models used in DVR, then two major areas of DVR: image space and object space rendering methods. The main distinction between the two approaches is that in object space methods the algorithm iterates through cells or voxels adding a contribution from each to the image. In contrast, image space methods such as ray casting iterate through pixels in image space fully evaluating each pixel before moving on to the next.

The most important disadvantage to volume rendering is execution time. The final section of this chapter will discuss optimisation techniques described in the literature. Significant groups of optimisations are frequency domain methods, octree methods and wavelet methods. All have the feature that slowly-varying regions of the volume are evaluated swiftly. Another group, opacity methods, reduce the accuracy of calculation in regions of high optical depth as errors are attenuated by intervening material.

2.1 Material Classification

The basic task of material classification is to assign optical properties to regions of the data set. In medical visualisation, for instance, a certain range of densities corresponds to air, and should thus be made transparent, another may correspond to bone, another to tumours, and so forth. In general, some features or some range of densities will be "interesting" and will contribute most to the image, while the rest will be "uninteresting" and will be edited out as far as possible.

Material classification is thus a form of feature recognition, and is the main mechanism for summarising megabytes of data into information intelligible to humans. It is a complex process, as many data acquisition techniques produce noise and uncertainty, the properties of the scanner may change during scanning causing drift in density values, and of course many important features are complex in shape and subtly distinguished from surrounding tissue.

Obviously to do correct classification of materials in a human body would require sophisticated feature recognition, detailed knowledge of anatomy, and a wealth of experience with scanning technologies. This is clearly beyond the reach of the current state of computing, and volume rendering research adopts a simpler approach that leaves the complex and subtle aspects of the task to the human user. Novins [Nov93, p. 29] comments: "most scientists are interested in seeing their data in essentially 'raw' form, with a minimum of automatic interpretation."

Most classification techniques restrict the parameters to ignore shape and context, concentrating instead on density. These techniques reduce to designing a function from density to optical properties. The desirable characteristics of this function are that:

- It is robust with respect to noise.
- It must also be robust with respect to drift and other inaccuracies.
- It must convey the maximum amount of information to the user. Thus if bones are the subject of study, they must look like bones, and other tissues must not obscure the bone structure.
- Materials will be identified probabilistically, so yes or no decisions will be avoided in favour of "fuzzy" percentage assigning decisions.
- Transitions between materials will be continuous.

Novins [Nov93, pp. 24–29] describes three algorithms for generating classification functions, from simple to complex.

The first of these, *thresholding*, creates a piecewise constant function where the set of densities is partitioned into intervals, and each interval is given a single set of optical properties. We have a set of N materials with density ranges $\mathcal{M} = \{(\rho_1, \rho_2), (\rho_2, \rho_3), \ldots (\rho_{N-1}, \rho_N)\}$. To classify a sample of density ρ_s, we find a pair $(\rho_i, \rho_{i+1}) \in \mathcal{M}$ such that $\rho_i \leqslant \rho_s < \rho_{i+1}$. The sample is then given optical properties corresponding to that material.

This has obvious disadvantages, as the resulting function is not continuous and little provision is made for uncertainty in classification.

For real data sets, voxels do not represent point samples in space but measurements of weighted sums of regions around a point. The volume represented by a sample is thus greater than zero. It is possible, indeed common, that a voxel sample contains more than one material, but this classification scheme must simply choose the closest material to the measured density, leading to loss of information. Also, if the measured density for materials drifts or fluctuates over the course of scanning, it may be impossible to choose a classification scheme that works: layers of tissue will contain holes and lumps will appear mysteriously inside other tissues.

However, thresholding can have serious defects even if the data set is perfectly sampled in such a fashion that good classification parameters are possible. Many renderers, such as the one discussed in this thesis and the one advanced by Novins, perform classification of density samples rather than of the reconstructed density field. The usual reason for this is efficiency. If we assume this to be the case, the following problem is inevitable.

Consider a transition between one material and another. As will be seen, surface type illumination makes these transitions the most important regions for overall image appearance. The raw data, if it is well-sampled, will have a smooth transition taking several cell lengths to complete. This can be reconstructed with high accuracy without resorting to high-order interpolation methods. After thresholding, however, all voxels in one half of the transition will be classed as one material, the other half another. The transition loses its smoothness, occurring entirely within one cell length. This is effectively impossible to reconstruct accurately, leading to a very blocky appearance in images where each cell is delineated from adjacent ones.

The second scheme was first introduced by Levoy in [Lev88, p. 138]. It avoids the necessity to choose one single material type for a particular density, as it allows materials to be mixed smoothly. He assumes that, given a set of materials ordered from lowest density to highest, material transitions only occur between adjacent materials. So a single sample may contain at most two materials which are adjacent in our ordering.

Formally, we have a set of N materials that have densities $\rho_1, \rho_2, \ldots \rho_N$ such that $\rho_i < \rho_j$ if $i < j$. Levoy assumes [Lev88, p. 138] no material of density ρ_a touches any material of density ρ_b such that $|b - a| > 1$. A piecewise linear mapping is then constructed that converts densities to optical properties. A sample of density ρ_s between ρ_a and ρ_b is given optical properties that are a linear interpolation of material a and material b. Levoy says “This scheme ensures that thin regions of tissue will still appear in the image, even if only as faint wisps.” He also notes that when the adjacency criterion is not met, the classification will not be correct. In practice, this situation is rare with typical material distributions. For instance, CT scan subjects do not usually have bone meeting air or even skin.[1] Novins [Nov93, p. 26] claims “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.”

[1]Drebin *et al* [DCH88, p. 68] give an exception to this: “within nasal passages mixtures of air and bone are common.”

The third scheme is a generalisation of Levoy's. It is a probabilistic method that works by assigning to each density value a set of coefficients that each give the probability this density represents a particular material. Samples of that density are rendered using optical properties that are a mix of each material with non-zero probability. The scheme is reported by Drebin, Carpenter and Hanrahan [DCH88] three months after Levoy's paper.

With each material m_i is associated a probability distribution $P_i(\rho)$. This distribution represents the probability the scanner will output a sample of a given density for a voxel containing material m_i. Of course, such probability distributions are rarely known in advance and must be estimated.

Drebin *et al* estimate the proportion of each material i in a sample of density ρ as:

$$p_i(\rho) \quad = \quad \frac{P_i(\rho)}{\sum_{j=1}^{n} P_j(\rho)}$$

Optical properties for each material m_i are then mixed according the proportions p_i and the sample is classified.

When the sample is a scalar, as is most often the case in DVR, this scheme can be implemented by a simple look-up table. The way Drebin *et al* define their probability distributions $P_i(\rho)$ for this case is as follows. Each material has a range of 100% probability classification with a linear drop-off to 0% on either side. Densities that are not certainly classified as a particular material are a linear mix of the two adjacent materials. (See figure 2 on page 68 of [DCH88].) This scheme has the same defect as Levoy's scheme: samples that are a mix of non-adjacent materials will never be classified as such. As [DCH88, p. 68] comments: "In performing the musculoskeletal classification described above, voxels are never classified as being a mixture of air and bone since the soft-tissue distribution lies between the air and bone distributions."

Indeed, while this maximum-likelihood scheme is in theory a generalisation of Levoy's scheme, as it is implemented by Drebin *et al* it is equivalent to it. Any classification that can be represented in one scheme can also be represented by the other. This is reasonable, as specifying each material in the way they have, as a range of densities with a drop-off on either side, is a good user-interface decision. It is an intuitive way to specify materials and is indeed used by the renderer described in this thesis.

These three schemes do not exhaust the topic of material classification. As they all fall short of the abilities of an experienced person, sometimes manual classification is required, where trained personnel identify objects in the volume on a slice-by-slice basis. Zuiderveld *et al* [ZvOC$^+$96, p. 369] describe an example of the process, for a computer tomography angiogram of a woman's heart:

> This segmentation is labour-intensive and highly operator-dependent, which could result in loss of diagnostic information. In this case, the

> segmentation took 35 minutes and the calcifications in the vessel wall [the subject's medical problem] were not shown.

However, it is currently the only way to perform a classification that can explore the subtleties of human anatomy missed by automatic classification schemes.

2.2 Optical Models

The basic task of volume rendering is to simulate light as it passes through the data set to a synthetic camera. Various models are possible in DVR, varying in computational complexity by several orders of magnitude. Simple models are sometimes sufficient for scientific visualisation requirements, where being true to the physical process of light transmission is not as important as displaying useful information about the data set. Complex models, on the other hand, often simulate physical processes with greater accuracy and are best suited to modeling the physics of light transfer. One application where this is necessary is the simulation of clouds, where light may be reflected many times between light sources and the camera. An optical model should be chosen on the basis of the features the user is interested in viewing.

Many papers discuss how the authors have implemented an optical model, usually using some variant of Lambertian or Phong illuminated material with an extinction coefficient to model light absorption. This is evaluated by compositing samples onto pixels, approximating line integrals through the volume. We shall discuss several papers in brief to show optical models commonly used by researchers, then discuss a range of options in detail to show the flexibility of DVR.

Levoy in [Lev88] uses an optical model designed to display isosurfaces within volumes. Materials within the volume are illuminated using the halfway vector formulation of the standard Phong equation, with a linear depth cueing function. For the normal vector required by the Phong illumination function, Levoy uses the normalised density gradient, and does not describe how his program responds to a gradient of zero. Samples at the selected isosurface value are opaque, and the opacity is chosen in a manner designed to make visible contours a constant non-zero thickness. To achieve this, samples near the chosen value have an opacity inversely proportional to the magnitude of the local gradient.

Drebin, Carpenter and Hanrahan in [DCH88] use a slightly different approach. Regions with constant proportions of materials have a constant colour, and the isosurfaces where proportions change are shaded using a standard surface illumination model. They reference Phong,[2] Blinn [Bli82] and Cook[3] without mentioning which they actually use. Whichever illumination function they use, the optical

[2]Phong Bui-Tuong. Illumination for computer generated images. *Communications of the ACM,* 18(6):61–67, June 1975.

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

model cannot be represented exactly by line integrals of continuous functions. The optical model was a hybrid model which is effectively a set of surfaces corresponding to density isovalues, rendered using a standard surface model, suspended in layers of translucent, constant coloured mist.

Sobierajski and Kaufman [SK94] use a low-albedo approximation to a rigorous light transport model from Krüger,[4] where light is scattered according to a function "allowing only ideal diffuse and specular reflection." They do not give details about what function is used, but describe the advantages of giving the user a great deal of control over it to produce specific shading effects.

Upson and Keeler [UK88] use standard Lambertian shading using the density gradient for the normal, normalising the gradient vector similarly to Levoy. They claim their goal is to "represent abstract data sets of natural phenomena ... this work differs from others whose goals are to simulate the visual aspects of the phenomena realistically." [UK88, p. 62]. Their renderer shares with this thesis the desire to present interesting features of data sets rather than physically-based modeling of light transfer. Opacity is calculated by a transfer function of density.

A good survey of different illumination options is found in [Nov93, ch. 2], where several of the models described below are outlined: *maximum value rendering, absorption only rendering, low-albedo scattering* and *high-albedo scattering.* Images of the same scene, a CT scan of a male pelvis, rendered with the first three models are included for comparison. Images of a cardiac ultrasound data set contrast low and high albedo models.

The best rigorous analysis of volume optical models is found in [Max95]. Max surveys six methods from the reasonably simple through to the complex, deriving formulae from a particle density model of light transmission. He also gives images demonstrating the various effects made possible by each method.

We shall survey those algorithms, comparing their various features and computation costs. However, derivation from first principles will be omitted; the interested reader is referred to [Max95]. We shall assume for ease of explanation that the image is constructed by rays being cast through the volume from each pixel of the image. The same model can be implemented by a number of algorithms, so this assumption is without loss of generality.

2.2.1 Maximum value rendering

This is the simplest method for assigning a colour to a pixel. As a ray traverses the volume, the highest density value encountered is recorded. This value is then mapped to a colour and displayed on the screen.

This method is extremely simple, and can be evaluated in "several seconds" per image, according to Zuiderveld *et al* [ZvOC$^+$96, p. 367]. An advantage of this

[4]W Krüger. The application of transport theory to visualisation of 3D scalar fields. *Computers in Physics* 397–406, July 1991.

shading model for certain applications is that material classification can be omitted. The application under discussion in that paper was visualising Magnetic Resonance Angiography, a technique where volume data contain large amounts of noise. Methods requiring classification, such as isosurface methods or indeed this thesis, would perform poorly and maximum value rendering is said to be the method of choice [ZvOC+96, p. 367].

Naturally, such a simple method has severe limitations. Of all the cells encountered by a ray, the colour of the pixel is entirely determined by the density of one. The vast majority of information in the data set is thus discarded. One particular deficiency of this method is that it is impossible to tell what is inside a closed volume bounded by dense tissue.

Examples of diagnostic importance where this may occur include CT scans of the brain inside the skull, or searching for low-density cysts inside the liver or kidneys. It is also important when examining CT scans of tumours to determine whether they are cystic (therefore likely to be benign) or malignant; the former are filled with fluid and the latter usually solid.

As an aside, the researchers in [ZvOC+96] used an alternative rendering method at interactive speeds, as is discussed below in section 2.2.4 on emission-absorption rendering.

One notes that the necessity of interactive rendering speed is increased when most of the information in the data set is discarded. The users must view the volume from many angles, because many important features will only be visible from certain angles. One tumour will hide any other tumours in front of or behind it, for example. A user would need fewer images if the illumination algorithm puts more information into each image.

2.2.2 Absorption only rendering

A simple optical model that does not discard data in the fashion described above simulates the absorption of light in a material. The density field is mapped to an extinction field $\tau : \mathbb{R}^3 \to \mathbb{R}$, rays are cast through the field from a bright background to the eye and pixels coloured according to the accumulated opacity of the ray.

[Max95] derives an equation to integrate the absorption function along a ray. A similar equation, with a different derivation, was used by Blinn in [Bli82]. Let the ray be parametrised $\mathbf{r}(t) = \mathbf{d}t + \mathbf{v}$, where $\mathbf{v}$ is the viewpoint and $\mathbf{d}$ the direction. If $I(t)$ is the intensity of light at distance t, and I_0 is the intensity entering the region being integrated, then

$$I(t) \quad = \quad I_0 e^{-\int_0^t \tau(t)dt}$$

If the transfer function mapping density to absorption is the identity function, the result of this illumination algorithm is similar to a standard X-ray. Indeed, one can use a CT scan of a patient to simulate an X-ray image from any viewpoint. Novins notes that the "simulation of X-rays is one important application of this model". [Nov93, p. 13]

It is possible to accelerate the calculation of the integral using the Fourier Slice Projection Theorem and the fast Fourier transform. This technique is more fully described in section 2.5.2.

Illumination models that convey more visual cues, however, need to include scattering and emission effects as in the following models.

2.2.3 Emission only rendering

A similar optical model to absorption rendering models only the emission of light. The medium is modeled as emitting light according to a function $\varepsilon(t)$ of position, calculated by a transfer function of the volume density at that position. The example given by Max [Max95, p. 101] is that of an incandescent gas, where light is emitted but the gas is transparent, so absorption and scattering effects can be ignored. The function is slightly simpler than absorption rendering:

$$I(t) \quad = \quad I_0 + \int_0^t \varepsilon(t)dt$$

Malzbender in [Mal93] has demonstrated that this model can be evaluated efficiently by Fourier methods. This was extended with depth cueing and simple shading in the Fourier domain by [TL93] one month later, as is described in section 2.5.2.

2.2.4 Emission and absorption

A useful optical model considers both absorption and emission effects in a medium, generalising on both the previous models. Let the amount of extinction of light at a point be $\tau(t)$ as before, and the emission similarly be $\varepsilon(t)$. For the moment we shall consider $\varepsilon(t)$ to be an arbitrary function of location but in section 2.2.5 we shall refine it to incorporate shading and scattering effects. The extinction coefficient is again a transfer function of density. [Max95] derives the well-known formula:

$$I(t_0) \quad = \quad \int_0^{t_0} \varepsilon(t) e^{-\int_0^t \tau(s)ds} dt \quad + \quad I_0 e^{-\int_0^{t_0} \tau(s)ds}$$

This kind of model is now of great practical use in illuminating volumes, although the amount of calculation makes interactive renderers difficult. Novins [Nov93,

p. 10] lists its advantages: "Its main attractions are its physical basis, simplicity and its generality." We note that while its physical basis is an improvement on the models previously described, there are many physical aspects being ignored. The modeling of *solid* materials, which occur in many scenes in the literature, by non-interacting particles floating in space is questionable. Max [Max95] bases these models on gaseous phenomena: clouds, flames, incandescent gas. However, it has proved popular because of the realistic appearance of the results, regardless of its physical basis.

There has been much research in accelerating emission-absorption renderers using techniques such as wavelet transformations [Wes96], octree encoding and adaptive ray termination [Lev90] and DCT encoding [YL95], as is described in section 2.5.

The authors of [ZvOC$^+$96] were able to implement this optical model in an interactive system, although without shading. 3D texture mapping hardware was used to map density values directly to RGBA colours and composited on the screen at interactive rates. They report a very positive response from clinicians who felt the system to be "a glimpse of their future" [p. 370].

2.2.5 Low albedo scattering

The emission-absorption model can be extended by defining the emission function $\varepsilon(t)$ to include scattering effects from a light source. The major distinction between scattering models is how many reflections are allowed to occur between the light source and the eye, which in turn depends on the *albedo* of the particles in the transmission medium. Albedo is defined as the proportion of incident light reflected off a particle, and is often used in astronomy to characterise heavenly bodies.[5]

A simple way to model light scattering through a medium is to assume that the albedo of the medium is low enough that one need only consider one scattering event per ray. Light can travel from the light source until it reflects off a particle in the medium to the viewpoint. The possibility of multiple internal reflections is ignored.

A feature of this model is that regions where density is changing, for instance the boundary between one tissue and another, can be visualised similarly to a surface. Shadows cannot be modeled as the light is assumed not to interact with the volume between the light source and the point being visualised. Novins claims that this is not necessarily a bad feature for medical visualisation, as will be seen in the next section.

[Max95] gives a general rule for describing illumination models of this sort. The emission term in the rendering integral is the sum of a "non-directional internal glow" E similar to ambient light in surface rendering, and a shading rule S. So, if $\mathbf{X}$ is the point under consideration, $\boldsymbol{\omega}$ is the direction to the viewpoint and $\boldsymbol{\omega}'$ is the direction to the light source:

[5]Blinn, who introduced the term in 1982 [Bli82, p. 23], was modeling the rings of Saturn.

$$\begin{aligned} \text{The emission function} \quad \varepsilon(\mathbf{X},\boldsymbol{\omega}) &= E(\mathbf{X}) + S(\mathbf{X},\boldsymbol{\omega}) \\ S(\mathbf{X},\boldsymbol{\omega}) &= r(\mathbf{X},\boldsymbol{\omega},\boldsymbol{\omega}')\; i(\mathbf{X},\boldsymbol{\omega}') \end{aligned} \tag{2.1}$$

where $i(\mathbf{X},\boldsymbol{\omega}')$ is the incoming light and $r(\mathbf{X},\boldsymbol{\omega},\boldsymbol{\omega}')$ the "bidirectional reflection distribution function" that specifies in what directions light is reflected by a particle. Further, rigorous models of light scattered by a density of particles follow the rule:

$$r(\mathbf{X},\boldsymbol{\omega},\boldsymbol{\omega}') \quad = \quad a(\mathbf{X})\; \tau(\mathbf{X})\; p(\boldsymbol{\omega},\boldsymbol{\omega}')$$

where $\tau(\mathbf{X})$ is the proportion of light extinguished by the particle, $a(\mathbf{X})$ is the albedo, or the proportion of $\tau(\mathbf{X})$ that represents scattering not absorption. Finally, $p(\boldsymbol{\omega},\boldsymbol{\omega}')$ is the phase function representing the proportion of light scattered in a particular direction.

Max then quotes from various sources suitable phase functions for modeling firstly particles of similar size to the wavelength of light, and then particles greater than a wavelength of light. The result is a calculation intensive physically-based model suitable for photo-realistic simulation of gaseous phenomena. The example images in the paper are of cloud simulations, a particularly demanding application as shown by the lack of realism in many computer generated cloud scenes. However, this physically-based model does not necessarily improve the quality of visualisation for medical purposes. The paper provides a simpler model that simulates diffuse illumination, an adaption of Lambertian illumination:

$$r(\mathbf{X},\boldsymbol{\omega},\boldsymbol{\omega}') \quad = \quad \max(\nabla f \cdot \boldsymbol{\omega}', 0)$$

This is a more promising illumination function, as it provides shading cues with only a modest amount of calculation. However, experiments show that the results are not always what one would expect. This will be explored further in section 3.6, where the illumination model for our renderer is described.

The reader will notice that the gradient term in the above equation is unnormalised. In surface shading, one normalises the normal of a polygon but some authors in volume rendering literature [Max95][DCH88] introduce the concept of the "strength" of a surface. Weak surfaces should be shaded less brightly than strong surfaces, to emphasize actual material changes instead of noise, and to allow slow material changes to be represented softly. [Lev88] uses the same concept to modulate opacity, providing constant thickness to visible layers within the volume, as has been described above.

2.2.6 High albedo scattering

In high albedo media, the probability of multiple internal reflections cannot be ignored. Light may interact with particles a large number of times between the light source and the camera.

One simple way to move from a low-albedo model to a high one is to include self-shadowing. This is similar to the previous section, except that the $i(\mathbf{X}, \boldsymbol{\omega'})$ term in equation (2.1) is not a simple evaluation of the strength of the light-source, but a ray-trace from point $\mathbf{X}$ to the light source, with a full line integral being evaluated.

It would seem that this would require $O(n^2)$ operations per ray, but an $O(n)$ algorithm is possible by using a pre-pass to calculate the attenuation of the light-source at each point in the volume, an operation that takes the same length of time as the final rendering. [Max95, p. 104]

Is this a useful technique for medical visualisation? [Nov93] implemented self-shadowing in this manner, and found that important information was actually obscured by this added realism. He says [p. 18]:

> 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 data set. 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”.

He states that, apart from computational complexity, the “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” [Nov93, p. 17]. Thus it would seem the added calculation involved would be wasted for our purposes.

[Max95] describes in detail methods for modeling multiple internal reflections, some of which are related to surface radiosity algorithms, while others use Monte Carlo techniques. Algorithms of order $O(n^6)$ and $O(n^7)$ are described, where n is the sidelength of the volume. While the added realism is useful for his purpose of rendering realistic clouds, he says these methods “are expensive in computer time and are overkill for most scientific visualisation applications.” [Max95, p. 105]

2.2.7 Discussion

The preceding discussion covers a range of optical models with a large variation in realism, and a computational complexity varying by several orders of magnitude. Within this range there appears to be a balance: some realism is required to convey useful information by visual cues, but added realism wastes computer time or, even more seriously, degrades the useful information content.

The first methods discussed, maximum value, absorption only and emission only, can be implemented in near real-time on modern desktop computers. However, they do not convey sufficient information to be appropriate for a high quality visualisation system. Emission-absorption rendering as in [ZvOC$^+$96] is a vast improvement, but the lack of shading hinders comprehension of the image.

The high albedo methods are too complex and do not increase information content. Multiple scattering effects and shadows therefore offer little to this thesis. Even physically-based modeling of particle densities is too much calculation for too little return.

Low-albedo emission-absorption rendering seems to be the most promising option for medical visualisation. It can provide realistic shading cues, and can simulate simpler models such as isosurface methods accurately if the user desires.

2.3 Image Order Methods

Image order methods perform calculations for each pixel successively, finding the final colour of one before moving to the next. A very popular image order method is ray casting. In ray casting, a single ray is cast for every pixel and a line integral along it evaluates the optical model.

As can be imagined, a complicated line integral for every pixel in an image is a large computation cost, and the bulk of research in this area has focused on speeding the process. This is more fully discussed in section 2.5.

An early example of ray casting is [Lev88], where Levoy describes a ray caster designed to display isosurfaces in volume data without the errors associated with fitting geometric primitives to the isosurface.

Another early example is provided by Sabella [Sab88], who demonstrates the flexibility of ray casting by mapping a number of features within the data to each pixel's colour in a single image. He combined attenuation along the ray, the maximum value encountered by the ray and depth cueing all into one colour using a mapping to the HSV colour space. The value component represents a measure of attenuation, similar to the normal rendering integral, the hue component represents the maximum density encountered by the ray, and the saturation component gives depth cueing. The result is that "hot spots" within the data stand out with a red hue, and distant features having low saturation appear to be seen through a fog. [Sab88, p. 54]

Later research includes [Lev90], which as will be seen in section 2.5.3 accelerates ray casting with an octree, and Sobierajski and Kaufman [SK94] who combine surface and volume data to create a hybrid ray tracer.

2.4 Object Order Methods

Object order methods traverse the volume instead of the image, building up pictures voxel by voxel. We shall discuss V-Buffer rendering, splatting, and HIACS cell projection as examples of this type of algorithm. A survey of the first two of these methods is given by Kulka [Kul98], to which the next two subsections are indebted.

2.4.1 V-Buffer

The V-Buffer algorithm by Upson and Keeler [UK88] was introduced in 1988. In this method, every cell in the volume was examined in a front to back order, which is trivial to evaluate as cells lie on a regular grid. Each cell was shaded and trilinear interpolation coefficients were evaluated for the colour parameters. Rays were then cast through pixels, and the segments intersecting the cell were integrated to yield RGBA values which were composited onto a buffer. When all cells had been processed, the buffer contains the final image.

This process performs the line integral of each ray through the volume with great accuracy, and is indeed functionally equivalent to ray casting, with the improvement that the image is built up in a progressively refined manner "visually akin to what one might experience watching a fog-shrouded hillside as the fog recedes and ever more detail of the landscape is revealed." [UK88, p. 61] However, the authors note that "the ray caster is generally more efficient for conventional machine architectures and opaque data volumes" [p. 61] as it can implement early termination for opaque rays, but the V-Buffer method cannot. However, "the cell-by-cell method can reach the same efficiency when the object is quite transparent." [p. 61]

2.4.2 Splatting

Splatting, introduced by Westover the following year in [Wes89], is a fast technique that can render at interactive speeds. Unfortunately, at least in its original form it lacked accuracy.

In this method, each voxel is examined in either a front-to-back or back-to-front manner, and a blurred version is composited on the screen. The metaphor giving the method its name is that of snow balls being thrown against a wall, where each snowball is a sample convolved with a reconstruction filter. [Wes89] describes an orthographic projection renderer, although the method was later extended to include perspective by Westover and Swan *et al.* [Wes90][SMM$^+$97]

The motivation for the method can be described by considering ray casting a single sample in space. In ray casting, the discrete data are reconstructed to produce a density field by convolution with a (usually trilinear) filter, and then rays are integrated to find pixel intensities. In our single sample example, then, the point is filtered to yield a fuzzy ball in space, which is then converted to a fuzzy circle in the image by evaluating a grid of parallel line integrals. Westover notes [Wes89, p. 12] that the intensity of the sample is a constant that can be factored out of the integrals. In other words the final image is always the same, merely scaled in brightness by the value of the sample.

If we use a radially symmetric reconstruction filter, for instance a radial Gaussian, that final image will also be the same for any orthographic view at any angle. Thus the *footprint* (set of coefficients for generating the image) is constant. As it is constant, it can be precalculated once and stored in a table. This makes the use of high order reconstruction filters possible, in contrast to ray tracing where even trilinear interpolation often dominates rendering time.

Realistic scenes have more than one visible sample. To render more interesting volumes, each voxel is "splatted" onto the screen in order, gradually building up a complete image. If the extents of the filters never overlap, and the attenuation of the splat by itself and illumination by a light source are ignored, this method is reasonably accurate.

However, in practice the extents of interpolation filters always overlap because practical Gaussian filters are several cell lengths wide. Thus all voxel splats will overlap their neighbours. The method assumes that one splat can be composited over the top of another regardless of whether their filtered samples overlapped in space, so the entire brightness of the further voxel will be attenuated by the entire opacity of the closer. In an accurate method, the overlapping region of the further voxel lies in front of part of the nearer one, and so will not be as attenuated.

This inaccuracy in splatting motivates the use of Gaussian filters for generating footprints, as the smoothing effect of the filter makes artifacts less noticeable. The usual effect of splatting without sufficient smoothing is that surfaces tend to vary in brightness between splats. As [SMM$^+$97, p. 198] notes, in animations of volumes this problem is most pronounced when the order of traversal changes. As a volume rotates, for instance, a sudden jump occurs when it makes an angle of $\frac{\pi}{4}$ radians with the eye.

The second major inaccuracy in splatting relates to shading and classification. In an ideal implementation, reconstruction occurs first, then classification and finally shading. For ray casting, this order is often juggled for efficiency reasons, but in splatting it is inherent in the method that reconstruction occurs last, as [SMM$^+$97, p. 198] comments.

However, as splats can be composited by texture mapping hardware in modern workstations, splatting is by far the fastest DVR method, and fast animations can be generated in real time. Splatting has been extended to include perspective projections by [Wes90] and [SMM$^+$97], but at some cost to either speed or accuracy: the footprint of a splat needs to be recalculated for every voxel to give an accurate integral of the reconstruction filter. [Wes90] approximates the answer by fitting elliptical splats to voxels [p. 375], and [SMM$^+$97] varies the size of splats depending on the depth of the voxel to ensure a uniform sampling rate.

Laur and Hanrahan [LH91] also use variable sized splats. They accelerate the algorithm by means of an octree structure and splat leaves instead of voxels. This will be more fully discussed in section 2.5.3.

2.4.3 HIACS cell projection

The last object space method we shall survey, advanced by Williams, Max and Stein [WMS98], is distinguished by the exceptional effort expended on image quality: the name HIACS is an abbreviation for HIgh ACcuracy. It is broadly similar to V-Buffer rendering, in that each cell is examined in a back-to-front order, and rays are cast through it for each pixel it projects onto.

However, cells do not need to be on a regular grid: tetrahedral, brick, prism or pyramid shapes are supported provided an acyclic visibility ordered graph can be found. Either orthographic or perspective projections can be used, and Phong illuminated isosurfaces can be included. For cells so small they fall between pixels, sub-pixel splatting is used to correct for sampling error. Integration techniques are used where known, otherwise high accuracy numerical methods are employed. The cell mesh can be nonconvex or even disconnected.

The renderer uses the emission-absorption optical model, and piecewise linear classification similar to Levoy's method described on page 9 of this thesis. The basic algorithm is follows.

- Firstly, cells are sorted to give a back-to-front order.
- For each cell, rays are cast through it corresponding to each pixel it projects onto.
- The cell is cut into slabs, within which material properties vary linearly. The boundaries of each slab have a density between adjacent linear intervals in the classification scheme.
- Accurate integration is applied to each ray segment within each slab.
- The resulting values are composited to generate the image.

Compromises in accuracy were not introduced to speed rendering, as their defined goal was "to design a volume rendering system to create benchmark images for use as a standard of comparison" [WMS98, p. 37], so that other systems using simplifying approximations may be compared reliably. The renderer is designed to display finite element simulations.

2.5 Optimisation Methods

The large computation time required by most rendering methods, and the continually increasing sizes of data sets, have led many researchers to focus on optimisation methods to produce images in a reasonable time. The usual goal is to approximate regions which contribute little to the final image, such as dim regions or regions lying behind opaque material. Homogeneous regions can be approximated by a constant average value. Such regions are sampled less often, by increasing the step size of a ray caster, or by increasing the splat size of a splatter.

We shall discuss a range of methods. Firstly, we shall describe methods that decrease calculation behind opaque regions, then frequency domain methods, octree and pyramid methods, and finally wavelet methods.

2.5.1 Opacity methods

These optimisations are generally relevant to ray casters, where a front-to-back traversal yields optical depth information during ray integration. Samples at a large optical depth have most of their intensity absorbed by the intervening material, and accurate calculations add little to the quality of the image.

The simplest method, used by Levoy [Lev90], is to terminate the calculation when the accumulated opacity of the ray exceeds a threshold. The value used by Levoy was 0.95, which does not affect the image noticeably. The same method was used by Upson and Keeler [UK88] for their ray tracer, except that the authors terminate the ray calculation when the opacity reaches unity. Levoy claims the speed up achieved by this method is between 1.3 and 2.2 [p. 254], while [UK88] does not give quantative results.

This method introduces a systematic bias into the image, since terminated rays will never be quite as bright as they should be. This is noted by Danskin and Hanrahan [DH92], who introduce two alternatives, which they call *Russian Roulette* and *β-acceleration*. Russian Roulette involves probabilistically terminating rays depending on their optical depth, and proportionally increasing the energy of all other rays so that the bias to dim images is eliminated. This yields a speed increase of 10% without visible artifacts; apparently lowering the threshold introduces noise faster than it increases speed [DH92, p. 96].

β-acceleration involves gradually increasing the step size taken by the ray tracer, depending on the opacity of the ray. Danskin and Hanrahan implement this as follows. Their renderer already evaluates an error metric as is discussed on p. 26 of this thesis, which is tested against a threshold to determine the level of approximation used. As the contribution of the error in a sample to the final image depends directly on how much the sample is hidden by opaque material, they multiply their error metric by the ray's accumulated opacity, a very simple change to make to the code. The increase in speed is about 10% [DH92, p. 96]. The authors note "Clearly, given a *homogeneity-acceleration* implementation, *β-acceleration* is worth the extra eight keystrokes it takes to implement." [DH92, p. 96] As an aside, they actually use nine keystrokes not including whitespace.

2.5.2 Frequency domain methods

The motivation behind frequency analysis of volume data for optimising rendering is the Fourier Projection Slice Theorem. As Totsuka and Levoy in [TL93, p. 271] explain, this theorem applied to a 3D volume states:

> The following two are a Fourier pair:
>
> - The 2D image obtained by taking line integrals of the volume along rays perpendicular to the image plane.
> - The 2D spectrum obtained by extracting a slice from the Fourier transform of the volume along a plane which includes the origin and parallel to the image plane.

In other words, to create an orthographic image using an optical model requiring only a line integral of a single quantity, we can obtain the Fourier transform of the volume, extract a slice from it, and find the inverse transform (in 2D) of the slice to get the image.

Malzbender in [Mal93] use this fact to write an emission-only volume renderer, and claims an improvement in the order of complexity over standard methods. He claims that both ray casting and splatting are "in some sense . . . $O(N^3)$ for an $N \times N \times N$ data array, since each sample point should be visited" [Mal93, p. 233] and that other optimisations do not improve the order of complexity [p. 234]. While this is something of a generalisation, ignoring factors such as image size, it is the case that his renderer has order $O(N^2 \log N)$.

A complication in Fourier rendering is that reconstructing the slice must be done with a great deal of accuracy, and even though this is an $O(N^2)$ operation it dominates rendering time. Malzbender uses sophisticated reconstruction filters with extent $5 \times 5 \times 5$ and $3 \times 3 \times 3$ to get an adequately sampled image spectrum.

Malzbender's technique evaluates the equation

$$Intensity = \int_{-\infty}^{\infty} \rho(t)dt \tag{2.2}$$

where t parametrises a ray perpendicular to the image plane. This is sufficient for either the emission only or absorption only models described above, but does not include shading or occlusion effects. The result is similar to a standard X-ray image.

[Mal93] claims an increase in speed of up to two orders of magnitude for practical data sets [pp. 244–247].

Totsuka and Levoy [TL93] extend Malzbender's method to include depth cueing and simple diffuse shading, both in the Fourier domain. They use two properties of Fourier transforms [TL93, p. 272]:

- Multiplication by a linear ramp in the spatial domain is equivalent to differentiation in the Fourier domain. Thus linear depth cueing can be implemented by differentiating the spectrum analysis.
- Differentiation in the spatial domain is equivalent to multiplication by a linear ramp in the Fourier domain. This allows simple diffuse shading to be implemented by multiplying the spectrum.

We note that the shading they use is only an approximation to Lambertian shading. [TL93] improves reconstruction speed by adaptively using a $1 \times 1 \times 1$ filter when frequency components are nearly zero [pp. 275–276]. The renderer runs about an order of magnitude faster than a spatial domain renderer for data sets of practical size. However, while these renderers are very fast, they can only evaluate limited optical models. Equation 2.2 is used to evaluate an emission-only model, and one can see that with minor alterations it could evaluate an absorption-only model, but combining the two models is impossible. All manipulations of the volume, such as the shading and depth cueing above, must be conducted in Fourier space, making the usual illumination functions difficult to implement.

The last frequency based method we shall examine is discussed by Yeo and Liu [YL95]. It uses the Discrete Cosine Transform (the compression technique used in JPEG image compression) to compress the volume in memory. The main goal of the technique is to fit larger data sets into RAM, but they report an increase in speed over standard ray tracing even aside from eliminating page faults and thrashing. The compression method segments the volume into overlapping 32^3 blocks, of which only one needs to be decompressed for rendering at a time. The data set can be compressed by a factor of 20 to 30 before artifacts occur.

The rendering speed was measured twice, once on a machine with sufficient memory (256Mb) for standard ray tracing, and once on a machine that could not hold the entire compressed data set at once (64Mb). The first measurement shows a speed up of about 30%, the second over 300%. The improvement in speed even when memory is not an issue was attributed to quicker disk access, better cache utilisation and fast traversal of homogeneous regions (where most DCT coefficients are zero).

2.5.3 Octree and pyramid methods

The next major type of DVR acceleration we shall discuss makes use of octrees to exploit spatial coherency in the data set. In splatting, nodes of the octree can be splatted instead of individual cells, so that regions that do not contribute much to the appearance of the image will be processed faster. In ray casting, the traversal of the octree can be altered to stay in coarser levels of the tree in unimportant regions.

We shall first discuss the uses octrees have been put to in DVR literature, citing representative papers for examples. We shall then discuss octree representations used in computer graphics, and examine their memory requirements and performance. These are particularly significant factors in a ray casting volume renderer, as unlike surface ray tracers the number of nodes in the octree can easily reach several million and memory requirements can become prohibitive.

We shall devote greater space to this discussion to provide background for the implementation of the renderer this thesis describes. A discussion of traversal methods for casting rays through octrees is deferred until the design of this renderer is discussed in section 3.4.

In this discussion, we shall carefully distinguish a number of terms. The term *voxel* refers to a single density sample in space, and a *cell* is the cuboidal region surrounded by eight adjacent voxels. A *node* is an element of the octree, containing a cuboidal set of cells, and a *leaf* is a node which is not subdivided into child nodes. This may be because all the cells it contains have identical densities, or because it contains a single cell with variable density. The top level of the tree is the root node containing the whole volume, the leaves are at the bottom of the tree.

Octree acceleration methods.

An early example of the use of octrees in volume rendering is given by Levoy [Lev90]. He used an octree to store a binary value representing the presence of opaque material. A one denotes the presence of nonzero cells that contribute to the image, a zero indicates only transparent cells. The binary values were stored in a volume pyramid, as is described below in section 2.5.3.

The renderer traversed the tree by searching down it until either a zero was found, in which case that subtree could be ignored, or the base level was reached, in which case samples were taken and composited onto the ray. The renderer then continued the traversal until it had exhausted the volume, or until the ray became effectively opaque.

Levoy's octree traversal was found to be a significant cost, and so was optimised so that it never rose to within two levels of the root node, as in interesting data sets zeros are rare that high up the tree. His traversal also never went lower than two away from the bottom of the tree, as in that situation a simple cell-by-cell traversal was faster. The overhead for climbing and descending the tree searching for zeroes was thus minimised.

Levoy reports that the use of octrees in this manner increased the speed of rendering by between 200% and 500% [Lev90, p. 254]. His three data sets included two medical CT scans and an electron density map of a ribonuclease molecule.

Laur and Hanrahan in [LH91] have a very different approach to the use of octrees. They implemented an optimised splatting renderer that again uses a volume pyramid, but instead of storing a binary value they store a measurement of the error associated with splatting that node. To render an image, the tree is traversed and when a node contains a smaller error than is deemed acceptable to the user, that node is splatted and its children are ignored. The error metric is a root mean square function comparing the average density of the node with the densities of its constituent cells.

This optimisation relies on the ability to efficiently splat nodes of different sizes. They make use of polygon rendering hardware by using splat footprints made out of several translucent polygons instead of the more common texture maps. They say "Recent [as of 1991] workstations have added the ability to interpolate α along with color, and to provide hardware assist for compositing" [LH91, p. 286]. Splatting footprints can be scaled very simply (apart from an opacity correction) and the complexity of scan converting different sized polygons passed on to graphics hardware. This approach is less common now as current workstations also provide texture mapping assistance, so footprints can be texture mapped onto a single polygon and still have the speed advantages of hardware.

With this hardware assisted adaptive error method, the quality of the image is made a function of the desired rendering time. A good preview of complex data sets can be achieved in five seconds [LH91, p. 288], although the authors do not mention the specific hardware used, apart from the fact that it was from Silicon Graphics. [p. 288]

In 1992, Danskin and Hanrahan [DH92], expanding on Levoy's work, used pyramid datastructures to implement a number of optimisations. They derived a formula for importance sampling: minimising error by sampling at a rate depending on the contribution to the final image. They found that the error in a sample depends on the presence of bright material, the accumulated opacity of the ray thus far, and the homogeneity of the volume surrounding it.

To accelerate ray casting within a certain allowable error, Danskin and Hanrahan's ray caster traversed nodes of the pyramid structure at a height depending on the intensity, accumulated opacity and homogeneity of the node's constituent cells. Pyramids were constructed containing three parameters of the density distribution of the constituent cells: average, maximum and range. The average density was used for rendering, the maximum value used to measure intensity and the range value used to measure homogeneity. Also, as was discussed earlier, as the opacity β accumulated on a ray increased, so did the traversal level in the pyramid.

The results indicate that the most significant factor for time efficiency was homogeneity, allowing a 5% error in a homogeneity-accelerated image resulted in a 300% increase in speed. Presence acceleration ran only 200% faster with the same level of error, while the β (opacity) acceleration (implemented within the homogeneity-accelerated renderer) as discussed above improved speed by another 10% [DH92, p. 96].

A survey of octree datastructures.

As will be shown, the overheads involved in storing an octree in memory can be far greater than the size of the original data set. This is because, unlike most surface renderers, volume octrees of practical size have millions of nodes, each storing perhaps only one density sample, which we assume to be a byte in size. We shall also assume a 32-bit machine, so a pointer is four bytes long.

Also, the speed of traversal supported by the datastructure is discussed, i.e. whether searching or tree climbing is required.

Our discussion will cover four diverse implementations that have been used for volume or surface ray tracing, adapted to the requirements of the renderer. These are:

- A naive pointer-based implementation,
- The hashtable based structure used by Glassner [Gla84] as a space subdivision scheme for surface ray-tracing,
- The sorted linear notation reported in Foley *et al* [FvDFH90, pp. 554–555],
- A pyramid of volumes similar to [Lev90], [DH92] and [LH91].

As the purpose of the octree in the original paper is often different from what is required here, the evaluation of the datastructure may differ from the original author's intent.

All the implementations here have a memory cost proportional to the number of leaves in the octree (denoted by n). Some also have a significant constant memory overhead. For reference, the $256 \times 256 \times 111$ pelvis CT scan mentioned in the introduction (page 5 has 1 297 313 leaves after classification.[6] The 256^3 head data set extracted from the Visible Woman archive also mentioned contains 4 814 886 leaves after classification into only two materials, air and flesh. The worst case for a 256^3 volume is 255^3 or 16 581 375 leaves. We shall evaluate the implementations in terms of the Visible Woman dataset and the worst case 256^3 volume.

Naive pointer implementation

This is the simplest, most intuitive and most versatile implementation for the octree data structure but all the authors surveyed have avoided it, perhaps often due to the large memory costs associated with pointer storage. Its structure is as follows: an internal node is an array of eight pointers to nodes, and a leaf node is a byte density value.

[6]The tissue was classified into four tissue types: air, skin / fat, muscle and bone.

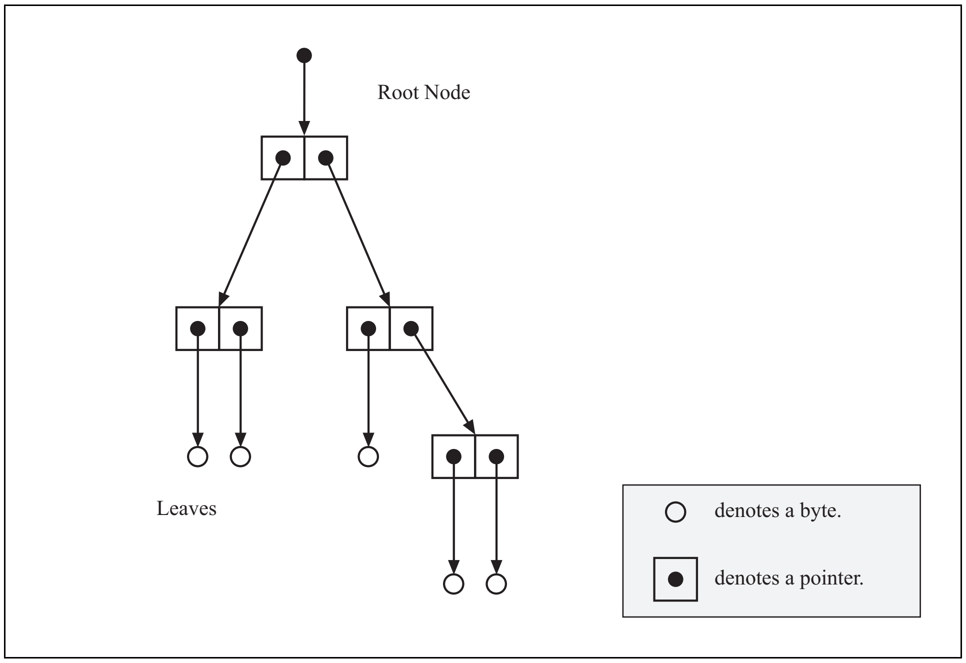

FIGURE 2.1: A naive tree implementation

See figure 2.1 for a simplified binary tree diagram of this structure. The memory cost of this implementation is the sum of that of the leaves themselves, the pointers to the leaves in the parent nodes, the pointers to the parent nodes, the pointers to the grandparent nodes, and so on up to the root node.

Let us first consider only the pointers in what we will assume for simplicity to be a balanced tree. The number of pointers can be closely approximated by a geometric series as for each level up the tree there are $\frac{1}{8}$ as many pointers as the one below. We shall assume for simplicity that this series is infinite, but in practice it stops at the root node of the tree. This approximation gives an error of $\frac{1}{7}$ of a pointer. Since n, the number of leaves in a tree, is only rarely lower than several hundred thousand in practical situations, this error is minute. We have:

$$\begin{aligned}\text{Number of pointers} &= n + \frac{n}{8} + \frac{n}{8^2} + \frac{n}{8^3} \cdots \\ &= \frac{n}{1 - \frac{1}{8}} \\ &= \frac{8}{7} n\end{aligned}$$

This is, by the same logic, the number of nodes in the octree. The total memory cost in bytes is the sum of the original data (one byte per leaf) and the pointers (assumed to be four bytes per pointer). Thus:

$$\begin{aligned}\text{Total cost} &= n + \frac{4 \times 8}{7} n \\ &= \frac{39}{7} n\end{aligned}$$

We have a cost of just over 5.5 bytes per leaf. Considering the volumes described on the previous page, this would require over 25.5Mb for the Visible Woman data set, and over 88Mb for a worst case 16Mb data set (one without coherency). This is clearly prohibitive.

The time to extract a node from this implementation is not too high, for every level between the root and the leaf one must follow a single pointer, and find the index of the child to examine using some simple integer arithmetic. To traverse the tree, though, a walk up to an ancestor node and down to the next is required. As will be seen in section 3.4.2, it is possible to traverse from one leaf to the next in constant time.

Glassner's hash-table implementation

As Whang *et al* [WSC$^+$95] notes, Glassner in [Gla84] was the first to use octrees to accelerate surface ray tracing. The octree adaptively subdivided space until the number of surfaces in each leaf is below a specified limit and as each ray passes through each leaf only that many surfaces need be tested for ray intersection. Thus the number of leaves in his octree is small compared to volume data sets, but the amount to be stored at each leaf is large — a list of surfaces instead of a single byte.

Glassner's method is to give each node a systematic name that encodes both size and position information. The name is then used as a key for storing the node in a hashtable, giving a versatile trade-off between access speed and memory usage. When more than one node is hashed to the same location, they are stored in a linked list. The name is constructed as a string of base 10 digits, with each digit representing the index of the child node and the whole string giving the complete path through the tree.

Glassner used the digits 1–8 to index the child nodes, and the digit 0 as a placeholder, making the digit 9 redundant. This increases the readability of the name at some slight cost to storage space. The name can be stored as an integer, and manipulated with simple integer arithmetic.

A 32-bit integer is sufficient for naming up to a 512^3 volume with two bits spare. We need to know whether the node is subdivided or not; this could be stored in one of the surplus bits. With the name we must store a pointer to the next node in the list, and of course the value of the node if it is a leaf. We have:

```
struct node {
        int     name;   // Including subdivision flag
        node*   next;
        byte    density;
};
```

This amounts to fully nine bytes per node. For each leaf, we must also count the cost of the parent nodes and by the same logic as before find there are $\frac{8}{7}$ as many nodes as leaves. This leads to a cost per leaf of over ten bytes.

This substantially worse than the naive implementation even before we include the size of the hashtable. We note that a linked list is not the most efficient way to store a list of values, and we can optimise the number of `next` pointers somewhat. Even if we assume this could be optimised to be negligible we do not gain anything over the naive implementation. If we ignore the cost of the hashtable, the total cost is 26Mb for the Visible Woman data set and over 90Mb for the worst case.

Access time is reasonable, as we must perform a hashtable look-up followed by searching a linked list, and assuming the hashtable size is small compared to the rest of the datastructure[7] this means examining several nodes on average. To find a given leaf we may follow half a dozen pointers.

Foley and van Dam's linear octree notation

The representation described in Foley *et al* [FvDFH90, p. 554] names nodes in a similar fashion to Glassner's method, except that the placeholder character is at the end of the digit string, not the beginning. They use an actual string of 4-bit characters instead of an integer representation, but there is no reason why Glassner's integer representation cannot be used.

Again, only the leaf nodes are stored, but they are stored in a sorted array instead of a hash table. Leaves can be accessed by any fast searching algorithm. If we wish to find the value of a particular cell, we calculate its name and search the array for the name. If it exists, that cell happens to be a leaf and we are done. If not, we extract the smallest leaf greater than the name we searched for (if the placeholder character is defined as greater than any other digit). This will be the leaf containing that voxel, no matter which level of the octree contains the leaf.

The memory cost for this is unfortunately still prohibitive, although an improvement over the naive scheme. For each leaf, we need to store one integer for the name and one byte for the value. No internal nodes need be stored. The cost is 5 bytes per leaf, or nearly 23Mb for the Visible Woman data set and 79Mb worst case.

The access time depends on the quality of the searching algorithm and the distribution of leaves in the array. A simple binary search will find any given node in $\log_2 n$ time, or 24 iterations for the Visible Woman data set. However, faster searches have been devised that make assumptions about the distribution of data. An interpolation search on an evenly distributed data set should be faster for the first few iterations[8]. A hybrid search algorithm would be appropriate.

A volume pyramid

Pyramids of volumes, where the data set is represented as a set of volumes with each volume one eighth the size of the one before, have been used in volume rendering by

[7]As we must, the memory resources are already prohibitive.

[8]An interpolation search works best when the data distribution is flat, searching in $\log(\log n)$ time, but in the worst case it searches in linear time. See [Man89, pp. 125–127]

Levoy [Lev90], Danskin & Hanrahan [DH92] and Laur & Hanrahan [LH91]. They are similar to the MIP maps used by Williams (quoted in [FvDFH90, p. 826]) for texture mapping, except for the extra dimension. In their renderers the complete octree was stored, every node was subdivided and all leaves were one voxel in size. The information at higher levels of the tree was a lossy abstraction of the information lower down.

In Levoy's case, the value stored was a boolean indicating whether the node contained only transparent material which could be traversed in one step. If the node was not completely transparent rendering would continue one level lower down the tree.

Danskin and Hanrahan's renderer stored the maximum, range and average values of the children at each node. The level a ray traversed the volume could be adapted to the user-defined acceptable error level. Regions that were nearly constant or nearly transparent could be traversed at high levels of the tree. We note that they chose the structure because "This is a compact representation for a pointerless octree allowing efficient neighbor, parent and child calculations" [DH92, p. 93], a powerful argument in favour of its use in volume rendering.

Laur and Hanrahan use a volume pyramid where each node stores the average of the child nodes. Their renderer used this pyramid for a splatting method that render images to a user-specified time limit, which can be as low as required to allow interactive animation.

A volume pyramid for the purposes of this thesis must store a single classified byte in each node, and effectively no pointers as an array representation can be used. As was discussed before, there are $\frac{8}{7}$ as many nodes as leaves. The simplifying approximation made on page 28, that the tree is balanced, is very well satisfied as the total tree is stored. The storage required for a 256^3 volume is thus 18Mb, regardless of the classification scheme or the coherence of the volume.

As shall be seen in section 3.4.2, with some extensions to this data set it is possible to traverse from one leaf to the next in constant time.

This is the cheapest way surveyed here to store the Visible Woman data set octree, and it has no further cost for volumes with no spatial coherency. It seems no coincidence that it has been used in the volume rendering applications surveyed, and it forms the basis of the octree representation in this thesis, described in section 3.3.

2.5.4 Wavelet methods

Wavelet analysis is a sophisticated mathematical technique that is a current topic of research in computer graphics in a number of areas, for instance image compression and feature analysis. Its use in optimising volume rendering is similar in principle to octree methods, but is more sophisticated and generates a sparser representation of the data set, so images can be generated with fewer splats in object space methods or a larger step size in image space methods.

Wavelet representations of data involve a multiresolution pyramid where each level stores information with half the resolution of the level above. In reasonable data sets, the finest levels are very sparse. A strength of the wavelet transform is its compact spatial and frequency support. The simple averaging commonly used in the volume pyramids described in the previous section may be viewed as a box filtering operation[9] very localised in space, but infinite in the frequency domain. In contrast, operations on Fourier transforms as in the section before have good localisation in the frequency domain but are infinite in extent in space. As Westermann notes [Wes96, p. 20], "This results in poor compression ratios and does not allow analyzing the signals [sic] characteristics locally."

Wavelet analysis offers a compromise: a transform that is reasonably localised in both space and frequency. Westermann writes [p. 36] concerning B-Spline wavelets that the spatial support of this type of wavelet is proportional to its order, and its Fourier transform tends to a perfect band-pass filter as order increases — "Thus, arbitrary localizations in space or frequency can be achieved." Characteristics of the signal can be examined on a local, rather than global, basis.

While wavelet analysis provides a useful basis for feature extraction and other ways of gaining information about volumes, and excellent compression ratios, in terms of how many wavelet coefficients need to be stored to reconstruct the volume accurately, performance is poor. Westermann admits [Wes96, p. 130] "Only for very sparse representations which allow one to take advantage of the proposed acceleration techniques, [do] the overall rendering times come close to the ones that can be achieved with standard visualization techniques." Rendering speed depends greatly on the spatial support of the wavelet used, as reconstructing density from a wavelet representation is a complex process.

2.6 Discussion

In this survey of the field of DVR, we have discovered a number of important points.

Firstly, material classification, the process of mapping optical properties to density samples, must occur in a smooth fashion. Binary decisions must be replaced with schemes where materials can blend gradually into one another, to eliminate noise and reduce the effects of errors in classification. Samples often represent more than one material and should be able to be classified as such.

Useful optical models need not be physically based to yield good visualisations of data, but a certain degree of realism is necessary to meet the user's expectation about how things should look. If surfaces are to be visualised, they should look like surfaces, and if bones are to be visualised they should look white, diffuse and opaque. An optical model that provides this flexibility without requiring excessive computation is the emission-absorption model, with low albedo materials that require only one light scattering event to be modeled.

[9]i.e. a Haar wavelet.

Splatting and ray casting were compared, the first is fast while the second is accurate and flexible. While other high quality object space methods exist, ray casting is faster and certainly easier to implement due to the intuitive relation between casting rays and evaluating line integrals. As these rendering methods use large amounts of computing resources, parallisability of algorithms is often discussed. Ray casting can be parallelised on a per ray basis, but a large portion of the data set must be stored at each computing node. Object space methods, on the other hand, are parallelisable on a per cell or per voxel basis, so the memory required at each node is small. However, each node must have fast access to the image buffer for each compositing operation.

Octrees were found to be a good way to optimise rendering, as they support a number of accelerations. Fourier rendering is extremely fast, but limited in the optical models it can evaluate and the shading functions it can use. Wavelets offer marvelous theoretical properties for a number of purposes but require complex calculations that mean rendering is faster only in a few cases.

Given this summary, we can refine the goals of this thesis described in section 1.3. A valuable contribution to the current state of DVR will have the following features. It would:

- use a piecewise linear classification method,
- implement a low albedo emission-absorption optical model with a Phong or Lambertian type illumination function.
- use a ray casting algorithm,
- use an octree representation of the volume.
- aim at *high quality*, since fast approximate rendering is already a well-explored topic.

Chapter 3

Ray Caster Design

A ray caster has been written to explore the points raised in the previous chapter. The renderer is designed with a primary goal of accuracy instead of rendering speed, although efficiency is addressed when accuracy is not affected.

We shall describe the renderer in the following fashion. Firstly, we shall discuss the rendering pipeline as used by the algorithm, from raw data to finished image. Secondly, we shall discuss each step in the pipeline in turn, explaining how the renderer performs the particular function and evaluating quality and efficiency issues. Then the time complexity of the renderer will be discussed. Lastly, we shall summarise with a discussion on how the method fulfills the goals enumerated at the end of the previous chapter, and how it therefore adds to the field of direct volume rendering.

3.1 High Level Structure

The basic tasks involved in producing an image are displayed in figure 3.1. They may be divided into two steps: preprocessing and rendering. The preprocessing step prepares the data for rendering, and consists of classification and octree generation.

The rendering step takes the classified octree and casts rays through it for each pixel. Each ray must traverse the octree, reconstruct a density field from discrete samples, resample the field along the ray and illuminate the resampled data. This gives us the parameters for the integration code which evaluates the rendering integral to give a colour and a transparency, which are composited onto the ray's state. When the octree traversal is complete, the final colour is stored for display.

It may be argued that a more rigorous approach to rendering would reconstruct data before classifying it, but this renderer does not do this. Classification occurs in a preprocessing step, but reconstruction is delayed until rendering. In well sampled data, there is no appreciable difference in quality involved in changing the order of

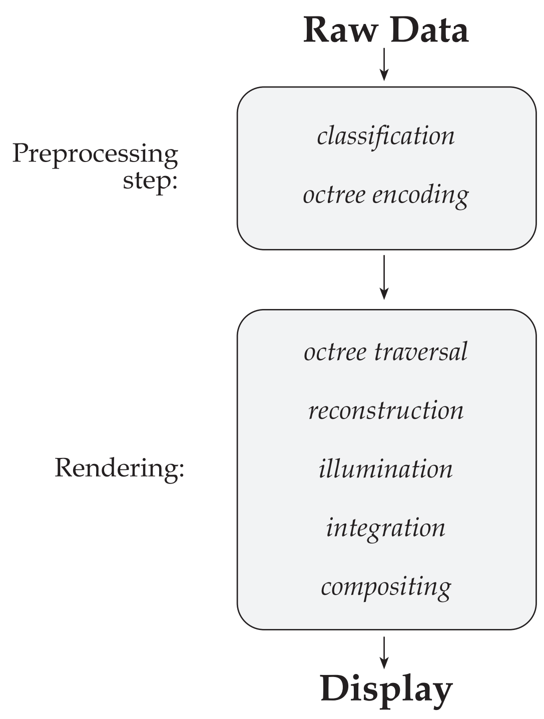

FIGURE 3.1: The rendering pipeline

these steps, but there is a large difference in efficiency as shall be described in the next section.

The parameters for each image, the volume, light source and eye data, collectively called a *scene* in this thesis, are stored in using an easily readable format the details of which are in Appendix B. Three files specify a scene:

- `foo.volume`

  The volume data set. This is stored as a 3D array of byte samples prefixed by a header containing the dimensions of the array.

- `foo.material`

  The material file. This plain text file stores the classification parameters which define the transfer functions from raw density to ambient colour, diffuse colour and opacity.

- `foo.scn`

  The scene file. This file, also plain text, stores pointers to a file of each of the previous types, and the transformations to be applied to the volume and the camera. This file also contains the direction to the light source.

Thus the components of a scene can be reused conveniently, and the scenes can be generated by popular text manipulation languages if desired.

3.2 Classification

The classification scheme used in the renderer is equivalent to Levoy's method and also to Drebin, Carpenter and Hanrahan's method, both of which are described in section 2.1. The user can create a set of materials with density ranges and optical properties, which in turn define a set of piecewise linear transfer functions from density to optical properties. The user interface thus resembles that of Drebin *et al* more than Levoy's.

The optical properties that are relevant to an emission-absorption renderer are simply the union of the parameters used by the illumination function and the attenuation parameters. The use of these properties is more fully discussed in section 3.6.

The transfer functions are constant for densities certain to be of one particular material, and linearly crossfaded for uncertain densities, as in figure 3.2. A raw density that is not classifiable with certainty is assigned a set of probabilities that it represents tissue of a particular material. The optical properties used to render samples of that density are a mixture of the properties of each material represented by it, so if a sample has a 10% probability of being bone and 90% probability of being muscle, it has an appearance that is 10% like bone and 90% like muscle. Probabilities can thus be interpreted as the proportion of a material present at a point. This interpretation is used below for ease of explanation.

The classification algorithm uses two steps to assign optical properties to a raw sample, both driven by table lookup. The first remaps the samples in the volume to classified values during preprocessing, which is the classification step proper. This value is later used as an index to the optical properties lookup table at render time, which gives all the information necessary for illumination. Thus we have two functions $f_1 : \mathbb{N} \to \mathbb{N}, f_1(\rho_{raw}) = \rho_{class}$ and $f_2 : \mathbb{N} \to \mathcal{P}, f_2(\rho_{class}) = p$, where $\mathcal{P}$ is the set of optical properties and $p \in \mathcal{P}$.

The first lookup table f_1 is formed in the following manner. Assume there are M materials to be classified, and that there is a desired level of classification detail D. This step takes the raw densities and maps them in such a fashion that there are M regularly spaced entries corresponding to the M materials, separated by $D - 1$ intermediate steps corresponding to mixtures of materials, giving a total of $(M-1)D + 1$ entries. D is chosen so that $(M-1)D + 1 < 256$, allowing the value to be stored in a single byte.

Each material description is read in from the material file. Included in the description is a range $[\rho_{min}, \rho_{max}]$ of raw densities within which samples are classified as being entirely of that material. All samples with raw densities within the range for material number m are given the classified value mD.

Samples with densities outside any material's density range are given a value which linearly interpolates the two adjacent materials. So a sample half way between the ranges for materials m and $m+1$ will have the classified value $\frac{mD+(m+1)D}{2} = (m + \frac{1}{2})D$, denoting the 50–50 mixture of two materials assumed to be present in a sample. This is illustrated in figure 3.2.

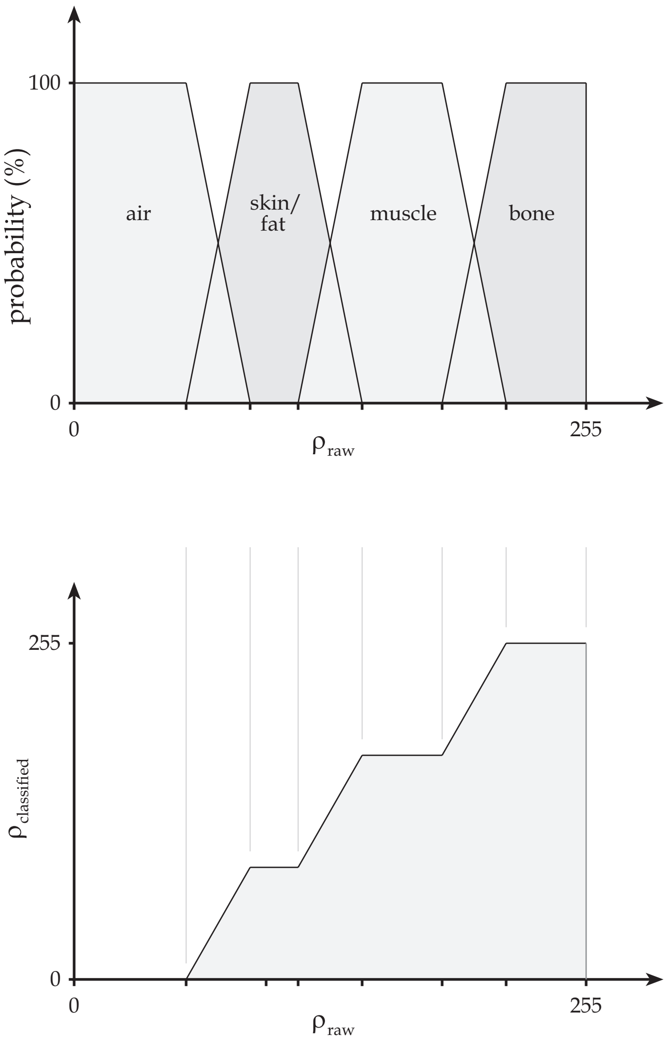

FIGURE 3.2: Classification

The advantages of this scheme are that, firstly, the entire set of optical properties can be represented by a single byte, which is a considerable saving in space. Secondly, the octree can be built using this classified density instead of raw density. Regions known to be entirely of one tissue type can be stored in a single leaf even if the raw density ranges over the entire range associated with that tissue type.

The savings in the number of leaves is considerable: octrees were built out of the two data sets referred to when discussing octree datastructures on page 2.5.3, using both the raw density and the classified density. The Visible Woman data set takes 6 486 276 leaves to store when raw values are used, but only 4 814 886 (74% of the former value) when classified into air and flesh. The pelvis data set takes 5 750 678 leaves for raw density, but 1 297 313 (22%) when classified into air, skin / fat, muscle and bone, and only 185 601 (3.2%) when classified into air and bone.

The second lookup table f_2 is generated in a similar manner. Each classified density represents a mixture of some proportion of at most two materials. The optical properties of those two materials are mixed in the same proportion and stored in the table. The result is a table containing a linearly changing set of optical properties with entries for each material and each mixture of materials contained in the volume.

3.3 Octree Encoding

3.3.1 Overview

The octree stores the volume data set after material classification. It is used to accelerate rendering in the usual manner — it enables the renderer to traverse empty space and homogeneous areas rapidly. The octree stores density values in a lossless fashion, with each leaf node storing the density of the identical cells in that node.

As the renderer is designed for 8-bit scalar data sets on a 32-bit machine, the value to be stored at a leaf is one byte large, while pointers, if they were used, would be four bytes large. This leads to the observation that storage of internal nodes in a pointer-based scheme will be costly compared to the storage of leaf nodes, as pointers are large compared with density values. This is not the case for many octree implementations, e.g. [Gla84].

Thus a volume pyramid scheme is used, as the lack of pointers substantially reduces memory cost. For the purposes of this renderer, there is no benefit in storing the complete octree, but the wasted space is far smaller than the housekeeping information of the schemes surveyed in section 2.5.3 on octree datastructures.

We shall discuss the datastructure itself, and then the main methods required to implement it: construction and leaf access.

3.3.2 The pyramid datastructure

The pyramid is based on cells within the volume, so leaves correspond to sets of cells rather than sets of voxels. We shall first show how the datastructure efficiently finds leaves, and secondly how the datastructure deals will variable density cells where a single byte sample does not adequately represent the value of the cell.

Finding a leaf

Unlike the three volume pyramid implementations described in section 2.5.3 where the highest level that satisfies the error metric is traversed, this renderer performs a leaf-by-leaf traversal and thus needs to know which level of the pyramid stores the leaf for a particular cell. Details of this traversal are in section 3.4. One method for storing the leaf information would be simply to store a boolean `leaf` for each element of the pyramid that denotes whether the element is a leaf or not. To access the data for a given voxel, we examine each level of the pyramid until we find the element referring to that voxel which has `leaf` set.

This scheme increases the cost of every element of the pyramid by one bit, in other words multiplying it by $\frac{9}{8}$. Recall that there are $\frac{8}{7}$ times as many nodes as cells, and we can see that the cost is:

$$\begin{aligned}\text{Total cost} &= \frac{9}{8} && \text{bytes per element}\\ &= \frac{9}{8}\frac{8}{7} && \text{bytes per cell}\\ &= \frac{9}{7} && \text{bytes per cell}\end{aligned}$$

However, we can improve on this and remove the need to search through the levels of the pyramid to find the leaf node. For each cell c in the volume, we will store the index of the level of the corresponding leaf, $l(c)$. This at first appears to be an expensive thing to store, but the cost can be reduced by the following observation:

If eight cells $\{c_1, c_2, \cdots c_8\}$ have coordinates whose binary representations are identical except for the least significant bit, these eight cells have the same value $l(c_i)$.

Proof: If all eight cells have the same density, they will be part of the same leaf node and so $l(c_i)$ will be the level of the same leaf and must be identical. If the cells have different density, they will each be separate leaves and thus $l(c_i) = 0 \;\; \forall\, i$.

The $l(c)$ information can be stored in an identical fashion to the second level of the pyramid, as one byte for every eight cells. We avoid the need to find an efficient way to add a bit to each element of the pyramid. The memory cost is thus:

$$\begin{aligned}\text{Cost} &= \frac{8}{7} + \frac{1}{8} && \text{bytes per cell}\\ &= \frac{71}{56} && \text{bytes per cell}\end{aligned}$$

Variable density cells

There are then two circumstances under which we can have a cell-sized leaf. A cell may be a leaf because, while the reconstructed density varies over its volume, it cannot be subdivided further. Cells may also be leaves because they have constant density.

The octree representation must differentiate between these two types of cell sized leaves as the type with varying density is reconstructed and integrated with care, while the second needs no reconstruction and simple integration as it is constant.

The original data set, a set of voxels, is stored in the base level of the pyramid. However, this level also stores the set of cells, both those with constant and those with varying density. To extract information about a particular varying density cell $C_{x,y,z}$, we retrieve the densities of eight adjacent voxels: $\{V_{x+i,y+j,z+k} \mid i,j,k \in \{0,1\}\}$. To extract information about a constant cell $C_{x,y,z}$, we need only the voxel $V_{x,y,z}$.

When examining a cell-sized leaf, the datastructure must be able to inform us efficiently what type of cell it is. One method would be to examine all the vertices of the cell and test whether they are all constant or not. This would effectively be doing the work of octree construction over again. It is faster to store this information when the octree is constructed, as a packed array of boolean variables. The extra amount of memory allocated is only one eighth the size of the original data set.

The total cost for the octree is the previous $\frac{71}{56}$ bytes per cell plus an additional $\frac{1}{8}$ bytes, totaling $\frac{78}{56}$ of a byte per cell. The access time is very short, as no searching, iteration or recursion is required. Given a particular cell, the index of any level of the pyramid or the leaf level information can be found with integer shift operations.

3.3.3 Construction and leaf access

The construction of the octree, given a volume `theVolume`, must first collect cell homogeneity information and secondly find leaf information. The tree is built upwards, from leaf to root. Internal nodes are tested for whether they satisfy the requirements of leaves, and if that is the case the tree is pruned. Pseudocode for the constructor is as follows:

```
Allocate memory;
Copy the contents of theVolume into pyramid[0];

\\ Homogeneity:
for (every cell c of level 0)
    if (all vertices of c have equal density)
        set homogeneity information for c;
    else
        clear homogeneity information for c;

\\ Leaves:
for (int level = 1; level < height; level++)
{
    for (all cells (i,j,k) on this level)
    {
        bool collapse = true;
        if some of the children are not leaves,
            collapse = false;
        if the children have differing density,
            collapse = false;
        if (level == 1)
            if a child has varying density,
                collapse = false;

        if (collapse)
        {
            Set this cell to the density of its children;
            Mark it as being a leaf;
        } else {
            Set the cell to its children's average density;
        }
    }
}
```

The other important method of the octree object is leaf access. The traversal method described in the next section queries the octree with a cell index, wanting the corresponding leaf information (density, sidelength, position and homogeneity). To get this data for a given cell `(x,y,z)` that occurs in that leaf is simple:

```
quadruple get(int x, y, z)
{
    int     level   = leaf.get(x/2, y/2, z/2);
    vector  (i,j,k) = (x >> level, y >> level, z >> level);
    vector  corner  = (i << level, j << level, k << level);
    int     density = pyramid[0].getDensity(x, y, z);
    bool    homog   = getHomogeneity(x, y, z);
    return  (density, 2^level, homog, corner);
}
```

The various `get` methods referred to in the code are simple array lookups and integer operations in constant time. For clarity, the pseudocode has been given a slightly different structure to the actual code given in the files `\code\*\COctree.cpp` on the attached CD-ROM for ease of explanation.

3.3.4 Summary

The octree datastructure has two fields apart from the volume pyramid itself. Firstly, leaf information is stored for every eight cells, using a one byte index into the pyramid. Secondly, cell-sized leaves with variable density are distinguished by a boolean map of every cell. Both of these extra fields take up only an eighth the size of the original volume.

The leaf access method required by the traverser takes only constant time, finding the relevant leaf by means of the leaf index field. All information necessary for the renderer can be found easily using integer shift operations, without climbing or descending the tree at any point.

3.4 Octree Traversal

During rendering, each ray must traverse the octree, partitioning the rendering integral into small intervals that can be evaluated without significant error. Samples are taken as the ray enters and leaves each leaf, and at intermediate points within leaves of variable density.

An adaptive traversal of the octree such as that of Danskin and Hanrahan [DH92] would be a positive feature for the renderer, but is currently unimplemented. In their system, the user could select an acceptable error limit, and rays would traverse the octree at various levels according to the error metric. As our renderer is intended primarily for high-quality imaging, the term “acceptable error” is to be treated carefully. It may be, perhaps counterintuitively, that speeding rendering with a very low error would make practical other techniques for increasing the quality of rendering, and thus be a positive step. This is discussed as possible future work in the conclusion of this thesis.

We shall first survey several traversal algorithms that have been reported in research, before discussing the final choice used in this thesis.

3.4.1 A recursive method

A numerically robust algorithm discussed in [FvDFH90, p. 552] traverses the octree from leaf to leaf. This is a promising algorithm as it corresponds well to the requirements of a ray caster where rays traverse the volume in a leaf-by-leaf manner. Foley and van Dam’s method ascends the octree from the current leaf until it finds a parent of both the current and next leaves. It then descends through the tree in a mirror image to the ascent, until finds the correct leaf. It is possible to compute the reflected path by means of table lookup, and to avoid floating point arithmetic. However, the algorithm requires fast ascent and descent of the tree, which proved to be a problem for [Lev90] where similar tree climbing was required.

3.4.2 Glassner's method

This traversal algorithm is well adapted to the octree representation used by the renderer. It is not recursive, but instead can make use of the fact that octree leaf access is done in constant time by a couple of array lookups as was shown on page 42. It was advanced by Glassner [Gla84], who uses it to accelerate surface ray tracing, and adapted here for use in our renderer.

The basic form of the algorithm is that we find a point guaranteed to be in the next leaf by advancing along the ray a certain distance. We then use the point to query the octree object, which finds the leaf containing that point. Glassner's original algorithm used the point's position to generate a key to a hashtable of leaves, as was described in detail in section 2.5.3. We shall describe the algorithm in more detail in a fashion consistent with volume rendering.

```
currentPoint = where the ray enters the volume;         Note (1)
currentCell  = the cell containing currentPoint;        Note (2)
while (currentCell is inside the volume)
{
    leafInfo = octree.get(currentCell);                 Note (3)
    Use leafInfo to integrate the ray over the leaf;
    Update ray colour and opacity;
    Set currentPoint to a point within the next leaf;   Note (4)
    Update currentCell to match the new currentPoint;
}
```

Notes:

1. We intersect the ray we are tracing with the surface of the volume. The ray needs to be tested only against the three faces (in the worst case) visible from the camera, and the position of those faces are precalculated during initialisation.

2. This is an easy operation in a sensible coordinate system: the coordinates of the cell are the coordinates of the point rounded down to the nearest integer.

3. The `COctree` class has a method `get()` that returns the necessary data concerning the leaf containing a given voxel, as described in the previous section.

4. This is the core of the algorithm. To find a point in the next cell, we simply find the time the ray exits the current leaf, and add an infinitesimal amount.

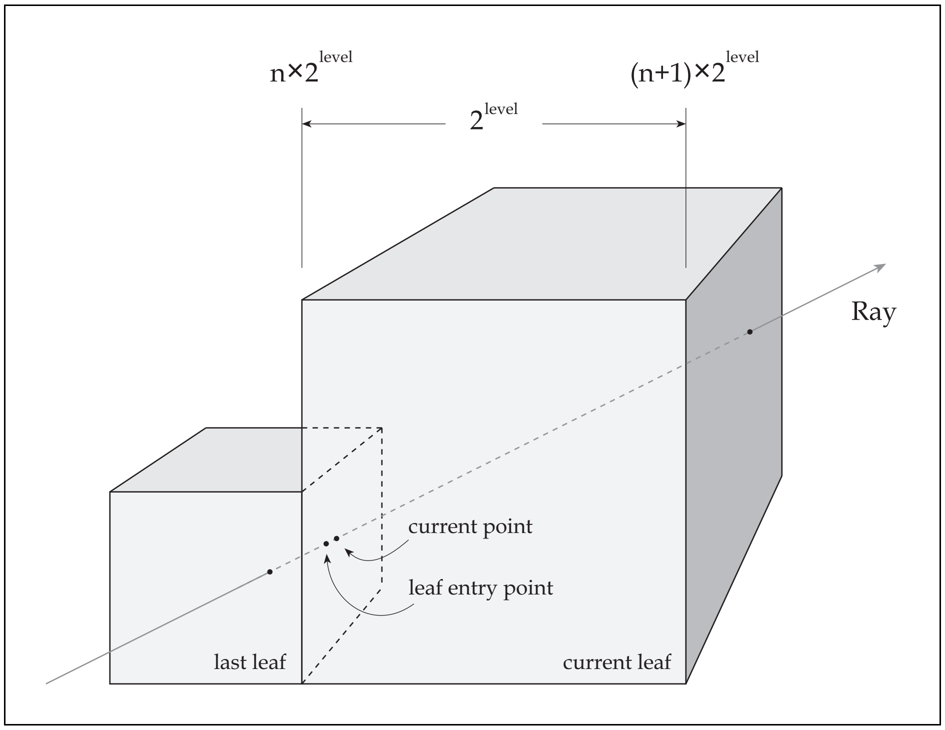

FIGURE 3.3: Leaf traversal

This is illustrated in figure 3.3. The leaf is an axis aligned cube whose faces are obtained by the current cell position and the size of the leaf, 2^{level}. In Glassner's original method, the ray is intersected with all six faces of the cube. Glassner notes that it is possible, but not worth the effort, to optimise this down to four. Here, we intersect the ray with only three faces as follows.

The left face of the leaf is the plane $x = a$ where a is `currentPoint.x` rounded down to the nearest multiple of 2^{level}. The right face is $x = a + 2^{level}$. We can find the other four faces similarly.

To reduce the number of necessary plane intersections from six to three, we note that if the x component of the ray's motion is positive, it is impossible for the ray to exit the leaf by the left face and we need only test for the right face, and *vice versa* if the x component is negative. Similarly we can cull two more tests based on the y and z components of the rays. The cost of halving the number of tests in this manner is thus three floating point comparisons per ray.

The algorithm then finds the times the ray passes through the three relevant faces, and takes the minimum of those, t_{min}, to be when the ray exits the leaf.

Numerical Stability

This algorithm, with its dependence on floating point calculations at certain points, is vulnerable to numerical stability problems which depend on how we find a point guaranteed to lie within the next leaf.

The point $\mathbf{p}$ on the ray at time t_{min} *should* lie on the plane dividing the current leaf from the next. It is the nature of floating point hardware that $\mathbf{p}$ will often lie some small but non-zero distance from the plane. There will be occasions when $\mathbf{p}$ is some distance inside the current leaf, and so the renderer will fail to traverse to the next leaf. In this case, the program will never terminate as the ray will loop infinitely within the current leaf.

The solution to this problem used by Glassner is complex and robust. To guarantee finding a point within the next leaf but not overshooting it, he keeps track of the minimum sidelength of a leaf — in our case the length of a cell. The point $\mathbf{p}$ is translated away from the leaf face by half this minimum length, preventing the possibility of overshooting the leaf.

Glassner also handles the cases where the point $\mathbf{p}$ is on an edge or a vertex of the new leaf, in which case $\mathbf{p}$ is translated diagonally. This makes his solution reliable for every possible case.

We have found that a simpler solution that still works reliably in practice, which is simply to advance the point $\mathbf{p}$ along the ray by a small amount ε. ε is set large enough to ensure that the probability of `currentPoint` still being in the current leaf is negligible.

However, if ε is set at too high a value, there is a chance that the code erroneously skips leaves. The distance a ray travels through the next leaf can be infinitesimally small if the ray passes through near a corner, and if it is smaller than ε the traverser will skip that leaf entirely.

The solution used in this renderer is to have the traverser use a small value for ε, and record the last face passed through. Under normal circumstances the value $\varepsilon = 0.001$ cell lengths is used, and the probability of skipping leaves is small. Even if a leaf were skipped, the contribution from it would be infinitesimal as its intersection with the ray must be less than 0.001 cell lengths long. The traverser keeps track of the last leaf face the ray passed through, and so can detect if it has got stuck within the same leaf. The last face is represented with integer values, instead of simply the last t_{min}, so this detection is not prone to floating point uncertainty.

If the traverser detects that it has got stuck, it continues in a constant step fashion until it gets to the next leaf, using a stepsize of 10ε. While this measure represents a theoretical failure case of the algorithm, in practise the constant step code is often not used when rendering an image, even test images specifically designed to exploit the flaw. This solution is sufficiently reliable as well as being fast and easily implemented.

3.5 Reconstruction

3.5.1 Discussion

The task of reconstruction is to define a continuous field corresponding to the discrete samples provided by scanning devices. This is a well studied area in signal processing and a large body of relevant literature exists. Most discussions on the topic start with a one-dimensional case, from which extension to three dimensional space is trivial.

Assume we have a function $f(x)$ which we sample at a constant rate d. We wish to generate from the samples an approximation $\widehat{f}(x)$ as close to the original as possible. Sampling theory states that it is possible to recover the function exactly, in other words to satisfy $f(x) = \widehat{f}(x)$, if the following two conditions are met: Firstly, the function $f(x)$ must be band limited so that none of the energy of the function occurs with frequencies higher than some frequency ω. Secondly, the sampling frequency $\frac{1}{d}$ must satisfy the equation $\frac{1}{d} > 2\omega$, i.e. the function must be sampled at at least double its highest frequency component. The quantity 2ω is known as the *Nyquist frequency*.

These conditions are not met for many practical applications, as any abrupt jump in $f(x)$ produces non-zero frequency components right up to infinity. However, we must assume that data sets are well sampled, i.e. sampling is smooth enough that high frequency components do not occur, or at least do not occur often. This is not too unreasonable since, as was discussed in section 2.1 on classification, most scanning technologies sample a weighted area around a point rather than the point itself, thereby smoothing out high frequency variations.

If these conditions are met, the function can be recovered exactly by convolving each sample with a $sinc$ function, the inverse Fourier transform of a box function. These two functions are defined as follows:

$$sinc(x) = \begin{cases} 1 & \text{if } x = 0 \\ \frac{\sin(\pi x)}{x} & \text{otherwise.} \end{cases}$$

$$Box(x) = \begin{cases} 1 & \text{if } |x| < k \\ 0 & \text{otherwise} \end{cases}$$

and the convolution operation is defined

$$f(x) * h(x) = \int_{-\infty}^{\infty} f(\alpha)h(x-\alpha)d\alpha$$

where $h(x)$ is the *reconstruction filter*, and $f(x)$ is the sampled function,

$$f(x) = \begin{cases} \rho(x) & \text{if } x \text{ is an integer,} \\ 0 & \text{otherwise.} \end{cases}$$

In practise, however, the $sinc$ filter is not used because it extends infinitely over $\mathbb{R}$ and would require an infinite amount of calculation. However, this pattern forms the basis of reconstruction. Instead of convolving our samples with such a complex filter, we can use simpler ones. The convolution is in practise not as difficult to evaluate as it appears since one of the functions is a discrete set of samples, reducing the convolution to a weighted sum. Also, practical filters extend only a short distance before becoming zero, so there are only a few terms in the sum.

Many filters have been designed for reconstruction, and most are adaptable to our three dimensional problem. The characteristics of a filter are firstly its extent, i.e. the radius of the interval within which it is non-zero, which is the basic measure of the calculation required for reconstruction. Secondly, filters are characterised by how closely they match the $sinc$ function. Marschner and Lobb [ML94] offer a number of metrics for filters that describe how a filter's Fourier transform departs from the ideal box shape of the $sinc$ function's transform.

Filters that resemble the $sinc$ function closely generally have large extents, and prove to be too slow for volume reconstruction. The best filter commonly encountered in volume rendering research is the *trilinear* filter $h_{trilin}(x, y, z)$, defined as:

$$\begin{aligned} h_{lin}(s) &= \begin{cases} s+1 & \text{if } s \in [-1, 0) \\ 1-s & \text{if } s \in [0, 1] \\ 0 & \text{otherwise} \end{cases} \\ h_{trilin}(x, y, z) &= h_{lin}(x)h_{lin}(y)h_{lin}(z) \end{aligned}$$

This filter has the disadvantage that it does not give a smooth result, as shown in figure 3.4. However, this effect is not extreme when used for well sampled volumes, and in practice the filter is adequate to its task.

3.5.2 Implementation

We shall describe the implementation of trilinear interpolation in greater detail. For simplicity, we will assume that the cell under discussion is at $(0, 0, 0)$. Let $\mathbf{r}(t) = \mathbf{m}t + \mathbf{c}$ be the ray, and let $\rho_{x,y,z}$ be the density of voxel (x, y, z).

Let the reconstructed density be $\widehat{\rho}(t)$.

Let $\mathbf{s}(t) = (1, 1, 1) - \mathbf{r}(t)$, and let the x, y and z components of a vector $\mathbf{a}$ be denoted a_x, a_y, a_z respectively. Then the reconstructed density is the weighted sum of the densities at the eight vertices of the cell:

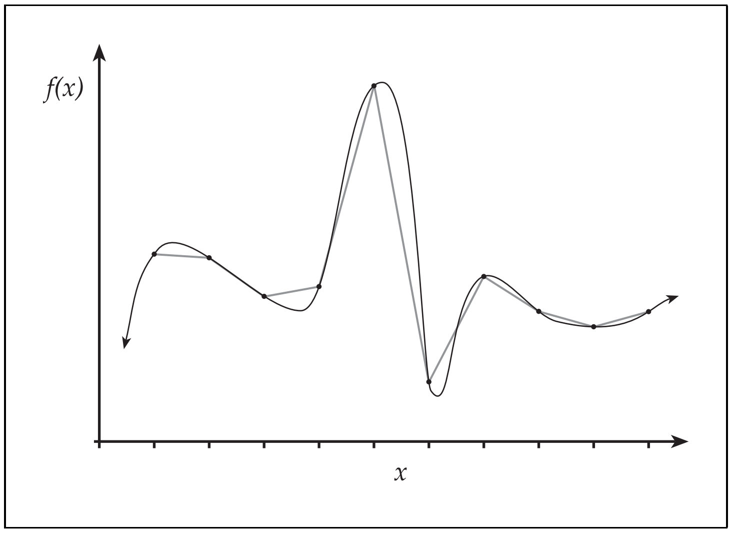

FIGURE 3.4: Linear interpolation

$$\begin{aligned}
\widehat{\rho}(t) \quad = \quad & \rho_{0,0,0}\ s_x s_y s_z\ + \\
& \rho_{0,0,1}\ s_x s_y r_z\ + \\
& \rho_{0,1,0}\ s_x r_y s_z\ + \\
& \rho_{0,1,1}\ s_x r_y r_z\ + \\
& \rho_{1,0,0}\ r_x s_y s_z\ + \\
& \rho_{1,0,1}\ r_x s_y r_z\ + \\
& \rho_{1,1,0}\ r_x r_y s_z\ + \\
& \rho_{1,1,1}\ r_x r_y r_z
\end{aligned}$$

The function of density along the ray path is thus a cubic function of the ray parameter t. As shall be seen, evaluating the rendering integral with cubic parameters is not possible, so this function is calculated only at the start and end of each integration step, and the result linearly interpolated over that portion of the ray.

For points between $\mathbf{r}(t_{in})$, the start of an integration step, and $\mathbf{r}(t_{out})$, the end, we linearly interpolate:

$$\widehat{\rho}(t) \quad = \quad \widehat{\rho}(t_{in}) + \frac{t - t_{in}}{t_{out} - t_{in}}\Big(\widehat{\rho}(t_{out}) - \widehat{\rho}(t_{in})\Big) \qquad \text{for } t \in (t_{in}, t_{out})$$

Linearly interpolating the density over the ray parameter is about as accurate as true trilinear interpolation. Its only defect is that the density at a point will differ depending on which ray is being used to reconstruct it. This could lead to changes in the resulting image as the camera is moved relative to the volume. In practice, though, these changes do not result in visible artifacts as the stepsize of the method is sufficiently small.

Gradient Calculation

Most illumination functions require the gradient of the density function, $\nabla\rho$, to compute the angle between a surface normal and a point light source. Novins in [Nov93, p. 33] describes a debate as to whether one should take the gradient of the raw data field or of the classified data field. [Lev88] uses the raw data, while [DCH88] uses classified data. Novins himself uses the raw data, claiming that either choice is possible as the debate has not been settled.

In this thesis, we also use the raw data for the following reason. After classification, some of the smooth variation between tissue types is lost. The classification step takes a smoothly varying function and transforms it into a union of distinct regions with reasonably sharp boundaries. At the most important points for normal-based shading, namely material transitions, accurate and smooth reconstruction of $\nabla\rho$ is most difficult. This is exacerbated by the fact that gradient filters are high-pass filters and tend to exaggerate noise. Möller *et al* [MMMY96], for instance, find that for equivalent methods errors in gradient reconstruction were larger than errors in interpolation.

The result of using classified data is that gradient vectors are not very accurately reconstructed, and images have very poor quality surfaces. The artifacts introduced by bad gradient reconstruction clearly delineate voxels giving a rough appearance to surfaces within the data, consistent with [MY96, p. 368] where reconstruction errors are shown to increase with distance from the nearest sample. Thus, the raw data set with its smoother material transitions is preferred.

There are a number of approaches to gradient reconstruction, the general approach (for instance [MMMY96][MY96]) being to take a density reconstruction filter and differentiate it. We employ three variations on this theme for use in the renderer, the derivative of the trilinear interpolant, central differences evaluated at cell vertices and trilinearly interpolated over the interior of the cell, and central differences tricubically interpolated.

The trilinear gradient.

As the trilinearly reconstructed field is a cubic function, its gradient is easily calculated. As was shown above,

$$\begin{aligned}\widehat{\rho}(t) \;=\; & \rho_{0,0,0}\; s_x s_y s_z + \\ & \rho_{0,0,1}\; s_x s_y r_z + \\ & \rho_{0,1,0}\; s_x r_y s_z + \\ & \rho_{0,1,1}\; s_x r_y r_z + \\ & \rho_{1,0,0}\; r_x s_y s_z + \\ & \rho_{1,0,1}\; r_x s_y r_z + \\ & \rho_{1,1,0}\; r_x r_y s_z + \\ & \rho_{1,1,1}\; r_x r_y r_z\end{aligned}$$

$$\begin{aligned}\Rightarrow \qquad \frac{\partial\widehat{\rho}(t)}{\partial x} &= (\rho_{0,0,0}-\rho_{1,0,0})s_y s_z + (\rho_{0,0,1}-\rho_{1,0,1})s_y r_z + \\ &\quad (\rho_{0,1,0}-\rho_{1,1,0})r_y s_z + (\rho_{0,1,1}-\rho_{1,1,1})r_y r_z, \\ \frac{\partial\widehat{\rho}(t)}{\partial y} &= (\rho_{0,0,0}-\rho_{0,1,0})s_x s_z + (\rho_{0,0,1}-\rho_{0,1,1})s_x r_z + \\ &\quad (\rho_{1,0,0}-\rho_{1,1,0})r_x s_z + (\rho_{1,0,1}-\rho_{1,1,1})r_x r_z, \\ \frac{\partial\widehat{\rho}(t)}{\partial z} &= (\rho_{0,0,0}-\rho_{0,0,1})s_x s_y + (\rho_{0,1,0}-\rho_{0,1,1})s_x r_y + \\ &\quad (\rho_{1,0,0}-\rho_{1,0,1})r_x s_y + (\rho_{1,1,0}-\rho_{1,1,1})r_x r_y.\end{aligned}$$

This method has the elegant feature that density reconstruction and density gradient reconstruction are consistent with each other. Also, the only samples required by the function are those already available from the density reconstruction. Unfortunately, this reconstruction is discontinuous over cell boundaries. A common approach that gives a continuous result is central differences.

Trilinearly interpolated central differences.

The central difference gradient is calculated at the vertices of the cell by:

$$D(x,y,z) = \frac{1}{2}\begin{bmatrix}\rho_{x+1,y,z} - \rho_{x-1,y,z}\\ \rho_{x,y+1,z} - \rho_{x,y-1,z}\\ \rho_{x,y,z+1} - \rho_{x,y,z-1}\end{bmatrix}$$

which, as the sampled density $f(x)$ is zero except at grid points, can be understood as the convolution of the sampled density with any filter $h_{CD}(x,y,z)$ satisfying

$$\begin{aligned} h_{CD}(x,y,z) &= \frac{1}{2} && \text{if } (x,y,z) \in \{(1,0,0),\quad (0,1,0),\quad (0,0,1)\}, \\ h_{CD}(x,y,z) &= -\frac{1}{2} && \text{if } (x,y,z) \in \{(-1,0,0),(0,-1,0),(0,0,-1)\}\end{aligned}$$

It is easy to show that such a filter is the derivative of the Catmull-Rom cubic filter, which Möller *et al* [MMMY96] described as the most accurate cubic derivative filter. If we evaluate the Catmull-Rom derivative filter at cell vertices, we find the result is the central differences filter from the above equation.

Having thus calculated the gradient at each vertex, we trilinearly interpolate this value over the interior of the cell to get a gradient field. This gives a continuous result that is a good deal smoother than the true trilinear gradient. If central difference gradients were precalculated, the result would not require much more computation than the earlier method. However this is not done as the memory required to store a vector for each voxel is prohibitive. This method is therefore slower as each gradient reconstruction requires 24 memory accesses instead of 8.

Tricubically interpolated central differences.

A third gradient reconstruction method was also explored. This is similar to the second except that it uses tricubic B-spline interpolation of the central differences gradients to give an even smoother result. The reconstruction filter h_{BS^3} is formed from a one-dimensional B-spline thus:

$$\begin{aligned} h_{BS}(s) &= \begin{cases} \frac{1}{6}s^3 + s^2 + 2s + \frac{4}{3} & \text{for } -2 \leqslant s < -1 \\ -\frac{1}{2}s^3 - s^2 + \frac{2}{3} & \text{for } -1 \leqslant s < 0 \\ \frac{1}{2}s^3 - s^2 + \frac{2}{3} & \text{for } \quad 0 \leqslant s < 1 \\ -\frac{1}{6}s^3 + s^2 - 2s + \frac{4}{3} & \text{for } \quad 1 \leqslant s \leqslant 2 \\ 0 & \text{otherwise.} \end{cases} \\ h_{BS^3}(x,y,z) &= h_{BS}(x)h_{BS}(y)h_{BS}(z) \end{aligned}$$

A lookup table for the filter is constructed, but as before the central difference gradients are not precalculated and 160 memory references are required. The performance of this filter is extremely slow.

3.6 Illumination

In section 2.2.7, the low-albedo emission-absorption optical model was advanced as being suitable for quality volume rendering. It was described fully in sections 2.2.4 and 2.2.5. The emission-absorption model is flexible and presents useful information to the user. The low-albedo assumption is superior to high-albedo methods in speed and image comprehensibility. The rendering integral used to evaluate this optical model is repeated here. Let the viewing ray be $\mathbf{r}(t) = \mathbf{d}t + \mathbf{v}$, where $\mathbf{v}$ is the viewpoint and $\mathbf{d}$ the direction. If $I(t_0)$ is the intensity of light at distance t_0, and I_b is the intensity entering the region being integrated, then:

$$I(t_0) \quad = \quad \int_0^{t_0} \varepsilon(t) e^{-\int_0^t \tau(s)ds} dt \quad + \quad I_b e^{-\int_0^{t_0} \tau(s)ds} \tag{3.1}$$

The two parameters $\varepsilon(t)$ and $\tau(t)$ are the emission and extinction coefficients, respectively. $\tau(t)$ is defined according to the density of the material at point t, and $\varepsilon(t)$ is calculated by an illumination function, taking its parameters from material density and the position of the light source.

3.6.1 Discussion on Lambertian shading

Lambertian illumination is a fast and easily implemented illumination function, giving realistic shading effects for the most part. As is usual, we define $\varepsilon(t) = k_{\mathrm{ambient}} + k_{\mathrm{diffuse}} N \cdot L$, where the constants are material properties, $N = \frac{\nabla \rho}{|\nabla \rho|}$ and L the direction to the light source. (The topic of normalising the gradient is discussed in section 3.6.4.) When this was implemented a curious effect was noticed — the isosurfaces being displayed were found to be too bright when viewed at large angles.

An example of this is given in Colour Plate II.a, where we have the sphere of opaque material referred to on page 5 being illuminated by a light source behind the viewpoint at infinity. The sphere appears flattened as its brightness does not vary much over its face. This goes against the natural expectation of a user, that a surface should vary in brightness depending on its angle. This is an important defect as shading is an important cue for surface orientation.

The scan line from the centre of the image in Colour Plate II.a was extracted and compared with a surface rendering of a Lambertian illuminated sphere. The result is displayed in figure 3.5, which clearly shows the defect. The center of the visible disk is correct, but the rim is brighter than a Lambertian shaded sphere should be. The reader will note that the volume rendering curve is slightly wider than the surface rendering curve, even though both spheres have the same radius. This is due to the sampling of the sphere in the volume dataset. The transition from transparent material to opaque material occurs over six cell lengths, and the radius of the sphere is measured from the centre of this transition. Thus the sphere has some bright material outside this range, making its curve slightly wider.

The explanation for this flat, disk-like appearance of the sphere relies on the fact that a surface illumination function has been adapted for use in a volume context, where the "surface" is actually a band six cell lengths thick. While the light emission $\varepsilon(t)$ is correctly evaluated to give a Lambertian appearance, the intensity $I(t_0)$ does not preserve this appearance. Consider the two rays in figure 3.6. As they meet the sphere at different angles, they each spend a different amount of time in the material transition region where Lambertian shading occurs. While each ray encounters the same materials in the same order, the transition takes twice as long for one as for the other.

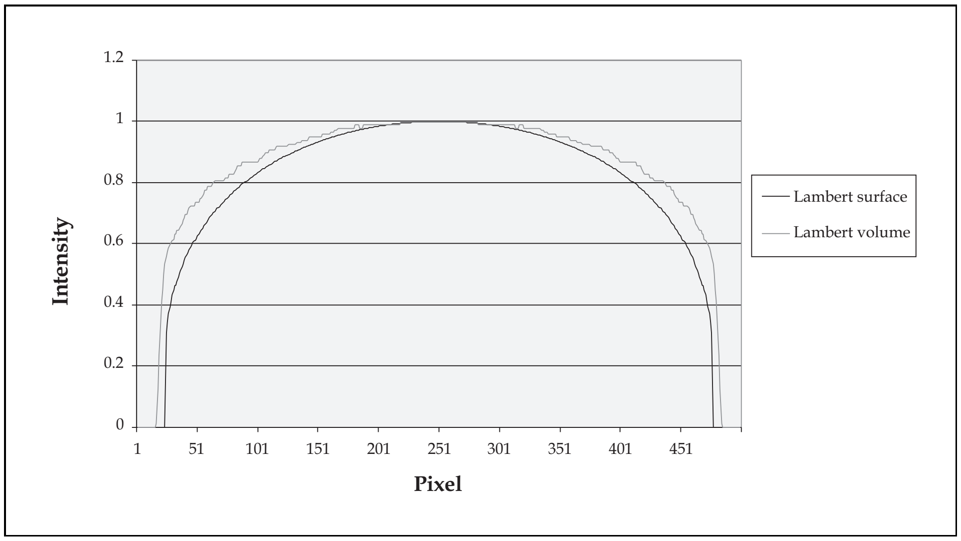

FIGURE 3.5: Surface and volume Lambertian illumination

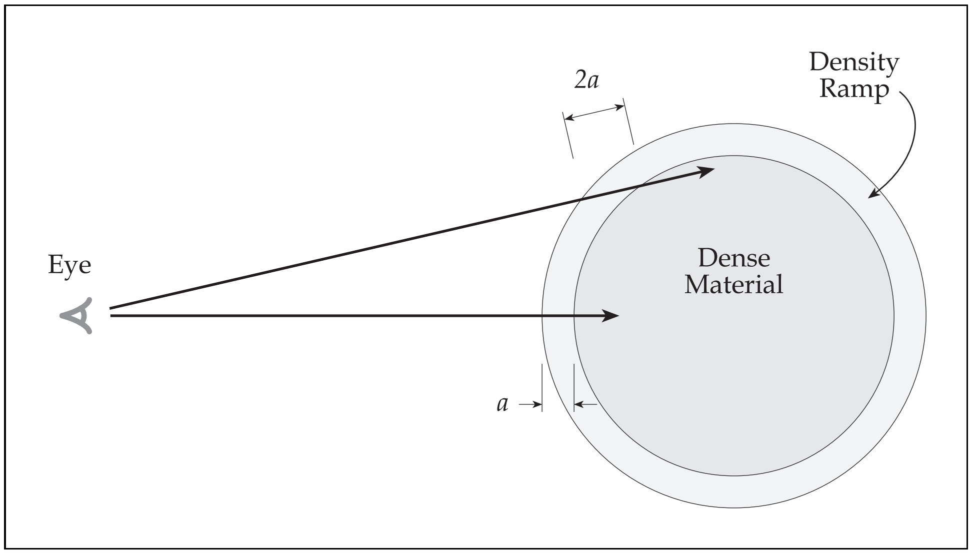

FIGURE 3.6: The Lambertian shaded sphere

We shall show that the rendering integral does not preserve intensity when material transitions are stretched in this manner. Consider two rays $\mathbf{r_1}(t)$ and $\mathbf{r_2}(t)$ that each travel from a transparent region to an opaque one, one over a distance a and the other over a distance $2a$. Let $t = 0$ be the time each ray enters the material transition. We shall make the simplifying approximation that within the material transition the density will rise linearly with t. Thus, the extinction coefficient is a linear function of t. As the density gradient is constant, and the light source is at infinity, the emission coefficient is also a linear function of t.

Let the emission coefficient within the opaque material be k_ε and the extinction coefficient be k_τ. Over the transition, the emission function encountered by $\mathbf{r_1}(t)$ is easily found to be $\varepsilon_1(t) = \frac{k_\varepsilon}{a}t$ as it rises from 0 at time $t = 0$ to k_ε at time $t = a$. Similarly, the emission function for the second ray $\mathbf{r_2}(t)$ is $\varepsilon_2(t) = \frac{k_\varepsilon}{2a}t$. The extinction functions are treated similarly and found to be $\tau_1(t) = \frac{k_\tau}{a}t$ and $\tau_2(t) = \frac{k_\tau}{2a}t$. For the first ray we have:

$$\begin{aligned}
I_1(a) &= \int_0^a \varepsilon_1(t)\ e^{-\int_0^t \tau_1(s)ds}\ dt \\
&= \int_0^a \frac{k_\varepsilon}{a}t \quad e^{-\int_0^t \frac{k_\tau}{a}tds}\ dt \\
&= \int_0^a \frac{k_\varepsilon}{a}t \quad e^{-\frac{k_\tau}{2a}t^2}\ dt
\end{aligned}$$

And for the second ray,

$$\begin{aligned}
I_2(2a) &= \int_0^{2a} \varepsilon_2(t)\ e^{-\int_0^t \tau_2(s)ds}\ dt \\
&= \int_0^{2a} \frac{k_\varepsilon}{2a}t \quad e^{-\int_0^t \frac{k_\tau}{2a}tds}\ dt \\
&= \int_0^{2a} \frac{k_\varepsilon}{2a}t \quad e^{-\frac{k_\tau}{4a}t^2}\ dt
\end{aligned}$$

Letting $u = \frac{t}{2}$, we change the variable of integration:

$$\begin{aligned}
\Rightarrow \qquad I_2(2a) &= \int_0^a \frac{k_\varepsilon}{2a}2u \quad e^{-\frac{k_\tau}{4a}(2u)^2}\ 2du \\
&= \int_0^a 2\frac{k_\varepsilon}{a}u \quad e^{-\frac{k_\tau}{a}u^2}\ du \\
&= \int_0^a 2\left(\frac{k_\varepsilon}{a}u\right)\ \left(e^{-\frac{k_\tau}{2a}u^2}\right)^2\ du
\end{aligned}$$

Comparing $I_1(a)$ with $I_2(2a)$, we see that the light emitted is doubled, but the transparency is squared. When the transparency is high, such as in this case where the ray is just entering an opaque region, this means the second ray will be brighter than the first. The gain in emitted light outweighs the loss of transparency.

Thus we see that when we render an object where material transitions occur over a long distance, Lambertian illumination suffers. In scenes with abrupt transitions, this effect is less pronounced.

We shall then depart from Lambertian shading in the fashion described by Max [Max95]. To derive his optical model, Max modeled a density of spherical particles, each able to absorb light or reflect it in a Lambertian manner. As this is quite unrelated to the idea of Lambertian illumination as used in surface rendering, it is unsurprising that the results look different. We shall follow his method but introduce a surface oriented element to restore the expected appearance of isosurfaces.

3.6.2 A revised Lambertian optical model

Max [Max95] derived a differential equation for emission-absorption rendering based on an optical of a density of particles suspended in space. So:

$$\frac{dI}{dt} = \varepsilon(t) - \tau(t)I(t)$$

where $\varepsilon(t)$ is equivalent to Max's source term and $\tau(t)$ is the extiction coefficient.

We shall set $\varepsilon(t) = a(t) + d(t)(n \cdot l)(n \cdot v)$, where n is the normalised gradient, l is the unit vector to the light source, v is the unit vector to the viewpoint, $a(t)$ is the amount of ambient light and $d(t)$ is the amount of diffuse shading. We have introduced the $n \cdot v$ term, which is unusual for Lambertian shading, to overcome the flat shading effect discussed in the previous section. The length a ray traverses through a material transition is approximately inversely proportional to $n \cdot v$, ignoring curvature in the material transition, so this additional term approximately corrects the resulting error.

We can apply Max's solution [Max95, p. 101] to the differential equation to get equation (3.1).

3.6.3 A comparison

The revised optical model was compared with the standard Lambertian optical model. Two scenes were rendered using each model, the first being the sphere referred to above and the second being the CT scan of a hip bone from page 1.3. The first has gradual material transitions such as the revised model was designed for, the second had abrupt transitions. The results are displayed in Colour Plates II.a–II.d.

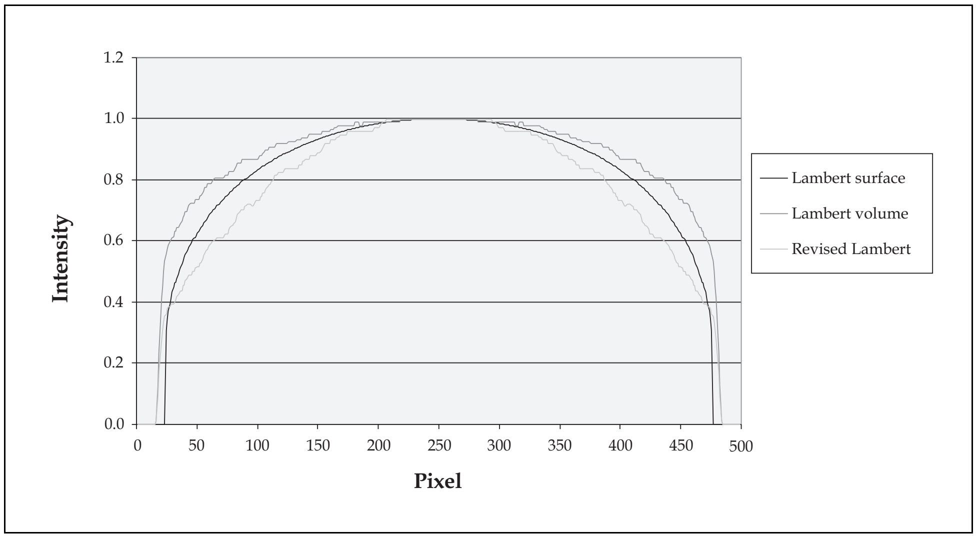

FIGURE 3.7: Standard vs revised Lambertian illumination

We can see that the sphere is improved in appearance — it does not look flat when the revised method was used. While the hip bone scene appears somewhat different, neither model has a clear advantage over the other.

Figure 3.7 shows the middle scan line from the revised image of the sphere superimposed on figure 3.5. We see that the revised model actually overcorrects in terms of meeting the expectation of a Lambertian surface in a volume renderer. The error of each method is approximately equal, but in opposite directions. We have used the revised model for most images in this thesis as it seems more sensible to exaggerate shading effects than omit them. Further work in this area is indicated to develop an optical model that makes a compromise between both methods. Such a model would give shading as realistic as surface Lambertian shading, and could be extended to include specular effects.

3.6.4 Surface strength shading

Several authors have used the "strength" of a surface to weight the illumination function, as was discussed on page 16. The usual measure of strength is the magnitude of the density gradient. The advantage of this is that abrupt transitions between materials will stand out clearly, while gradual changes are deemphasized. Apart from conveying more information to the user about the type of surface being viewed, this should also reduce noise.

However, as section 3.5.2 pointed out, gradient reconstruction is more difficult than density reconstruction. Examples of weighted shading are shown in Colour Plates III.a–III.d, where the sphere and hip bone scenes are rendered with and without it. The weighted shading images are clearly inferior, as the magnitude of the gradient reflects deficiencies in reconstruction more than the strength of the surface. The filter used is trilinearly interpolated central differences, described on page 51. The reader will note that this filter does produce a smooth interpolant, and that the sphere is well sampled with a wide Gaussian filter, and thus easy to reconstruct. We can only conclude that weighting shading in this manner greatly exaggerates reconstruction problems.

Possible solutions to this include using a more sophisticated surface detection algorithm, or using very high quality gradient reconstruction filters. A third possibility that does not involve a substantial computation cost is, instead of using a raw $|\nabla\rho|$, manipulate it using a sigmoid function $f(|\nabla\rho|)$. Some degree of weighting would still be present, but the large fluctuations in the magnitude that produces artifacts in the images would be eliminated.

However, an exploration of these is beyond the scope of this thesis and surface strength weighting is not used.

3.7 High Order Integration

3.7.1 Standard compositing raycasters

To use the emission-absorption optical model, we must evaluate the integral:

$$\text{Intensity along a ray} = \int_0^{t_{max}} \varepsilon(t) e^{-\int_0^t \tau(s)ds} dt \quad + \quad I_b e^{-\int_0^{t_{max}} \tau(s)ds}$$

where $t = t_{max}$ is the point where the ray exits the volume, the camera is positioned at the point $t = 0$ and I_b is the intensity of the background light entering the volume. For this thesis, we shall assume this is zero, i.e. that the object rests in front of a black background.

Most ray casters use the following method to evaluate the line integral. They take a number of RGBA samples and composite them using the Porter-Duff **over** operator, given in [Nov93, p. 44] as:

$$A \textbf{ over } B \quad = \quad (A_c + A_t B_c,\ A_t B_t)$$

where the RGBA samples A and B are treated as pairs of colour $\left(A_c, B_c \in \mathbb{R}^3\right)$ and transparency $(A_t, B_t \in \mathbb{R})$. We shall discuss this method assuming the emission-absorption optical model used by this thesis. As the rendering integral deals with scalar emission and extinction parameters, colour values are usually found by repeating the calculation three times at different frequencies. Most researchers leave extinction constant over different frequencies, hence the use of RGBA samples.

We shall introduce for convenience the notation:

$$I(a,b) = \int_a^b \varepsilon(t)\, e^{-\int_a^b \tau(s)ds} dt \quad + \quad I_b\, e^{-\int_a^b \tau(s)ds}$$

making $I(a,b)$ the ray intensity at the point $t = a$. The use of the background term is no longer trivial, as I_b is usually nonzero when $b < t_{max}$.

The intensity $A_i \in \mathbb{R}$ and transparency $A_t \in \mathbb{R}$ of a sample at a given frequency are calculated using the following approximations.[1] A sample (A_i, A_t) at $t = t_0$ represents $I(t_0, t_0 + \Delta t)$. The emission coefficient ε and extinction coefficient τ are assumed to be constant over the integration interval. Then:

$$I(t_0, t_0 + \Delta t) = \int_{t_0}^{t_0+\Delta t} \varepsilon\, e^{-\int_{t_0}^{t} \tau ds} dt \quad + \quad I_b\, e^{-\int_{t_0}^{t_0+\Delta t} \tau dt}$$

$$\begin{array}{lrl}
\text{We have} & A_t &= e^{-\int_{t_0}^{t_0+\Delta t} \tau dt} \\
\Rightarrow & &= e^{-\tau\Delta t} \\
\text{and} & A_i &= \int_{t_0}^{t_0+\Delta t} \varepsilon\, e^{-\int_{t_0}^{t} \tau ds} dt \\
 & &\approx \int_{t_0}^{t_0+\Delta t} \varepsilon\, e^{-\tau\Delta t} \\
\Rightarrow & &= \varepsilon\, \Delta t\, e^{-\tau\Delta t}
\end{array}$$

We see there are two approximations made by such ray casters. Firstly, as was seen above, the ε and τ coefficients of the material are assumed to be constant, whereas in fact they vary with density. The second approximation is made when using $A_t = e^{-\tau\Delta t}$ as the inner integral for calculating A_i. This has the effect of placing all the opacity of the sample "in front" of the emission, thus underestimating A_i. One of the purposes of this thesis is to tackle both of these approximations.

We note that the use of the **over** operator itself does not involve any numerical approximations; it is relatively simple to show the rendering integral can be partitioned into intervals and solved piecewise in this fashion. The approximations occur when taking samples of intensity and transparency. In the remainder of this chapter, we shall firstly give a formula for correctly calculating samples assuming piecewise constant ε and τ, and then we shall extend this formula to include piecewise linear $\tau(t)$ and then piecewise linear $\varepsilon(t)$. We shall then discuss the use of these formulae in the renderer.

[1]Exceptions include Williams, Max and Stein [WMS98], Novins [Nov93] and Upson & Keeler [UK88], all of whom use more sophisticated methods.

3.7.2 Solutions to the rendering integral

For simplicity, we will assume that the interval to be integrated is $[0, x]$. A problem using the interval $[a, b]$ can easily be reduced to a problem of this form.

Case 1.

We shall first consider the case when ε and τ are assumed to be constant. We shall eliminate the second error alluded to above concerning the absorption of light within a sample. Let c_τ be the absorption coefficient, and c_ε be the emission coefficient. We have:

$$\int_0^x c_\varepsilon e^{-\int_0^t c_\tau ds} dt \quad + \quad I_b e^{-\int_0^x c_\tau ds} \quad = \quad \frac{c_\varepsilon}{c_\tau} + \left(I_b - \frac{c_\varepsilon}{c_\tau}\right) e^{-x c_\tau} \tag{3.2}$$

The proof of this statement is in Appendix A.1.

Case 2.

For the next case, we shall generalise the emission function from piecewise constant to piecewise linear. Let $\varepsilon(t) = m_\varepsilon t + c_\varepsilon$. Then:

$$\int_0^x (m_\varepsilon t + c_\varepsilon) e^{-\int_0^t c_\tau ds} dt \quad + \quad I_b e^{-\int_0^x c_\tau ds} \quad = $$
$$\frac{c_\varepsilon}{c_\tau} + \frac{m_\varepsilon}{c_\tau^2} + e^{-c_\tau x} \left(I_b - \frac{c_\varepsilon}{c_\tau} - \frac{m_\varepsilon}{c_\tau^2}(c_\tau x + 1)\right) \tag{3.3}$$

The proof is in Appendix A.2.

Case 3.

Finally, we shall let the extinction be piecewise linear, so $\tau(t) = m_\tau t + c_\tau$. So:

$$\int_0^x (m_\varepsilon t + c_\varepsilon)\, e^{-\int_0^t m_\tau s + c_\tau ds}\, dt \quad + \quad I_b e^{-\int_0^x m_\tau s + c_\tau ds} \quad = $$
$$\sqrt{\frac{\pi}{2m_\tau^3}}\, (c_\varepsilon m_\tau - c_\tau m_\varepsilon)\, E \; + \; \frac{m_\varepsilon}{m_\tau} \; + \; e^{-x\left(c_\tau + \frac{1}{2} x m_\tau\right)} \left(I_b - \frac{m_\varepsilon}{m_\tau}\right) \tag{3.4}$$

where

$$E = e^{\frac{c_\tau^2}{2m_\tau}} \left(\operatorname{erf}\left(\frac{c_\tau}{\sqrt{2m_\tau}} \right) - \operatorname{erf}\left(\frac{c_\tau + x m_\tau}{\sqrt{2m_\tau}} \right) \right)$$

The proof is in Appendix A.3.

3.7.3 The use of these formulae in the renderer

In case 1, equation (3.2) took constant parameters. This equation can be used for processing cells with constant density in the renderer as such cells have constant material properties and zero gradient, so the parameters do not vary.

For cells with varying density, emission and extinction are not constant. Due to the non-linear nature of the illumination function and of trilinear interpolation, the emission and extinction functions are complicated. Extinction is a cubic function and emission is a polynomial of degree seven if we assume the first method of gradient calculation in section 3.5.2, and even higher if the other methods are used. As no solution to the rendering integral is known for such complicated parameters, some approximation must be used.

While a good approximation would be to assume both emission and absorption are piecewise linear and use equation (3.4), this has not been implemented due to the complexity of the formula. The possibility is raised as future work in section 5.2. A promising technique is to use equation (3.3), and subdivide the integral further to reduce the error caused by this approximation. Only a small number of subdivisions turn out to be necessary to give good results. The number three was used in our renderer, but the advantages of an adaptive scheme are explored in section 5.2.

3.8 Time Complexity

We shall now derive the time complexity of the renderer, comparing the octree optimisation with a basic cell by cell traversing renderer. There are two significant steps to generating an image: preprocessing and rendering. We shall assume the dataset is cubical and the image square, for simplicity. Let n be the sidelength of the volume in voxels, and m be the sidelength of the image in pixels.

3.8.1 The cell by cell renderer

The preprocessing step for the cell-by-cell renderer, without the octree, simply calculates whether a cell has variable or constant density. This information is used to decide what integration code to use when tracing rays. The relevant pseudocode is the code fragment labeled `Homogeneity` on page 41.

As the body of the main loop runs in constant time, and as there are $(n-1)^3$ cells, the preprocessing step is obviously $O(n^3)$. The rendering step is equally easy to calculate. m^2 rays are cast into the volume. For each of these, the number of integrations, reconstructions and so forth is directly related to how many cells each ray traverses. This is proportional to the sidelength of the volume, n, as each ray passes linearly through it.

The time complexity of the cell-by-cell renderer is thus the complexity of the preprocessing step plus the number of rays times the complexity for each one, or $O(n^3 + nm^2)$.

3.8.2 The octree renderer preprocessing step

With the octree optimisation, the preprocessing step includes a great deal more calculation. Firstly, the homogeneity calculation described above is performed. Secondly the program iterates through each level of the volume pyramid, setting the octree node information. The pseudocode is on page 41.

It is clear that the number of times the inner loop is executed is equal to the number of cells in the volume pyramid, which is a geometric sum related to the number of cells in the original volume. Let l be the number of levels in the pyramid.

$$\begin{aligned}\text{Number of cells } &= n^3 + \frac{n^3}{8} + \frac{n^3}{8^2} + \cdots + \frac{n^3}{8^{l-1}} \\ &= \frac{n^3\left(1 - 8^{l-1}\right)}{1 - \frac{1}{8}} \\ &= \frac{8}{7}n^3\left(1 - 8^{l-1}\right)\end{aligned}$$

Let the time taken to execute the inner loop be c. As the heights of pyramids in practical situations is above 7 or 8, the quantity in brackets is closely bounded by 1:

$$\text{Total time} \approx \frac{8}{7}cn^3$$

The octree node calculation is thus $O(n^3)$. As the homogeneity and octree node calculations are both $O(n^3)$, the preprocessing step as a whole is $O(n^3)$.

The octree rendering step

The time complexity of the rendering step is more difficult to calculate. As before, the number of rays cast is m^2, but the number of steps involved in integrating a single ray is not well defined. If the dataset has constant density, the octree will be traversed in one step, making the complexity of the rendering step $O(m^2)$; if the dataset is totally incoherent, the octree will be traversed in $O(n)$ steps, making the complexity $O(nm^2)$.

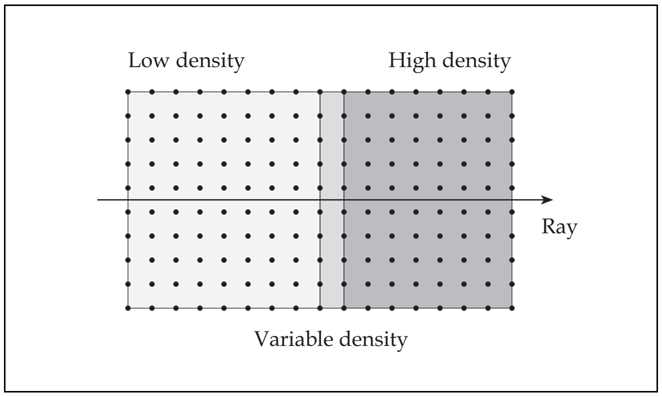

FIGURE 3.8: A simple octree

We must therefore make an assumption about the coherency of the dataset. Medical datasets, after classification, consist of regions of constant density corresponding to various tissue types, separated by layers of varying density. We shall choose a simple dataset of this form, derive the time complexity of the octree traversal, and take this as an approximation to the general case the renderer is intended to handle. Consider an axis-aligned ray passing through a volume consisting of two cuboidal regions of different density. For simplicity, we shall assume simple point sampling and trilinear reconstruction so the region of varying density lies between two voxels, as in figure 3.8.

We shall prove by induction that for this volume the time complexity of octree traversal is $O(\log n)$, wherever in the volume the material transition occurs. Consider a volume of thickness $n_1 = 2^1$ voxels. Obviously, the ray traverses this volume in 1 step as it is one cell thick.

Assume that a volume of thickness $n_k = 2^k$ voxels is traversed in k steps. If we add a layer of voxels 2^k thick to one or other side of the volume, with the density of the material on that side of the volume, we now have a volume of thickness $n_{k+1} = 2^{k+1}$ that satisfies the coherency assumption. As the new layer of cells is of constant density, has a thickness that is a power of two and is adjacent to the edge it will be collapsed into one octree leaf. Thus it will be traversed by the ray in one step. The number of steps in the total octree traversal is thus $k + 1$, and the proof is complete. The volume of size 2^i is traversed in i steps, thus the traversal is $O(\log n)$. The complexity of the rendering step is thus $O(m^2 \log n)$.

If, however, the coherency assumption is not met, as was stated above a ray will take $O(n)$ steps to traverse the volume, so the complexity of the rendering step is $O(nm^2)$.

As the preprocessing step is $O(n^3)$, and the rendering step $O(m^2 \log n)$, the octree renderer has complexity $O(n^3 + m^2 \log n)$ when the octree is operating on a coherent volume and $O(n^3 + nm^2)$ when it is not.

3.9 Summary

This thesis has advanced a renderer design that has the following features.

- It uses a piecewise linear, probabilistic classification method that resists noise and drift, but supports an intuitive way to specify material properties,
- It uses the low-albedo emission-absorption optical model, with a revised Lambertian shading function that performs well when material transitions are gradual,
- It is an octree accelerated ray caster. The octree is stored with only a modest increase in memory, and traversed using a fast method based on Glassner [Gla84],
- It uses a higher order approximation for evaluating the rendering integral than is found in most other research. This reduces dramatically the number of steps required to accurately perform integration.

Therefore, the renderer meets the requirements listed on page 33 to explore high-quality direct volume rendering.

Chapter 4

Results

4.1 General Discussion

To explore high quality rendering using desktop computers, the renderer was implemented for Microsoft's Windows NT 4.0 operating system. Test images were generated using a Pentium 200MMX machine with 64Mb of RAM, demonstrating the practicality of volume ray casting on what is by current standards a modest computing resource. A more powerful 400MHz Pentium II machine with 128Mb RAM was used to generate several images for which 64Mb was not sufficient.

We shall firstly describe the data sets used to generate results. Next, the effect of various choices in constructing the renderer will be discussed, noting how they affect quality and execution time. Lastly we shall conclude by outlining future directions of research and summarising the knowledge gained by this research.

Test data sets include a CT scan of a male pelvis, with the original resolution $256 \times 256 \times 56$ voxels, resampled to $256 \times 256 \times 111$ to yield a unit aspect ratio. This data set is able to be reliably classified into fat, muscle and bone tissue types, but suffers from irregularities around the subject's stomach, as the subject was breathing during the scan. Adjacent slices are thus slightly out of alignment depending on the point in the subject's breathing cycle at which the slice was measured.

A data set has also been extracted from the Visible Woman archive [PBB98], with resolution $256 \times 256 \times 256$ voxels. The Visible Woman data set was extracted from the photographic images in the archive. The archive consists of 5189 slices, each a full colour photograph scanned at a resolution of 1664×1216, giving a huge volume about 30Gb in size. The volume has a constant aspect ratio, each cell being a cube $\frac{1}{3}$mm on a side. To extract a data set useful for the purposes of this renderer, images from a subset of the archive were mapped to a grey scale using a function designed to distinguish body tissue (yellow and red) from the surrounding area (blue and black). The grey scale images were then cropped and concatenated to create a large 768^3 volume containing the Visible Woman's head and neck. This was then downsampled to the required size of 256^3 using a Gaussian filter. The result is a well-sampled data set containing a large amount of facial detail. The face contains some scarring due to the process of data collection, where each slice was manipulated during photography.

The above two data sets represent the practical application of the renderer using human-derived data with complex structure. Of theoretical value, however, are two artificial data sets demonstrating that the renderer produces verifiably correct results. The first is a $120 \times 120 \times 120$ volume representing a sphere of high density sampled with a Gaussian filter of coarse radius. This data set was designed to demonstrate that the shading model produces the appearance of realistic surfaces when operating on a slow (six cell lengths) transition between materials. This requirement is important if the render is to meet the expectations of users, as the human visual system is designed to visualise surfaces and can extract more information from them than from blurry fog or mist.

The fourth data set is adapted from [ML94] and explores the accuracy of the reconstruction in the renderer. This $32 \times 32 \times 32$ data set was designed to contain large density fluctuations near the Nyquist limit, thus producing the worst case data that can in theory be accurately reconstructed. The function used to generate this data set is:

$$\rho(x, y, z) = \frac{\big(1 - \sin\frac{\pi z}{2}\big) + \alpha\big(1 + \rho_r\sqrt{x^2 + y^2}\big)}{2(1 + \alpha)}$$

where

$$\rho_r(r) = \cos\Big(2\pi f_M \cos\frac{\pi r}{2}\Big).$$

The function was sampled in the interval $x, y, z \in [-1, 1]$, using parameters $f_M = 6$ and $\alpha = 0.25$. The authors claim that approximately 99.8% of the energy of the signal lies below 10 cycles per unit length, which is the Nyquist frequency as samples are 0.05 apart. The function is therefore able to be reconstructed with great accuracy using perfect filters, making it a perfect test for filters of lesser quality.

4.2 Results

4.2.1 Integration

One important question raised by this thesis is whether the effort of calculating the volume rendering integral with a higher order of accuracy results in a gain in image quality. Most volume ray tracers examined use a simpler method where the transparency and the emission are used to calculate the "effective colour" which is then composited directly onto the ray's brightness, as was discussed in section 3.7.1. While both methods tend to the correct answer in the limit, in practice renderers cannot use an infinitesimal step size and we must ask which method yields the least error for practical amounts of computation.

The renderer using the higher order integration formula evaluated a cell-by-cell traversal of the volume, partitioning each ray into segments that are wholly contained by a single cell. It did not use the octree optimisation, which will be discussed below. A distinction was made between cells of constant density and cells of variable density, and appropriate integration code employed for each. Constant cells were integrated in a single step using a simple formula, while variable cells were integrated in three steps using the formula able to handle linear emission functions described in section 3.7.2. The subdivision into three steps was found to be sufficient to overcome the constant attenuation approximation that is still required by the formula in most cases. The pelvis scene discussed below is an exception to this, and a more elegant solution to this problem is presented as future work in section 5.2.2. Volumes with large homogeneous regions were thus rendered faster than equal-sized heterogeneous volumes.

Five scenes were rendered using both techniques. The images are displayed in Colour Plates IV and V. The first two scenes use the pelvis test data set. The first, called the hip bone scene in the figures, uses a classification scheme that distinguishes bone from other tissue, resulting in a volume with large homogeneous regions. The second scene, the pelvis, uses a classification scheme that distinguishes a number of tissue types: skin/fat, muscle and bone. The third scene, called head, is of the Visible Woman's head. The fourth, sphere, is of the Gaussian smoothed sphere and the last, ripples, is of the function taken from [ML94].

The classification parameters used are contained in the attached CD in the directory `\materials`. The format for these files is an easily readable text file described in Appendix B.

To measure image quality, a set of comparison images were generated using the constant step renderer with a step size of 0.1. At this step size, integration errors are negligible and the images can be regarded as a standard to measure the two techniques against. The root mean square error was taken between each test image and the reference image. It was measured as follows:

$$\text{RMS error} \quad = \quad \sqrt{\sum_{\text{all pixels } i} \quad \sum_{\lambda \in \{r,g,b\}} \Big(A_\lambda(i) - B_\lambda(i) \Big)^2}$$

where the colour of pixel i in image I is $\big(I_r(i), I_g(i), I_b(i)\big)$, A is the reference image, and B is the test image. This was then normalised against triple the number of pixels, to give an average error per channel per pixel. The results are given in figure 4.1 as fractions of a gray level. To verify that errors were indeed negligible in the reference images, the pelvis scene was rendered by the constant step renderer at a stepsize of 0.05 and compared with the pelvis reference image. The result was an error of 0.0013, many times less than the difference between the reference image and the high order integration renderer. This value indeed substantially smaller than the lowest error reported in the graph.

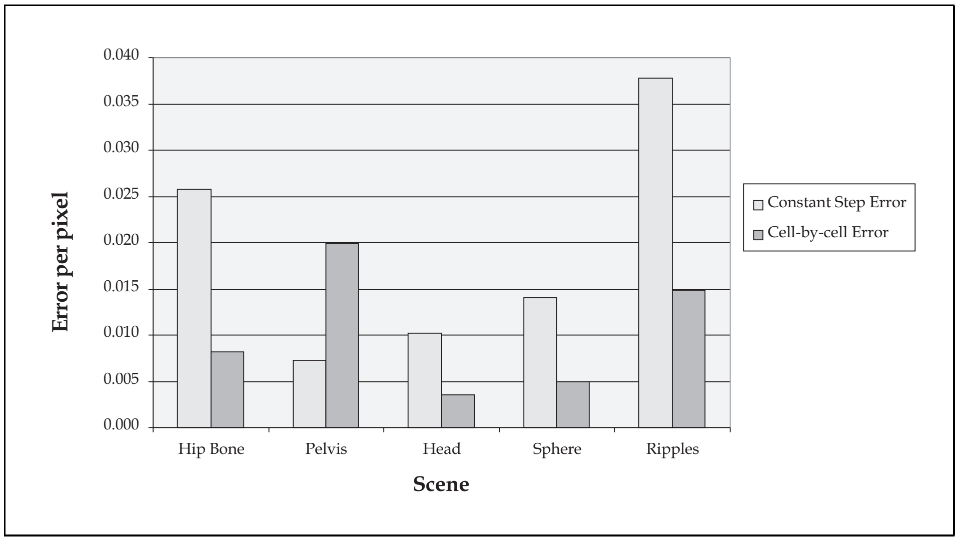

FIGURE 4.1: Root mean square errors

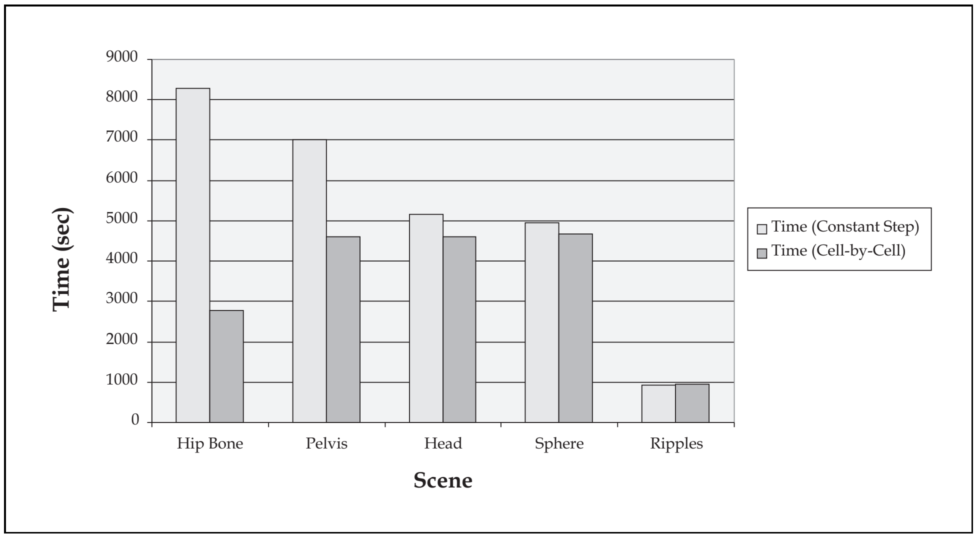

FIGURE 4.2: Constant step vs integration

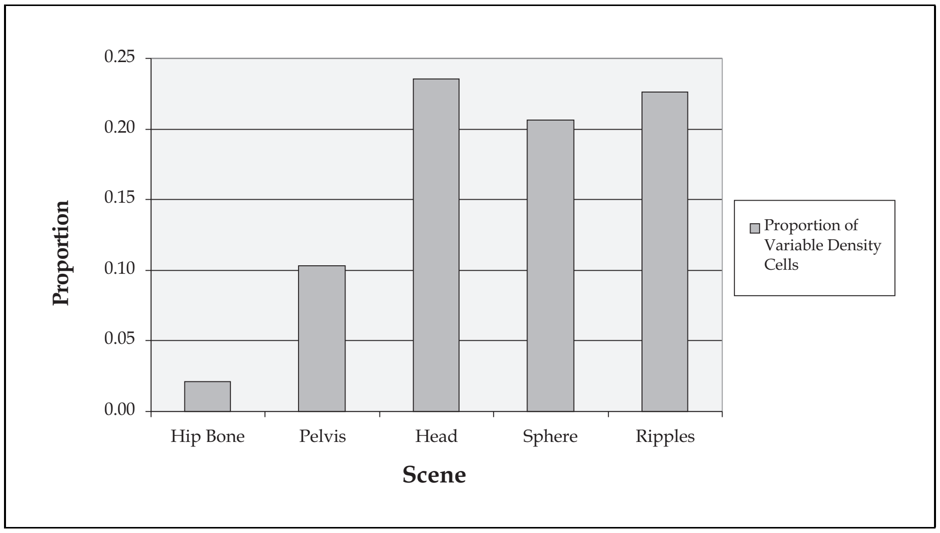

FIGURE 4.3: Proportion of variable density cells

The results show that the cell-by-cell renderer with high order integration in general outperforms the constant step compositing renderer in both speed and accuracy. The step size used to generate test images for the second renderer was 0.4 of a cell width. A higher value may speed it up to the level of the high order renderer, with a loss in quality, or *vice versa*, a lower value will give it the same quality but degrade performance. The single exception to this trend is the second scene, the pelvis with muscle and skin / fat classification. Here, the image quality of the high order renderer is reduced due to the sharpness of the classification parameters. As material properties are changing quickly in some regions of the volume, the approximation due to constant attenuation referred to above becomes inadequate.

We can see from figure 4.2 that with the hip bone scene the high order renderer outperformed the constant step renderer by a large margin. The other data sets contain a higher proportion of variable density cells as indicated in figure 4.3, so for those scenes the performance of the two renderers were equivalent.

4.2.2 Gradient calculation

As [MMMY96, p. 12] notes, gradient reconstruction is more important than density reconstruction for image quality. Three different schemes for calculating the gradient of the density field have been examined in this thesis, in section 3.5.2. The first is to simply differentiate the reconstruction of the density field directly. As trilinear interpolation is used, the result is a simple quadratic function that is quick to evaluate. However, it is not continuous at cell boundaries and degrades image quality. The second method is to calculate a central differences gradient at each vertex, and trilinearly interpolate those values to obtain a gradient at any point in the volume. This method gives a result that is continuous at cell boundaries, but requires more calculation as vertex gradients are not stored and re-used due to memory considerations. The last method tested also uses central differences, but

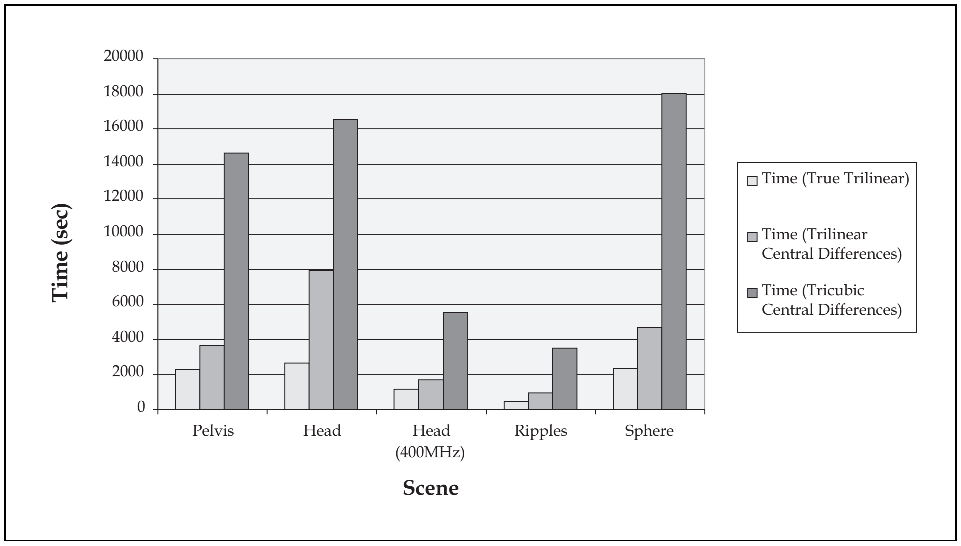

FIGURE 4.4: Gradient reconstruction

uses B-spline tricubic interpolation instead of trilinear. This method has a higher order of accuracy, and should have produced smoother shading on surfaces.

From figure 4.4, we see that the difference in execution time is large. A high proportion of the execution time is spent calculating gradients, especially for the more accurate methods. The generated images are on Colour Plates VI and VII, and it is evident that the true trilinear gradient is insufficiently smooth for high quality image generation. However, the other two, despite the large difference in speed, produce roughly equivalent images. While the pelvis and to a smaller extent the Visible Woman show some minor improvement, the additional calculation time could be better spent elsewhere.

We can conclude that trilinearly interpolated central differences offers good accuracy for properly sampled data, and that the tricubic method offers too little return for the additional rendering time.

4.2.3 Octree optimisations and rendering time

It is evident from the execution times of the test images that the renderer is not fast. There are a number of reasons why the code lacks efficiency. Firstly, the code has not been optimised and could be significantly accelerated by standard methods. Secondly, some of the Microsoft Visual C++ compiler options adversely affect program correctness and have been switched off. While they produce faster code, they introduce voxellated artifacts into the image. This has reduced speed by approximately 30–40%.

In section 3.8 we predicted that the octree optimisation will increase efficiency for sufficiently large volumes. To evaluate the effectiveness of the use of octrees, a number of images were rendered at a variety of image and volume resolutions,

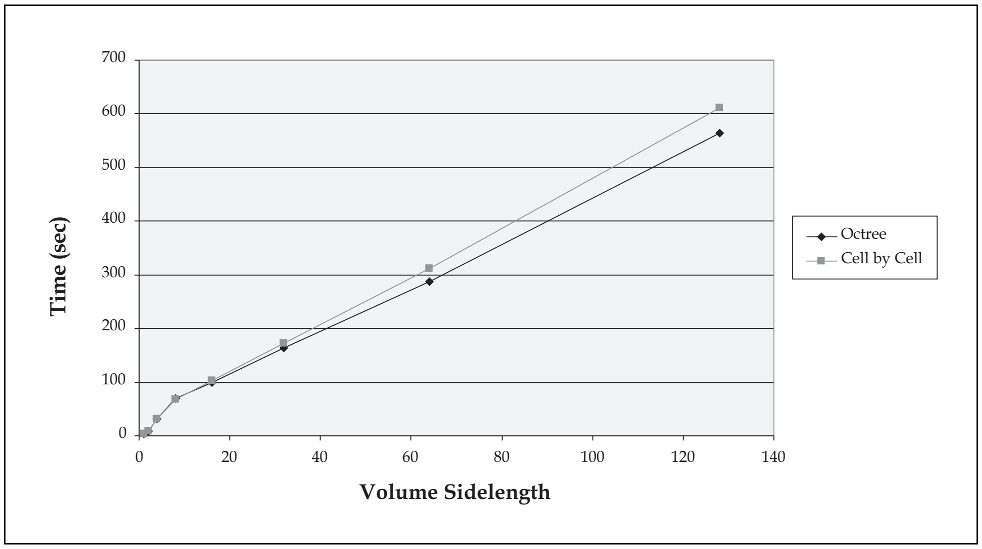

FIGURE 4.5: Rendering time per volume sidelength for the Visible Woman

using high order integration and trilinearly interpolated central differences gradient calculation. The shading model was revised Lambertian without surface strength weighting, as described in section 3.6.

Figure 4.5 shows the impact of volume size on rendering time for the Visible Woman data set, which was resampled at a range of resolutions. The reader will note from figure 4.8 that the octree optimisation does not work particularly well for this volume. Therefore, as was discussed in section 3.8 we expect a time complexity of $O(n^3 + nm^2)$ for both the octree and cell-by-cell renderers, where n is volume sidelength and m is image sidelength. This is due to the high number of leaves in the octree, as the classification scheme is quite smooth. In fact, figure 4.5 shows that both renderers faired equally well for these data sets.

As the $O(n^3)$ preprocessing figure is small compared to the overall render time, both graphs are expected to follow a linear curve. Overall this is correct, but for each renderer there is a significant departure from a linear graph: The smallest volumes render faster than expected.

This phenomenon is explained by the memory hierarchy of the machine. The small volumes are of such a size that they fit entirely within the second level cache, and thus render quickly. As the volume size increases, however, the cache and then the main memory are filled. Rendering the larger volumes gives a very good linear fit, but the 256^3 volume caused the 64Mb machine these tests were conducted on to thrash, causing the CPU usage to drop below 30% for long periods of time. This test has been omitted from the graph, but the octree renderer fared slightly worse as octrees require more memory.

Figure 4.6 shows the times taken to render the hip bone scene (see Colour Plate VIII) with image resolutions from 10^2 to 1000^2. The expected curve given by complexity analysis is a quadratic function of sidelength, but the graph is given as a function

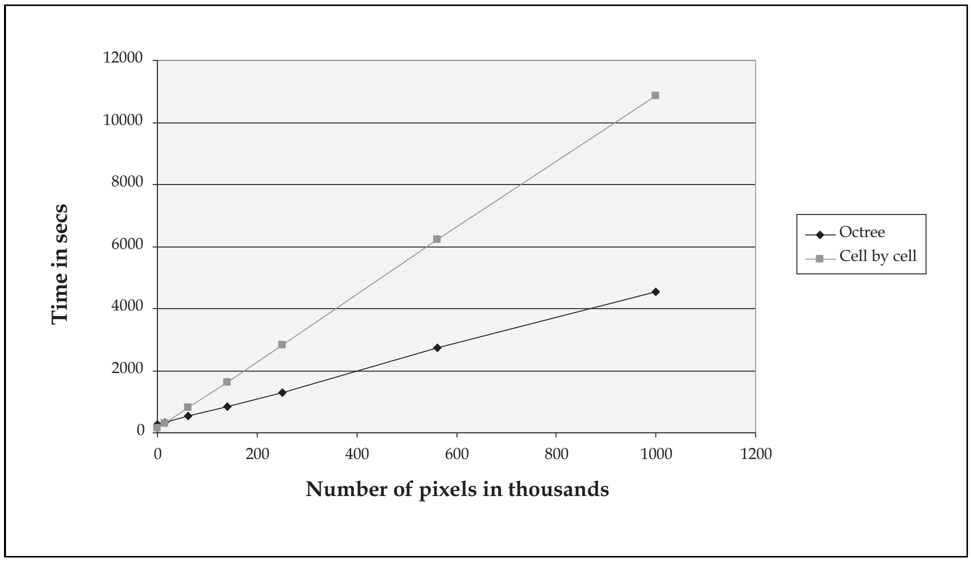

FIGURE 4.6: Time to render per image size

of the number of pixels. Curves of the expected form will thus have a linear shape, and we see curves are indeed straight lines. We also see that, as expected, the octree renderer has a longer preprocess time than the cell-by-cell renderer, but quickly overtakes it at an image sidelength of about 150, so that the octree renderer is faster for all practical image sizes.

We can see in figure 4.7 the proportion of total render time spent in the precalculation step: distinguishing constant from variable density cells in the cell-by-cell renderer or building the octree in the octree renderer. The Visible Woman when rendered on the 64Mb machine is the only scene that does not follow the trend that a higher proportion of time is spent in preprocessing for the octree; this is because the rendering time for that image was lengthened by the number of page faults during rendering. This is further demonstrated by figure 4.8, where it was the only scene to take longer when the octree was used. The same scene rendered on the 400MHz machine with 128Mb memory gives more meaningful data.

The hip bone scene shows the largest proportion of time spent building the octree. Figure 4.8 gives the reason for this: The octree optimisation is most effective for that data set. The overall rendering time was dramatically reduced, even though the preprocess time increased. The Sphere scene, however, shows that rendering was far more significant than preprocessing for either algorithm. The sphere data set has a large number of variable density cells in the thick material transition layer, which uses up the majority of the rendering time for either renderer.

We can see in figure 4.9 a measure of the effectiveness of the octree optimisation: the proportion of leaves in each data set compared with the worst case for a data set of similar size. The reason for the success of the optimisation with the hip bone scene is that it exploits the highly coherent nature of the data. Most of the volume, in fact, is empty space, as hip bones are thin, plate-like structures only several voxels thick at this resolution. In contrast, the artificial volumes and the Visible Woman have large

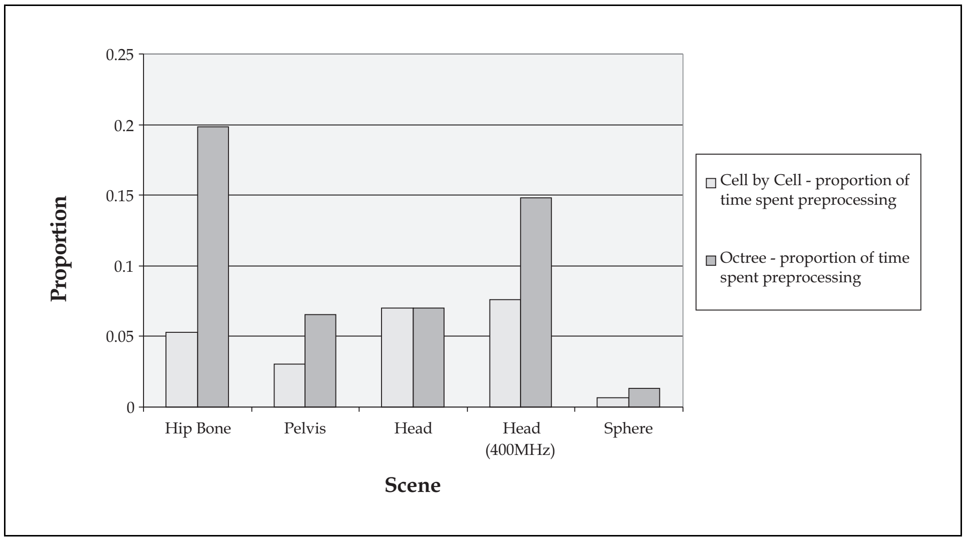

FIGURE 4.7: Time devoted to preprocessing for 500^2 pixel images

variable regions that decrease savings. The ripples data set is of course designed to have this feature; the other two have been sampled with a Gaussian filter with a heavy smoothing effect.

4.3 Summary

The use of high order integration methods has proved to be a success for this renderer. The high order integration method was tested against a normal constant step compositing renderer. It is possible to choose a step size for the latter method that allows double the error (averaged over all five scenes) but still does not perform as fast as the high order method.

Different methods of reconstructing density were surveyed. The best method is trilinearly interpolated central differences, which is not prohibitively expensive but does give good results.

The use of octrees does speed up rendering in all cases where the octree datastructure can fit in RAM. The size of the speed gain is modest for some data sets, as high as a factor of two for highly coherent data. This is achieved without any loss in image quality. The amount of acceleration depends on the number of leaves in the octree as compared to the number of cells. For the two real data sets used, the pelvis CT scan has only 2.6% as many leaves as cells when classified into air and bone, 18.1% when classified into air, skin / fat, muscle and bone. The Visible Woman data set has 29.0% as many leaves as cells, due to the smoother sampling of the data. This value is approaching the break-even point where both renderers are equally fast.

The time complexity of rendering both with and without octrees was verified, results agree with expectations except where the speed of different levels of the memory hierarchy of the machine must be taken into account.

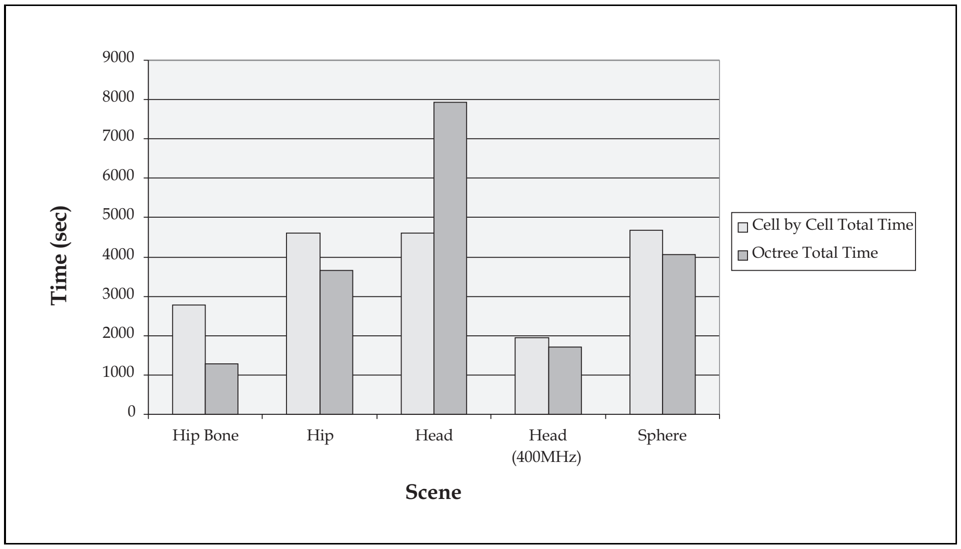

FIGURE 4.8: Cell by cell rendering vs Octrees

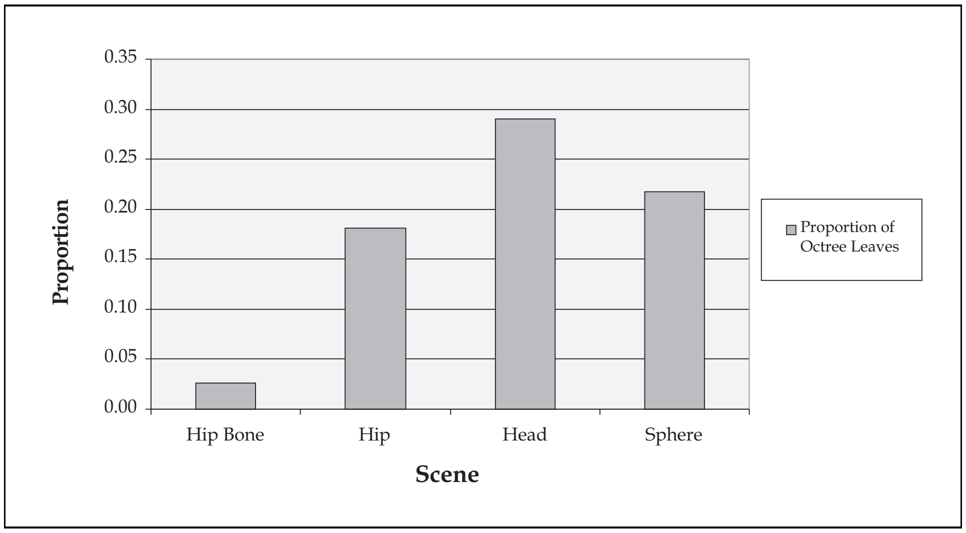

FIGURE 4.9: Savings in octree leaves

Chapter 5

Conclusions

5.1 Discussion

Research in volume rendering most often concentrates on acceleration techniques to render quickly with acceptable error. This thesis has concentrated instead on the quality of images, trying to maximise the useful information about the data set provided by each image. We have examined high quality rendering with the goal of producing an efficient renderer that will run on modest computer systems.

This thesis has explored the use of exact evaluation of the rendering integral as a higher quality alternative to compositing effective colour. This has been achieved by approximating the absorption and emission functions along the ray with piecewise constant and piecewise linear functions, respectively. Each ray is partitioned into segments in such a fashion that the error of this approximation is small. This has demonstrated advantages over the standard techniques that take the exact absorption and emission functions and perform numerical calculations to approximate the true integral: images are rendered faster and with lower error. Far fewer steps need to be taken by the renderer, even though each step requires more complex calculations.

A sensible method of partitioning rays into integrable intervals is with an octree, as the length of an interval depends on the homogeneity of the region. Excess calculation in regions of constant density is thus avoided. A pyramid volume representation of the octree was chosen over pointer representations for memory efficiency reasons. While the full tree is stored to the level of single cells, when only the leaves are used, the lack of pointers or similar housekeeping information more than outweigh the waste. This is due to the fact that unlike octrees as used elsewhere, only one byte-sized density value need be stored at each leaf, whereas a pointer is four bytes long. The octree representation used takes about 1.4 bytes per voxel to store, as compared to 5.6 bytes per leaf for a simple pointer representation.

Another advantage of the octree representation chosen is that data for a given cell can be extracted in constant time, unlike the more common recursive structures that require a tree search. This fact is taken advantage of by the octree traversal method chosen, adapted from Glassner in [Gla84], which traces rays by finding a point guaranteed to be in the next leaf, and querying the octree for leaf data based on the point's position.

The thesis also explored shading models. Lambertian shading was shown to have some defects when applied to volumes with smooth material transitions, so a refinement was developed to give isosurfaces within such volumes an improved appearance. Surface strength shading was also explored, where the gradient magnitude is used to measure the strength of shading. While this is useful to highlight sharp transitions between tissue types, it exaggerates problems with interpolation and is useful only for smoothly sampled data sets.

Three different ways of calculating gradients were compared: the gradient of the trilinear interpolated field, and central differences calculated at voxels with trilinear and with tricubic interpolation. Trilinearly interpolated central differences is the favoured method. Using the true trilinear gradient is fast and is elegantly consistent with the interpolated density field, but the result is not continuous at cell boundaries. This results in a blocky appearance to surfaces. Tricubic B-spline interpolation of the central differences gradient should in theory give very smooth surfaces, but the result is rarely distinguishable from the trilinear central differences method and slows rendering by a factor of two or three.

In conclusion, the result is that the images produced by the renderer are smooth and show fine detail, at least for well sampled data sets. The renderer runs within the memory constraints of a 64Mb system quite well. Only for the largest data set was more RAM desirable. At the time of writing, 128Mb machines are easily within a modest budget and can easily handle volumes 16Mb in size. The secondary goal of speed has not been met, the renderer can take of the order of an hour for many scenes of reasonable complexity.

This initial survey should provide a platform for further research in high quality volume rendering, and shown that quality need not be compromised to be practical as an ordinary desktop application.

5.2 Future Directions of Research

This thesis has explored high quality volume rendering, and describes a renderer that uses high quality techniques on ordinary desktop computers. One attribute the renderer lacks, however, is execution speed. There are a number of avenues that can be explored to address this, including other octree optimisations and a cache data-structure to store reusable results of calculation. Other areas for further work consist of ways to increase the quality of the rendering, including efficient high-quality interpolation and integration code that handles a more general set of emission and absorption transfer functions.

5.2.1 Increasing speed

Section 2.5.3 described how octree representations of data are useful for more than fast traversal of homogeneous regions. They can be used to find approximate ray traversals satisfying error metrics that optimise dim and near-homogeneous regions. This is consistent with the goal of high quality rendering, even though eliminating approximation is the primary motivating force. Techniques exist that can increase the quality of images further, as will be described below, but require greater computation time. Optimising rendering for regions of the volume which are not seen would make tricubic interpolation, for example, a practical technique.

As [DH92] describes, an error metric weighted for bright, quickly varying regions of the volume can save a large amount of calculation without significant loss of quality. Speed increases of several hundred percent have been reported, but a high quality renderer would need to have strict error tolerances so savings would not be likely to reach this. However, significant gains are still possible before errors reach a single grey value.

The volume pyramid representation of the octree that has been used for this renderer also makes it possible to adaptively traverse the volume at any level of detail. [DH92] uses this fact to eliminate excess calculation when ray opacity is high.

The current implementation of the renderer makes little use of ray coherency. Consecutive rays pass through similar regions of the data set, but recalculate quantities such as density gradient from scratch. A datastructure can be devised to save recent calculations for future use, similar to caching systems in modern hardware. Such a datastructure is easily adapted from a queue, and has the advantage that the size can be adjusted depending on the available memory, so that excess memory on large systems can be used to accelerate ray tracing, without precluding the use of small systems.

In high resolution images, many rays pass through each voxel, and like many renderers reconstruction consumes a large proportion of computing time. The speed advantages of use of such a datastructure are likely to be significant.

One last question to do with execution speed remains to be asked. The number of steps required to evaluate the rendering integral accurately is far smaller using high order methods. In this renderer each step is evaluated without optimised code or hardware acceleration, and the reduction in the number of steps outweighs the greater cost of each step. Are the high order methods explored here as able to take advantage of optimisation and hardware acceleration as present techniques?

5.2.2 Increasing quality

This thesis has explored the use of different filters for calculating the density gradient, but has only used standard trilinear interpolation for density itself. While very high quality reconstruction such as windowed sinc and Gaussian filters are unrealistic for use with reasonably sized volumes, tricubic filters yield smooth density fields with a smaller increase in computation time. Such filters would be useful for creating high quality images from badly sampled data, where sharp density fluctuations degrade image quality. However, tricubic filters have an extent of three cell widths instead of one, reading 64 samples not 8. Such an extension would therefore have a large impact on rendering time.

The current integration code handles piecewise linear emission functions and piecewise constant absorption. Ray paths must be partitioned finely to avoid errors due to the restriction on the attenuation function, which is overcome in the present renderer by dividing cells with varying density into three segments. One useful extension to the renderer would be an adaptive mechanism to use as many segments as necessary, which may in general also increase speed as the value three was chosen to handle extreme cases.

However, it is possible to find exact formulae for the rendering integral when this is linear. An area for further investigation is whether image quality is gained more efficiently by using more sophisticated integration formulae or by using finer partitions. One would expect, since the exponentiations and error function evaluations can be implemented as table lookups, that this would be the most effective, as well as being the most elegant.

It is possible for the rendering integral to be solved when emission is a piecewise quadratic function, but the solution is complex in the extreme and unlikely to provide a useful area of research.

Appendix A

Proofs

The proofs of the three results in section 3.7.2 are now presented.

A.1 Constant Emission and Constant Extinction

$$\int_0^x c_\varepsilon e^{-\int_0^t c_\tau ds} dt \;+\; I_b e^{-\int_0^x c_\tau ds} = \frac{c_\varepsilon}{c_\tau} \;+\; \left(I_b - \frac{c_\varepsilon}{c_\tau}\right) e^{-xc_\tau}$$

Proof. Firstly,

$$e^{-\int_0^t c_\tau ds} = e^{-tc_\tau}$$

Therefore,

$$\begin{aligned}
\int_0^x c_\varepsilon e^{-\int_0^t c_\tau ds} dt \;+\; I_b e^{-\int_0^x c_\tau ds} &= \int_0^x c_\varepsilon e^{-tc_\tau} dt \;+\; I_b e^{-xc_\tau} \\
&= -\frac{c_\varepsilon}{c_\tau} e^{-tc_\tau}\Big|_0^x \;+\; I_b e^{-xc_\tau} \\
&= \frac{c_\varepsilon}{c_\tau} e^0 \;-\; \frac{c_\varepsilon}{c_\tau} e^{-xc_\tau} \;+\; I_b e^{-xc_\tau} \\
&= \frac{c_\varepsilon}{c_\tau} \;+\; \left(I_b - \frac{c_\varepsilon}{c_\tau}\right) e^{-xc_\tau}
\end{aligned}$$

□

A.2 Linear Emission and Constant Extinction

$$\int_0^x (m_\varepsilon t + c_\varepsilon) e^{-\int_0^t c_\tau ds} dt \quad + \quad I_b e^{-\int_0^x c_\tau ds}$$

$$= \frac{c_\varepsilon}{c_\tau} \;+\; \frac{m_\varepsilon}{c_\tau^2} \;+\; e^{-c_\tau x}\left(I_b - \frac{c_\varepsilon}{c_\tau} - \frac{m_\varepsilon}{c_\tau^2}(c_\tau x + 1)\right)$$

Proof. We shall use the result, available from any table of integrals, that

$$\int u e^{cu} du \quad = \quad e^{cu}\left(\frac{u}{c} - \frac{1}{c^2}\right)$$

We have:

$$\int_0^x (m_\varepsilon t + c_\varepsilon) e^{-\int_0^t c_\tau ds} dt \quad + \quad I_b e^{-\int_0^x c_\tau ds}$$

$$= \int_0^x (m_\varepsilon t + c_\varepsilon) e^{-tc_\tau} dt \;+\; I_b e^{-xc_\tau}$$

$$= m_\varepsilon \int_0^x t e^{-tc_\tau} dt \;+\; \int_0^x c_\varepsilon e^{-tc_\tau} dt \;+\; I_b e^{-xc_\tau}$$

$$= k_1 + k_2$$

where

$$k_1 = m_\varepsilon \int_0^x t e^{-tc_\tau} dt$$

$$k_2 = \int_0^x c_\varepsilon e^{-tc_\tau} dt \;+\; I_b e^{-xc_\tau}$$

From section A.1, we see:

$$k_2 = \frac{c_\varepsilon}{c_\tau} \;+\; \left(I_b - \frac{c_\varepsilon}{c_\tau}\right) e^{-xc_\tau}$$

Using the result mentioned above,

$$k_1 \;=\; m_\varepsilon e^{-tc_\tau}\left(\frac{t}{-c_\tau} - \frac{1}{c_\tau^2}\right)\Bigg|_0^x$$

$$= m_\varepsilon e^{-xc_\tau}\left(\frac{x}{-c_\tau} - \frac{1}{c_\tau^2}\right) \;+\; \frac{m_\varepsilon}{c_\tau^2}$$

$$= -e^{-xc_\tau}\frac{m_\varepsilon}{c_\tau^2}\,(c_\tau x + 1) \;+\; \frac{m_\varepsilon}{c_\tau^2}$$

Therefore,

$$k_1 + k_2 = -e^{-xc_\tau}\frac{m_\varepsilon}{c_\tau^2}(c_\tau x + 1) \;+\; \frac{m_\varepsilon}{c_\tau^2} \;+\; \frac{c_\varepsilon}{c_\tau} \;+\; \left(I_b - \frac{c_\varepsilon}{c_\tau}\right)e^{-xc_\tau}$$

$$= \frac{c_\varepsilon}{c_\tau} \;+\; \frac{m_\varepsilon}{c_\tau^2} \;+\; e^{-c_\tau x}\left(I_b - \frac{c_\varepsilon}{c_\tau} - \frac{m_\varepsilon}{c_\tau^2}(c_\tau x + 1)\right)$$

□

A.3 Linear Emission and Linear Extinction

$$\int_0^x (m_\varepsilon t + c_\varepsilon)\, e^{-\int_0^t m_\tau s + c_\tau ds}\, dt \quad + \quad I_b e^{-\int_0^x m_\tau s + c_\tau ds} \quad =$$
$$\sqrt{\frac{\pi}{2m_\tau^3}}\,(c_\varepsilon m_\tau - c_\tau m_\varepsilon)\,E \;+\; \frac{m_\varepsilon}{m_\tau} \;+\; e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}\left(I_b - \frac{m_\varepsilon}{m_\tau}\right)$$

where

$$E = e^{\frac{c_\tau^2}{2m_\tau}}\left(\operatorname{erf}\left(\frac{c_\tau}{\sqrt{2m_\tau}}\right) - \operatorname{erf}\left(\frac{c_\tau + xm_\tau}{\sqrt{2m_\tau}}\right)\right)$$

Proof. We shall make two observations. Firstly,

$$\operatorname{erf}(u) = \frac{2}{\sqrt{\pi}}\int_0^u e^{-x^2} du$$
$$\Rightarrow \qquad \frac{d}{du}\operatorname{erf}(u) = \frac{2}{\sqrt{\pi}}e^{-u^2}$$

and secondly,

$$e^{-\int_0^x m_\tau s + c_\tau ds} = e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}$$

Thus,

$$\int_0^x (m_\varepsilon t + c_\varepsilon)\, e^{-\int_0^t m_\tau s + c_\tau ds}\, dt \;+\; I_b e^{-\int_0^x m_\tau s + c_\tau ds} \quad =$$
$$\int_0^x (m_\varepsilon t + c_\varepsilon)\, e^{-t\left(c_\tau + \frac{1}{2}tm_\tau\right)}\, dt \;+\; I_b e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}$$

The proof proceeds as follows. Firstly, we shall show by differentiation that:

$$\int (m_\varepsilon t + c_\varepsilon)\, e^{-t\left(c_\tau + \frac{1}{2}tm_\tau\right)}\, dt \quad =$$
$$\sqrt{\frac{\pi}{2m_\tau^3}}e^{\frac{c_\tau^2}{2m_\tau}}\operatorname{erf}\left(\frac{c_\tau + tm_\tau}{\sqrt{2m_\tau}}\right)(c_\varepsilon m_\tau - c_\tau m_\varepsilon) \;-\; \frac{m_\varepsilon}{m_\tau}e^{-t\left(c_\tau + \frac{1}{2}tm_\tau\right)}$$

Then, we shall derive an expression for

$$\int_0^x (m_\varepsilon t + c_\varepsilon)\; e^{-t\left(c_\tau + \frac{1}{2}tm_\tau\right)}\; dt \;+\; I_b e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}$$

giving the desired result.

Thus,

$$\frac{d}{dt}\left(\sqrt{\frac{\pi}{2m_\tau^3}}e^{\frac{c_\tau^2}{2m_\tau}}\,\mathrm{erf}\left(\frac{c_\tau + tm_\tau}{\sqrt{2m_\tau}}\right)(c_\varepsilon m_\tau - c_\tau m_\varepsilon)\;-\;\frac{m_\varepsilon}{m_\tau}e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}\right)$$

$$= \sqrt{\frac{\pi}{2m_\tau^3}}e^{\frac{c_\tau^2}{2m_\tau}}\,(c_\varepsilon m_\tau - c_\tau m_\varepsilon)\,\frac{d}{dt}\,\mathrm{erf}\left(\frac{c_\tau + tm_\tau}{\sqrt{2m_\tau}}\right)\;-\;\frac{m_\varepsilon}{m_\tau}\frac{d}{dt}e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)} \tag{A.1}$$

Now,

$$\frac{d}{dt}\,\mathrm{erf}\left(\frac{c_\tau + tm_\tau}{\sqrt{2m_\tau}}\right) = \frac{2}{\sqrt{\pi}}e^{\frac{-(c_\tau+tm_\tau)^2}{2m_\tau}}\sqrt{\frac{m_\tau}{2}}$$

and

$$\frac{d}{dt}e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)} = -(c_\tau + m_\tau t)e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}$$

So equation (A.1) is:

$$= \sqrt{\frac{\pi}{2m_\tau^3}}e^{\frac{c_\tau^2}{2m_\tau}}\,(c_\varepsilon m_\tau - c_\tau m_\varepsilon)\,\frac{2}{\sqrt{\pi}}e^{\frac{-(c_\tau+tm_\tau)^2}{2m_\tau}}\sqrt{\frac{m_\tau}{2}} \quad +$$

$$\frac{m_\varepsilon}{m_\tau}(c_\tau + m_\tau t)e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}$$

$$= \frac{1}{m_\tau}\,(c_\varepsilon m_\tau - c_\tau m_\varepsilon)\,e^{\frac{c_\tau^2-(c_\tau+tm_\tau)^2}{2m_\tau}} \;+\; \frac{m_\varepsilon}{m_\tau}(c_\tau + m_\tau t)e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}$$

$$= \frac{1}{m_\tau}\,(c_\varepsilon m_\tau - c_\tau m_\varepsilon)\,e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)} \;+\; \frac{m_\varepsilon}{m_\tau}(c_\tau + m_\tau t)e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}$$

$$= \left(\frac{1}{m_\tau}\,(c_\varepsilon m_\tau - c_\tau m_\varepsilon) + \frac{m_\varepsilon}{m_\tau}(c_\tau + m_\tau t)\right)e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}$$

$$= (c_\varepsilon + m_\varepsilon t)\,e^{-t\left(c_\tau+\frac{1}{2}tm_\tau\right)}$$

as desired. Now, we shall show the final result:

$$\int_0^x (m_\varepsilon t + c_\varepsilon)\, e^{-t\left(c_\tau + \frac{1}{2}tm_\tau\right)}\, dt \;+\; I_b e^{-\int_0^x m_\tau s + c_\tau ds}$$

$$= \sqrt{\frac{\pi}{2m_\tau^3}} e^{\frac{c_\tau^2}{2m_\tau}} \operatorname{erf}\left(\frac{c_\tau + tm_\tau}{\sqrt{2m_\tau}}\right)(c_\varepsilon m_\tau - c_\tau m_\varepsilon) \;-\; \frac{m_\varepsilon}{m_\tau} e^{-t\left(c_\tau + \frac{1}{2}tm_\tau\right)}\Bigg|_0^x$$

$$+ I_b e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}$$

$$= \sqrt{\frac{\pi}{2m_\tau^3}} e^{\frac{c_\tau^2}{2m_\tau}} (c_\varepsilon m_\tau - c_\tau m_\varepsilon) \operatorname{erf}\left(\frac{c_\tau + xm_\tau}{\sqrt{2m_\tau}}\right) \;-\; \frac{m_\varepsilon}{m_\tau} e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)} \;-$$

$$\sqrt{\frac{\pi}{2m_\tau^3}} e^{\frac{c_\tau^2}{2m_\tau}} (c_\varepsilon m_\tau - c_\tau m_\varepsilon) \operatorname{erf}\left(\frac{c_\tau}{\sqrt{2m_\tau}}\right) \;+\; \frac{m_\varepsilon}{m_\tau} \;+\; I_b e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}$$

$$= \sqrt{\frac{\pi}{2m_\tau^3}} e^{\frac{c_\tau^2}{2m_\tau}} (c_\varepsilon m_\tau - c_\tau m_\varepsilon) \left(\operatorname{erf}\left(\frac{c_\tau}{\sqrt{2m_\tau}}\right) - \operatorname{erf}\left(\frac{c_\tau + xm_\tau}{\sqrt{2m_\tau}}\right)\right) \quad +$$

$$\frac{m_\varepsilon}{m_\tau} \;+\; e^{-x\left(c_\tau + \frac{1}{2}xm_\tau\right)}\left(I_b - \frac{m_\varepsilon}{m_\tau}\right)$$

which trivially reduces to the result. □

Appendix B

File Types

The scene description used by the renderer is a formatted text file. The user is thus able to generate scene descriptions without being limited to the animation facilities provided by the application, using whatever text processing language she or he is familiar with.

The following is a description of the scene file. Tokens are delimited by tab or newline characters, so spaces can be used to enhance readability. We shall list each field with the type of data expected. Vectors are a list of three doubles, and matrices are lists of sixteen doubles, using homogeneous coordinates. Boolean variables are one of $\{0, 1\}$.

```
Tachos Scene Description v1.0

Volume:                 file path
Material file:          file path
Image size:             int x int
Antialiasing:           boolean
Interpolation:          int
Optimisation:           int

Frame

Lightsource position:   vector
Lightsource brightness: double
Camera transform:       matrix
Volume transform:       matrix

End
```

The antialiasing, interpolation and optimisation fields are intended for future expansion, and are currently usused.

The material description is as follows. The specular and shininess fields are also for future expansion. The density range field gives the boundaries of the interval within which a sample is classified with certainty. The ranges must satisfy the properties that the lower bound of the first material is 0, the higher bound of the last material is 255, and that all intervals are disjoint. As many materials as desired may be specified, within these constraints.

```
Tachos Material Description v1.0

name1

Absorption RGB:         double  double  double
Diffuse RGB:            double  double  double
Diffuse brightness:     double
Specular RGB:           double  double  double
Shininess:              int
Density range:          int     int

name2

.
.
.

nameN

.
.
.

End
```

Bibliography

[AK87] James Arvo and David B. Kirk. Fast ray tracing by ray classification. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21:55–64, July 1987.

[Arv90] James Arvo. Ray tracing with meta-hierarchies. In *SIGGRAPH '90 Advanced Topics in Ray Tracing course notes*, August 1990.

[BJNN97] Martin L. Brady, Kenneth Jung, HT Nguyen, and Thinh Nguyen. Two-phase perspective ray casting for interactive volume navigation. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 183–190, November 1997.

[Bli82] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics (SIGGRAPH '82 Proceedings)*, 16(3):21–29, July 1982.

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

[Dee95] Michael Deering. Geometry compression. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 13–20. ACM SIGGRAPH, Addison-Wesley, August 1995.

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

[DT81] L. J. Doctor and J. G. Torborg. Display techniques for octree-encoded objects. *IEEE Computer Graphics and Applications*, 1:29–38, July 1981.

[FI85] Akira Fujimoto and Kansei Iwata. Accelerated ray tracing. In Tosiyasu Kunii, editor, *Computer Graphics: Visual Technology and Art (Proceedings of Computer Graphics Tokyo '85)*, pages 41–65. Springer Verlag, 1985.

[FS97] Jason L. Freund and Kenneth Sloan. Accelerated volume rendering using homogenous region encoding. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 191–196, November 1997.

[FSHR96] Shiaofen Fang, Rajagopalan Srinivasan, Su Huang, and Ragu Raghavan. Deformable volume rendering by 3D texture mapping and octree encoding. In *IEEE Visualization '96*. IEEE, October 1996.

[FTI86] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. ARTS: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, pages 16–26, April 1986.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. The Systems Programming Series. Addison-Wesley Publishing Company, second edition, 1990.

[Gla84] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[GLDH97] M. H. Gross, L. Lippert, R. Dittrich, and S. Häring. Two methods for wavelet-based volume rendering. *Computers and Graphics*, 21(2):237–252, 1997.

[HHE96] P. Hastreiter, W. Hopfer, and T. Ertl. Semi-automatic registration of 3d-multi-modality brain images based on an information theoretic approach. In B. Arnolds, H. Müller, D. Saupe, and T. Tolxdorff, editors, *4. Workshop: Digitale Bildverarbeitung in der Medizin*, pages 132–137. Dt. Ges. f. med. Inf., Biom. u. Epidem. (GMDS) e.V., 1996.

[Hil90] F. S. Hill, Jr. *Computer Graphics*. Macmillan Publishing Company, 1990.

[HK96] Taosong He and Arie Kaufman. Fast stereo volume rendering. In *IEEE Visualization '96*, pages 49–56, October 1996.

[Kul98] Peter Kulka. A survey of projection methods for direct volume rendering. In *Proceedings of the Image and Vision Computing '98 Conference*, November 1998.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[LC96] Chyi-Cheng Lin and Yu-Tai Ching. An efficient volume-rendering algorithm with an analytic approach. *The Visual Computer*, 12(10):515–526, 1996.

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

[Lev90] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[LGE97] Christoph Lürig, Roberto Grosso, and Thomas Ertl. Implicite adaptive volume ray-casting. In *Proceedings Graphicon 97*, pages 114–120, 1997.

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

[LY96] A. Law and R. Yagel. An optimal ray traversal scheme for visualizing colossal medical volumes. Technical report, Ohio State University, 1996.

[Mal93] Tom Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.

[Man89] Udi Manber. *Introduction to Algorithms: A Ceative Approach.* Addison-Wesley Publishing Company, 1989.

[Max95] Nelson Max. Optical models for direct volume rendering. *IEEE Transcations on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[ML94] Stephen R. Marschner and Richard J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Visualization '94*, pages 100–107. IEEE Conputer Society Technical Committee on Computer Graphics, IEEE Computer Society Press, October 1994.

[MMMY96] Torsten Möller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. Classification and local error estimation of interpolation and derivative filters for volume rendering. In *1996 Volume Visualization Symposium*, pages 71–78. IEEE, October 1996.

[MMMY97] Torsten Möller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. A comparison of normal estimation schemes. In *IEEE Visualization '97*. IEEE, November 1997.

[MS90] C. Montani and R. Scopigno. Rendering volumetric data using the STICKS representation scheme. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 87–93, November 1990.

[MY96] R. Machiraju and R. Yagel. Reconstruction error characterization and control: A sampling theory approach. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):364–378, December 1996.

[Nov93] Kevin Lawrence Novins. *Towards Accurate and Efficient Volume Rendering.* PhD thesis, Cornell University, 1993.

[PBB98] Heinz-Otto Peitgen, Wilhelm Berghorn, and Matthias Biel. The complete visible human. Springer-Verlag CD-Rom, 1998. See `http://www.nlm.nih.gov/research/visible/visible_human.html`.

[Sab88] Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics*, 22(4):51–58, August 1988.

[Sam89] Hanan Samet. Implementing ray tracing with octrees and neighbor finding. *Computers and Graphics*, 13(4):445–60, 1989.

[San85] J. Sandor. Octree data structures and perspective imagery. *Computers and Graphics*, 9(4):393–405, 1985.

[SK94] Lisa M. Sobierajski and Arie E. Kaufman. Volumetric ray tracing. In Arie E. Kaufman and Wolfgang Krueger, editors, *1994 Symposium on Volume Visualization*, pages 11–18. ACM SIGGRAPH, October 1994.

[SMM+97] J. Edgar Swan, II, Klaus Mueller, Torsten Möller, Naeem Sharif, Roger Crawfis, and Roni Yagel. An anti-aliasing technique for splatting. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 197–204. IEEE, November 1997.

[TL93] Takashi Totsuka and Marc Levoy. Frequency domain volume rendering. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:271–278, August 1993.

[UK88] Craig Upson and Michael Keeler. V-buffer: Visible volume rendering. *Computer Graphics*, 22(4):59–65, August 1988.

[vW94] Allen van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, October 1994.

[Wes89] Lee Westover. Interactive volume rendering. In *Proceedings of the Chapel Hill Workshop on Volume Rendering*, pages 9–16. ACM, 1989.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24:367–376, August 1990.

[Wes96] Rüdiger Westermann. *A Multiresolution Framework for Volume Rendering*. PhD thesis, Universität Dortmund, 1996.

[WHG84] Hank Weghorst, Gary Hooper, and Donald P. Greenburg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.

[WJ95] D. M. Weinstein and C. R. Johnson. Hierarchical data structures for interactive volume visualization. Technical Report UUCS-95-012, University of Utah, 1995.

[WMS98] Peter L Williams, Nelson L Max, and Clifford M Stein. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37–54, January 1998.

[WSC+95] Kyu-Young Whang, Ju-Wong Song, Ju-Woong Chang, Ji-Yun Kim, Wan-Sup Cho, Chong-Mok Park, and Il-Yeol Song. Octree-R: An adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):343–349, December 1995.

[Wün97] Burkhard Wünsche. A survey and analysis of common polygonization methods & optimization techniques. Technical report, Auckland University, August 1997.

[Yag96] R. Yagel. Towards real time volume rendering. In *Proceedings of GRAPHICON'96*, pages 230–241, July 1996.

[YKFT84] K. Yamaguchi, T. L. Kunii, K. Fujimara, and H. Toriya. Octree-related data structures and algorithms. *IEEE Computer Graphics and Applications*, 3:53–59, January 1984.

[YL95] Boon-Lock Yeo and Bede Liu. Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, March 1995.

[ZvOC$^+$96] Karel J. Zuiderveld, Peter M. A. van Ooijen, John W C Chin-A-Woeng, Pieter C. Buijs, Marco Olree, and Frits H. Post. Clinical evaluation of interactive volume visualization. In *IEEE Visualization '96*, pages 367–371. IEEE, October 1996.