INTERACTIVE VOLUME RENDERING TECHNIQUES

by

Joe Michael Kniss

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

The University of Utah

May 2002
SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Joe Michael Kniss

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

______________________________
Chair: Chuck Hansen

______________________________
Peter Shirley

______________________________
Chrisopher Johnson
THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of ______ Joe Michael Kniss _______ in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

______________________________________________
Date 

Chuck Hansen
Chair, Supervisory Committee

Approved for the Major Department

______________________________________________
Thomas C. Henderson
Chair/Dean

Approved for the Graduate Council

______________________________________________
David S. Chapman
Dean of The Graduate School
ABSTRACT

Interactive volume rendering is an important visualization technique. It allows scientists to quickly gain understanding of bio-medical, industrial, and simulation data. Interactivity provides a user with valuable feedback and a visualization experience that cannot be attained with a single image or movie. Recent advancements in commodity programmable graphics hardware allow not only interactive volume rendering, but image quality and shading options that rival sophisticated software based volume rendering techniques. Unfortunately, hardware memory limitations restrict the size of interactively renderable datasets. This thesis presents a parallel pipelined approach for visualizing volumetric datasets which are orders of magnitude larger than those that can be interactively rendered on a single graphics card.

Most direct volume renderings produced today employ one-dimensional transfer functions, which assign color and opacity to the volume based solely on the single scalar quantity which comprises the dataset. Though they have not received widespread attention, multi-dimensional transfer functions are a very effective way to extract materials and their boundaries for both scalar and multivariate data. However, identifying good transfer functions is difficult enough in one dimension, let alone two or three dimensions. This thesis demonstrates an important class of three-dimensional transfer functions for scalar data, and describes the application of multi-dimensional transfer functions to multivariate data. It presents a set of direct manipulation widgets and new interaction modalities that make specifying a transfer function intuitive and convenient. This thesis also describes how to use modern graphics hardware to both interactively render with multi-dimensional transfer functions and to provide interactive shadows for volumes. The transfer functions, widgets, and hardware combine to form a powerful system for interactive volume exploration.
To my parents.
For their love and support.

To my friends.
For their patience and understanding.
CONTENTS

ABSTRACT ................................................................. ii

LIST OF FIGURES ......................................................... vi

ACKNOWLEDGEMENTS ................................................... ix

CHAPTERS

1. INTRODUCTION .......................................................... 1
   1.1 Speed ............................................................... 6
   1.2 Size ............................................................... 6
   1.3 Quality ........................................................... 6
   1.4 Overview ......................................................... 7

2. VOLUME RENDERING BACKGROUND .................................... 9
   2.1 The Modern Direct Volume Rendering Pipeline .................. 10
       2.1.1 Preprocessing ............................................... 12
       2.1.2 Reconstruction ............................................. 12
       2.1.3 Classification .............................................. 13
       2.1.4 Shading .................................................... 15
       2.1.5 Compositing ............................................... 16
   2.2 3D Texture Based Volume Rendering ............................. 17
   2.3 Related Work .................................................... 18
       2.3.1 Ray Casting ............................................... 18
       2.3.2 Shear-Warp ............................................... 19
       2.3.3 2D Texture Based Volume Rendering .................... 19
       2.3.4 Volume Pro ............................................... 20
       2.3.5 Splatting .................................................. 20
       2.3.6 Parallel Volume Rendering ............................... 21
       2.3.7 General Light Transport ................................. 21

3. PARALLEL HARDWARE VOLUME RENDERING ....................... 23
   3.1 TRex System Overview .......................................... 23
       3.1.1 Pre-processing ............................................ 25
       3.1.2 The TRex Pipeline ....................................... 25
       3.1.3 Stage 1: Subvolume Reader .............................. 26
       3.1.4 Stage 2: Render .......................................... 28
       3.1.5 Stage 3: Compositor .................................... 29
       3.1.6 Stage 4: User Interface ................................. 30
   3.2 Discussion ..................................................... 31
   3.3 Immersive TRex ................................................ 34
3.4 TRex Widgets .......................................................... 35

4. MULTI-DIMENSIONAL TRANSFER FUNCTIONS .............. 38
   4.1 Scalar Data .......................................................... 39
   4.2 Multivariate data .................................................... 43
   4.3 Discussion ............................................................ 48

5. INTERACTION AND TOOLS ............................................. 52
   5.1 Probing and Dual-Domain Interaction ............................ 54
   5.2 Data Probe Widget .................................................. 57
   5.3 Clipping Plane Widget .............................................. 57
   5.4 Transfer Function Widget .......................................... 57
   5.5 Classification Widgets ............................................. 58
   5.6 Shading Widget ..................................................... 60
   5.7 Color Picker Widget ............................................... 60
   5.8 Discussion ............................................................ 60

6. RENDERING AND HARDWARE ........................................... 62
   6.1 Dependent Texture Reads .......................................... 62
   6.2 Classification ...................................................... 63
   6.3 Surface Shading .................................................... 64
   6.4 Shadows .............................................................. 65

7. CONCLUSIONS AND FUTURE WORK .................................. 70
   7.1 Parallel Volume Rendering ......................................... 70
   7.2 Multi-Dimensional Transfer Functions ........................... 71
   7.3 Shading ............................................................... 72
   7.4 Limitations .......................................................... 73
   7.5 Summary ............................................................. 73

REFERENCES ................................................................. 75
LIST OF FIGURES

1.1 An example of X-ray Computer Tomography of the Visible Male. .......... 2
1.2 Industrial X-ray Computer Tomography of the GE Turbine Blade. .......... 2
1.3 An example of Magnetic Resonance Imaging of the Duke Mouse. .......... 3
1.4 A volume rendering of a single neuron from Limited-Angle CT. .......... 4
1.5 Two examples of computer generated physical simulations. ............. 4
1.6 Three volume rendered timesteps from a numerical weather simulation. ... 5
2.1 Isosurface renderings of the Human Tooth CT. ............................. 9
2.2 Volume renderings of the Human Tooth CT. ............................. 10
2.3 The modern direct volume rendering pipeline. ............................ 11
2.4 A comparison of pre-classification verses post-classification volume renderings of the inner ear. ............................................ 14
2.5 A comparison of pre-classification and pre-shading verses post-classification and post-shading volume rendering pipelines. ...................... 14
2.6 View aligned slicing and spherical shell slicing of volume data. ............ 17
3.1 Time-varying Raleigh-Taylor fluid instability dataset (1024^3). .......... 24
3.2 The TRex Pipeline........................................................................... 26
3.3 SGI Origin 2000 architecture. ..................................................... 27
3.4 Texture Lookup Table. Note: the Alpha Band (top) has been multiplied by the Color Band (bottom) to show the resulting alpha weighted colors. ...... 28
3.5 Texture Lookup Table with the Semi-Automatic generation of alpha mappings. The top band allows the user to select data values based on their distance from an ideal boundary detected in the volume. The middle band shows the generated alpha mapping multiplied by the color band. .......... 29
3.6 Stage 3, compositing, detail. In this example 4 image buffers from stage 2 are composited as D over C over B over A. Image buffer D would then be downloaded to the UI’s frame buffer in Stage 4. ......................... 30
3.7 The entire system: renders at the top, compositors in the middle, final image at the bottom. The labels are (subvolume, image-stripe). ............. 32
3.8 View Point construction for Semi-Immersive Environments (A), and Fully-Immersive Environments (B) for an arbitrary head orientation. Notice: View directions are parallel and perpendicularly intersect the view surface. The overlap region in B is variable on many head mounted displays ....... 35
3.9 TRex Widgets, (a) demonstrates the Color Map and Clip Widgets, (c) demonstrates the Data Probe widget and shows intermediate renderings to the left. .................................................. 37

4.1 Material and boundary identification of the Chapel Hill CT Head with data value alone (a) versus data value and gradient magnitude ($f'$), seen in (b). The basic materials captured by CT, air (A), soft tissue (B), and bone (C) can be identified using a 1D transfer function as seen in (a). 1D transfer functions, however, cannot capture the complex combinations of material boundaries; air and soft tissue boundary (D), soft tissue and bone boundary (E), and air and bone boundary (F) as seen in (b) and (c). ............... 40

4.2 The frontal and maxillary sinuses of the Visible Male CT. While a 1D transfer function can show the sinuses along with the skin, it cannot capture them in isolation. Only a higher dimensional transfer function, in this case a 2D transfer function using data value and gradient magnitude, can uniquely classify them. ................................................................. 41

4.3 Material and boundary identification of the human tooth CT with data value and gradient magnitude ($f'$), seen in (a), and data value and second derivative ($f''$), seen in (b). The background/dentin boundary (F) cannot be adequately captured with data value and gradient magnitude alone. (c) shows the results of a 2D transfer function designed to show only the background/dentin (F) and dentin/enamel boundaries (G). The background/enamel (H) and dentin/pulp (E) boundaries are erroneously colored. Adding the second derivative as a third axis to the transfer function disambiguates the boundaries. (d) shows the results of a 3D transfer function that gives lower opacity to non-zero second derivative values. .. 42

4.4 The behavior of primary data value ($f$), gradient magnitude ($f'$), and the second directional derivative ($f''$) as a function of position (a) and as a function of data value (b). ................................................................. 43

4.5 The Visible Male RGB (color) data. The opacity is set using a 3D transfer function, color is taken directly from the data. The histogram (a) is visualized as projections on the primary planes of the RGB color space. .. 44

4.6 A kidney from the Visible Male RGB dataset. The renal vein is labeled (E). A clipping plane reveals internal structure (right). ......................... 45

4.7 Frontal zones of a numerical weather simulation. (a) shows a reference map of the simulation domain. (b) shows the results of an expert analysis using scalar data visualizations similar to (c) and (d). (c) and (d) are slices through the dataset with a spectral color map. (e) shows a 2D log-scale joint histogram of temperature versus humidity. Region (A) shows the ranges of these data values that represent a mid-latitude front, (B) identifies the warm air mass, (C) identifies the cold air mass. (f) shows a volume rendering using a 3D transfer function which emphasizes regions (B) and (C). $\|G\|$ from Equation 4.3 is used as the third axis of the transfer function for the rendering on the right to emphasize the portions of the air masses near the front. The dotted line shows a path through the dataset, the values along this line are shown in (e). .................................................. 49
4.8 A default transfer function for scalar data applied to the Chapel Hill CT. Hue varies along the data value axis and opacity varies along the gradient magnitude axis. A clipping plane reveals internal structure (right)........... 50

4.9 The dentin of the Human Tooth CT. (a) shows that a 1D transfer function, simulated by assigning opacity to data values regardless of gradient magnitude, will erroneously color the background/enamel boundary. A 2D transfer function, shown in (b) can avoid assigning opacity to the range of gradient magnitudes that define this boundary. ......................... 51

5.1 The direct manipulation widgets. ........................................ 53

5.2 Dual-Domain Interaction .................................................. 56

5.3 The brain of the Visible Male CT. The transfer functions were created using dual-domain interaction. A detail region shows how small the region that identifies this subtle feature is in the transfer function domain. .......... 61

6.1 Modified slice axis for light transport............................... 67

6.2 Volume renderings of the Visible Male CT (frozen) demonstrating combined surface shading and shadows. ........................................ 69

7.1 A virtual reality interface for manipulating multi-dimensional transfer functions. ................................................................. 72
ACKNOWLEDGEMENTS

I would like to thank Chuck Hansen for giving me an opportunity to prove myself as a student and a scientist. His intuition and guidance have always been spot on. I thank Chris Johnson for providing me with seemingly endless support and enthusiasm, and Pete Shirley for countless impromptu meetings over coffee and lunch to discuss everything from light transport to evolutionary psychology. I would like to give a special thanks to Gordon Kindlmann for being a friend and mentor.

I extend my gratitude to the staff at Los Alamos National Lab’s ACL for kick-starting my research and providing a rich environment to grow as a scientist, with special thanks to Al McPherson, Pat McCormick, and Jim Ahrens. Thanks to my colleagues at the University of Utah. Milan Ikits for mentoring me in the ways of virtual reality. Richard Coffee and Nathan Galli for always on time media magic. Allyne Betancourt, Heidi Borjesson, Raelynn Potts, and Sharla Mallory for always going out of their way for needs of myself and others. David Hart and Nate Robbins for their friendship and help. Robert Cummins for being patient and prompt with even the smallest of my system problems. Greg Jones, Steve Parker, Rob MacLeod, and Ross Whitaker for enlightening conversations and many last minute paper reads. Thanks to my colleagues abroad. Juergen Schulze-Doebold, Uli Lang, and Uve Woessner for the opportunity of a lifetime to visit Germany and conduct research with amazing scientists. Christof Rezk-Salama, Klaus Engel, Markus Hadwiger, and Rudiger Westermann for their collaboration and encouragement. Special thanks to Klaus for Figure 2.4.

Many thanks to those in the “real world” for their enthusiasm and support. Richard Grossen from SGI for the serendipitous Octane 2 loan. Cass Everitt, John Spitzer, and the rest of the NVIDIA development team for generous technical support and hardware gifts. Evan Hart and the OpenGL development team at ATI for their hands on interest in this work and hardware gifts. The Coffee Garden and its many talented baristas for thousands of espressos at eleven o’clock pm.

My most sincere thanks to my friends and family for standing by me as I attempt my biggest undertaking yet.
The contents of this thesis are the result of nearly three years of research. Chapters 3, 4, 5, 6, and 7 are taken from previously published work. It is important to recognize my coauthors on these papers for their valuable contributions. Chapter 3 is taken from “TRex: Interactive Texture Based Volume Rendering for Extremely Large Datasets” [34], which was coauthored with Patrick McCormick, Allen McPherson, James Ahrens, Jamie Painter, Alan Keahey, and Charles Hansen. Chapters 4, 5, and 6 are taken from “Interactive Volume Rendering using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets” [32] and “Multi-Dimensional Transfer Functions for Interactive Volume Rendering” [33], which was coauthored by Gordon Kindlmann and Chuck Hansen. Chapter 4 also contains results from “Volume Rendering Multivariate Data to Visualize Meteorological Simulations: A Case Study” [30], which was written in collaboration with Michel Grenier and Tom Robinson from the Canada Meteorological Centre. Figures 1.4, 1.5(b), and 2.2(b) were generated using a volume shading technique developed in collaboration with David Ebert and Simon Premoze [35].
CHAPTER 1

INTRODUCTION

Visualization is the process of generating images from scientific data to gain an understanding of its qualitative and quantitative behavior. In many cases, this data is spatially varying, meaning that the structures it represents are three-dimensional (3D). While scientists can gain some insight from visualizing the data as a series of two-dimensional (2D) slices, a 3D reconstruction of the dataset provides a more natural and comprehensive view of important features. Direct volume rendering has proven to be an important visualization modality as a flexible and efficient method for producing images of three-dimensional data fields [13, 39]. Many disciplines benefit from volume visualization, including radiology [40], microscopy [16], and computational fluid dynamics [60].

Medical imaging systems are capable of acquiring 3D images of the human body, which allow physicians to non-invasively assess trauma and illness [21]. Computer Tomography (CT) uses a series of X-ray beams to generate cross-sectional images that indicate tissue density at discrete locations. By acquiring multiple slices, a 3D image of the body can be assembled. Figure 1.1 shows an example of a 2D CT slice and a volume rendering generated from the same data. Industrial CT scanners are capable of imaging fine metal structures, which provides important information about the quality and safety of mechanical parts. Figure 1.2 shows an example of Industrial CT. Another common medical imaging modality is Magnetic Resonance Imaging (MRI). This technique is well suited for imaging the soft tissues in the human body, such as the brain, by characterizing tissues based on measurements of atomic spin. Figure 1.3 shows an example of a MRI slice and a volume rendering of the data. Other medical imaging techniques, such as Positron Emission Tomography (PET), Single Photon Emission Computer Tomography (SPECT), and Functional Magnetic Resonance Imaging (fMRI), provide valuable insight into the metabolic activity of living tissues. These modalities can be used to identify certain types of cancer, which may be difficult to locate with other imaging methods. They can also help researchers identify regions of the brain that are responsible for physical actions.
Figure 1.1. An example of X-ray Computer Tomography of the Visible Male.

Figure 1.2. Industrial X-ray Computer Tomography of the GE Turbine Blade.
or specific types of cognitive thought. Medical imaging technology can also be used to examine microscopic structures. Laser Scanning Confocal Microscopy and Limited-Angle CT offer scientists a three dimensional view of structures as small as a single neuron, as shown in Figure 1.4. Computer generated simulations of physical systems allow researchers to investigate phenomena such as explosions and long term weather patterns, which may be difficult to study directly [18]. Physical simulations have a wide range of application areas, from the formation of galaxies in Astrophysics to the propagation of electro-magnetic fields in biological systems. Visualization plays an important role in the evaluation and assessment of these mathematical models. Figure 1.5 shows two examples of volume visualization applied to physical simulations. Figure 1.5(a) shows how stress waves propagate through a collection of tightly packed grains made of a highly energetic, explosive, material. Figure 1.5(b) shows a computational fluid dynamics simulation of a heptane fire.

State of the art simulations of physical systems, like the weather simulation shown in Figure 1.6, can generate terabytes to petabytes of time-varying data where a single time step can contain more than a gigabyte of data per variable. Even modestly sized bio-medical datasets can contain millions of data values. The key to understanding this data is the ability to visualize the global and local relationships of data elements. Direct volume rendering is an excellent method for examining these properties. It allows
Figure 1.4. A volume rendering of a single neuron from Limited-Angle CT.

Figure 1.5. Two examples of computer generated physical simulations.
each data element to contribute to the final image and provides the ability to query not only the spatial relationship of data elements, but their quantitative relationships as well. Each imaging modality or numerical simulation generates data that has its own characteristics and behavior, which may require specific techniques for creating appropriate visualizations. Identifying the ranges of data values that correspond to features of interest can be a tedious trial and error process.

This thesis investigates techniques that make the process of volume visualization genuinely interactive, expressive, and intuitive. Direct volume rendering can be made interactive by leveraging graphics hardware and the efficient use of available parallel computing resources. It can be made expressive by expanding feature discrimination with derivative information and multiple data values. It can be made intuitive through the use of interaction modalities and tools that guide the user toward identifying and emphasizing features of interest. The overall quality of the visualization can be further improved with realistic lighting. These techniques combine to form a powerful system for volume exploration and visualization.

There are three primary constraints that must be considered when designing a volume rendering application:

1. **Speed** Interactive applications must maintain high frame rates.

2. **Size** Large volumes may be difficult to visualize at interactive frame rates.

3. **Quality** A volume rendering system must provide an expressive mechanism for visualizing features of interest and producing high quality images with realistic lighting.

![Figure 1.6](image_url)  
**Figure 1.6.** Three volume rendered timesteps from a numerical weather simulation.
1.1 Speed

Hardware accelerated volume rendering is an approach that allows users to achieve interactive display rates for reasonably sized datasets. The immediate feedback provided by interactivity permits a better understanding of the behavior of the data and spatial relationships of features of interest. It also allows users to fine tune the transfer function and visualization in ways that are not possible with off-line rendering techniques. This thesis presents approaches for high-quality interactive volume rendering using graphics hardware, specifically 3D texture memory and per-pixel shading operations.

1.2 Size

As memory sizes and processor speeds continue to increase at a high rate, the size of datasets will likely increase at a comparable rate. Increased computational power allows scientists to conduct numerical simulations on a finer scale, achieving more accurate results than have ever been possible. However, fine scale simulations mean larger datasets, which make the need for interactive visualization even greater. The size of interactively renderable datasets is a function of the hardware’s available texture memory and fill rate. Current hardware graphics implementations place an upper bound on physical texture memory, which is used to store a dataset, of approximately 256MB. Visualizing datasets that are larger than available texture memory requires paging the data from slower memory resources, which can significantly impact rendering performance. This thesis presents a scalable, pipelined approach for rendering datasets which are larger than the limitations of a single graphics card. This is achieved by taking advantage of multiple hardware rendering units and parallel software compositing.

1.3 Quality

Often, there are features of interest in volume data that are difficult to extract and visualize with 1D transfer functions. Many medical datasets created from CT or MRI scans contain a complex combination of boundaries between multiple materials. This situation is problematic for 1D transfer functions because of the potential for overlap between the data value intervals spanned by the different boundaries. When one data value is associated with multiple boundaries, a 1D transfer function is unable to render them in isolation. A benefit of higher dimensional transfer functions is their ability to portray subtle variations in properties of a single boundary, such as its thickness.
When working with multivariate data, a similar difficulty arises with features that can be identified only by their unique combination of multiple data values. A 1D transfer function is simply not capable of capturing this relationship.

Using multi-dimensional transfer functions in volume rendering is complicated. Even when the transfer function is only 1D, finding an appropriate transfer function is generally accomplished by trial and error. This is one of the main challenges in making direct volume rendering an effective visualization tool. Adding dimensions to the transfer function domain only compounds the problem. While this is an ongoing research area, many of the proposed methods for transfer function generation and manipulation are not easily extended to higher dimensional transfer functions. In addition, fast volume rendering algorithms that assume the transfer function can be implemented as a linear lookup table (LUT) can be difficult to adapt to multi-dimensional transfer functions due to the linear interpolation imposed on such LUTs.

This thesis presents a detailed exposition of the multi-dimensional transfer function concept, a generalization of multi-dimensional transfer functions for both scalar and multivariate data, as well as novel techniques for interactive volume surface shading and the generation of volumetric shadows. To resolve the potential complexities in a user interface for multi-dimensional transfer functions, a set of direct manipulation widgets is introduced, which make finding and experimenting with transfer functions an intuitive, efficient, and informative process. Together, the widgets and the hardware form the basis for new interaction modes which can guide users towards transfer function settings appropriate for their visualization and data exploration interests.

1.4 Overview

Chapter 2 introduces the volume rendering pipeline, 3D texture based volume rendering, and provides descriptions of other volume rendering techniques. Chapter 3 introduces a parallel pipelined approach for volume rendering extremely large datasets. This chapter also introduces a suite of direct manipulation widgets that demonstrate the importance and utility of interactive tools for volume exploration. Chapter 4 introduces the concept of multi-dimensional transfer functions. Examples are provided for scalar data with derivatives and general multivariate datasets. Chapter 5 introduces a novel interface for manipulating a multi-dimensional transfer function and new interaction modalities that link the spatial and transfer function domains in an intuitive way. The hardware
implementation details for multi-dimensional transfer functions and advanced shading
techniques are provided in Chapter 6. Chapter 7 reaches conclusions and proposes future
enhancements.
CHAPTER 2

VOLUME RENDERING BACKGROUND

Direct volume rendering is the process by which a three dimensional data field is assigned optical properties and imaged directly using a reasonable illumination model. This contrasts with surface based rendering techniques, such as isosurfacing [41], which must extract an explicit, polygonal, surface that is subsequently rendered with standard graphics hardware. An example of isosurfacing can be seen in Figure 2.1(a). Visualizing internal structure and multiple isosurfaces requires that all polygons be sorted so that they can be rendered in order. Figure 2.1(b) shows an example of multiple semi-transparent isosurfaces. The view dependent sorting required for rendering semi-transparent isosurfaces can have a major impact on the rendering performance. Direct volume rendering is advantageous because it allows each sample to potentially contribute to the final rendering. This technique can not only render isosurfaces, like those shown in Figure 2.1, but it can also render any combination of isosurfaces, boundaries between materials (as shown in Figure 2.2(a)), and the materials themselves (as shown in Figure 2.2(b)).

![An isosurfaced tooth.](image1)

![Multiple isosurfaces.](image2)

**Figure 2.1.** Isosurface renderings of the Human Tooth CT.
does not yet exist a surface based visualization technique that can generate images like those seen in Figure 2.2, which uniquely show each of the major material boundaries and accurately capture the subtle appearance of light attenuation through a material.

2.1 The Modern Direct Volume Rendering Pipeline

The most common volume rendering model used in computer graphics and scientific visualization was proposed independently by Levoy [39] and Drebin et al. [13]. This model renders the volume data onto the image plane by assigning optical properties to data values, sampling the volume at regular intervals, and compositing the results using the over or under operators [55]. This process, known as ray casting, and other equivalent approaches are described further in the previous work section (2.3). Opacity and reflective color are the only optical properties required for this model. These optical properties are assigned to data values using a transfer function, which is implemented as a table lookup. Assuming that opacity and color are always in the range zero to one, the use of Porter and Duff’s compositing operators forces the accumulated light intensities to always be in the range zero to one as well. Shading is implemented using a surface based shading model, such as the ubiquitous Blinn-Phong [2, 54] shading model, where the normalized gradient of the scalar data field is used as an approximation to a surface normal. Volumetric shadows can be added to the model by multiplying the light intensity, or reflective color, at a sample by one minus the accumulated opacity between that sample

![Image](image_url)

(a) Material boundaries. (b) Materials and boundaries with volumetric light attenuation.

Figure 2.2. Volume renderings of the Human Tooth CT.
and the light source.

The modern direct volume rendering pipeline, as proposed by Levoy and Drebin et al., has five stages, also seen in Figure 2.3:

**Preprocessing** This stage converts the data into a form suitable for rendering images. This includes data quantization, filtering, resizing, cropping, brickling, and derivative estimation.

**Reconstruction** This is the process of interpolating discretely sampled data fields to create a continuous function.

**Classification** This stage maps data values to optical properties such as opacity and color.

**Shading** A shading model can be employed to modulate the brightness and color of each sample, providing valuable surface curvature and depth queues.

**Compositing** The numerical solution of the volume rendering equation can be expressed as a simple iterative interpolation based on opacity. This interpolation is precisely the over and under operators used for compositing pairs of images [55].

![Diagram](image)

**Figure 2.3.** The modern direct volume rendering pipeline.
2.1.1 Preprocessing

Volume datasets come from a variety of sources. Medical and industrial imaging systems, including CT, MRI, or PET, acquire data from physical objects, such as the human body. These acquisition techniques require an initial reconstruction step to convert the data from its acquired representation to a spatial representation suitable for visualization. Datasets acquired using these techniques also contain noise, which may need additional filtering to remove. Numerical simulations and other synthetic data generation methods also produce datasets that benefit from volume visualization. Filtering may be required with these datasets as well to ensure that they are band limited. In many cases, features of interest only occupy a subset of the original volume. Cropping can be employed to extract the essential parts of a larger dataset. Bricking serves two important roles in volume rendering applications. First, since volume data has an inherent spatial representation, fine scale bricking can accelerate the volume rendering process by reordering the volume data to match potential memory access patterns [52]. Most hardware graphics architectures that support 3D textures perform this kind of bricking automatically. Second, large scale bricking allows a large dataset to be efficiently rendered in parts when the whole volume will not fit in high speed memory resources. The gradient and the gradient magnitude often play a role in shading and classification. These quantities must be computed from the volume data. While gradient estimation can be computed during the rendering process, it is often desirable to compute and store these quantities ahead of time.

2.1.2 Reconstruction

Reconstruction, or interpolation, refers to the process of estimating the value of a discretely sampled function at positions between samples. Reconstruction falls in the domain of signal processing, provided certain criteria are met [19]. Reconstruction is expressed mathematically as the convolution of a discretely sampled function with a reconstruction kernel. An evaluation of common reconstruction kernels for volume rendering applications can be found in Marschner and Lobb’s “An Evaluation of Reconstruction Filters for Volume Rendering” [44]. Most commercially available graphics hardware only support linear interpolation on rectilinear grids. Since this thesis is focused on interactive texture based volume rendering techniques using graphics hardware, all volume renderings in this document use tri-linear interpolation of volume data stored in three dimensional
rectilinear grids with regular spacing.

2.1.3 Classification

Classification is a term that refers to the assignment of optical properties to data values. Classification is one of the most important steps in the volume rendering pipeline since it is these optical properties that will either emphasize an important feature or de-emphasize unimportant ones. The assignment of optical properties to data values is accomplished using a transfer function, which is typically implemented as a table lookup based on data value. The order in which reconstruction and classification are executed in the volume rendering pipeline can make a significant difference in the quality and speed of renderings. A pre-classification volume renderer executes classification prior to interpolation. This means that the optical properties themselves are interpolated from the grid samples. It is important that opacity-weighted colors, i.e. color multiplied by opacity, are interpolated rather than interpolating the colors and opacities independently, which causes an artifact known as color bleeding [66]. This approach is advantageous since classification table lookups occur only once per-sample, resulting a significant performance enhancement. A disadvantage is the fact that important features may be missed if the data value changes rapidly from one grid sample point to the next and a classified region falls between them. Post-classification, or interpolation prior to classification, avoids this problem by classifying each interpolated element individually. This approach, however, comes at a greater computational cost since the table lookups occur much more frequently. Figure 2.4 shows the difference in image quality between a pre-classified volume renderer and a post-classified volume renderer. Figure 2.5 shows an example of pre-classification and post-classification volume rendering pipelines.

Designing a transfer function is difficult because the user must typically explore the space of data values and sometimes higher order information, such as gradient magnitude, with little dataset specific guidance. A number of researchers have investigated automatic transfer function generation. Design galleries [25] approach transfer function generation as a high dimensional problem space. They attempt to sample the problem space via a random heuristic and order the resulting images based on some similarity function. The result is a collection of clustered thumbnail images which the user selects and inspects. The down-side to this approach is that it is not based on information inherent in the data, and may cause the user to miss important details. Semi-automatic transfer function
Figure 2.4. A comparison of pre-classification versus post-classification volume renderings of the inner ear.

Figure 2.5. A comparison of pre-classification and pre-shading versus post-classification and post-shading volume rendering pipelines.
generation [29, 27] classifies the volumetric data based on data value and gradient magnitude, similar to Levoy’s method [39]. This method, however, utilizes second derivative information to select portions of the 2D transfer function space which are most likely to be boundaries. The user has access to a boundary emphasis variable which selects data values in the 2D transfer function that are within the specified signed distance to an ideal boundary. Unfortunately, this process may fail on volumes with very little differentiation in data values or very sharp boundaries. Also, the pre-process of generating histogram volumes for transfer function generation requires setting two free variables based on the preferred boundary model and material properties. Previous hardware volume rendering implementations only use 1D transfer functions, with the exception of the Volume Pro graphics card which allows a limited class of separable 2D transfer functions. 1D transfer functions can significantly limit the types of boundaries that can be visualized.

2.1.4 Shading

The most common shading model used in volume rendering applications is the Blinn-Phong surface shading model. This shading model approximates point light source illumination. Since it is a strictly local model, it does not include a visibility term or global illumination effects. It has three primary parts: an ambient term that represents the average incoming light from all directions, a diffuse term that represents the isotropically scattered direct lighting, and a specular term that represents the amount of light reflected toward the eye and the sharpness of the highlights. The surface normal required by this model is approximated as the normalized gradient of the scalar field at the sample being shaded. Volumes are typically shaded as though the light reaches a sample without being attenuated by the portion of the volume between it and the light. Adding a visibility term, or volumetric shadows, to this shading model is straightforward. The color of the interpolated, classified, and surface shaded sample is multiplied by one minus the opacity between it and the light source.

Just as the order in which reconstruction and classification occur is important, as is the order of reconstruction and shading. The difference is similar to Gouraud verses Phong shading on polygonal surfaces. A pre-shading volume renderer, see in Figure 2.5(a), evaluates the shading model at the grid samples and interpolates the illumination, just as Gouraud shading evaluates the shading model at the vertices of a triangle and interpolates the illumination across the polygon. A post-shading pipeline, shown in Figure 2.5(b),
interpolates the normals and evaluates the shading model for each reconstructed sample, just as Phong shading interpolates the normals across the polygon and shades each rasterized pixel. The difference between pre-shading and post-shading is not as dramatic as the difference between pre-classification and post-classification. In fact, pre-shading may be preferable to post-shading. Normals are de-normalized when they are interpolated, which means that a re-normalization step is required per-reconstructed sample. It is also more computationally expensive to evaluate the shading model per-reconstructed sample.

There is an additional subtlety to be considered when reconstructing the gradient of the scalar field, which is used as the normal in the shading model. Just as the reconstructed data value at a position between grid samples is estimated using a reconstruction kernel, the gradient of the reconstructed data field at that position is accurately estimated using the derivative of the reconstruction kernel. A pre-computed and interpolated gradient at that position is blurred when compared to the accurate gradient. It may, however, be preferable to use the less accurate blurred gradient as the normal, since gradient estimation is sensitive to noise and blurring is effectively a low-pass filter.

### 2.1.5 Compositing

Compositing is the final stage in the volume rendering pipeline. The model proposed by Levoy and Drebin et al. uses the under operator to composite interpolated, classified, and shaded samples taken from the front of the volume toward the back with reference to the eye. Max’s Particle Model shows how this operator can be derived from a physically based model [45]. The under operator is defined as:

\[ C_{out} = C_{old} + (1 - \alpha_{old}) \alpha_{in} C_{in} \]  
\[ \alpha_{out} = \alpha_{old} + (1 - \alpha_{old}) \alpha_{in} \]  

Similarly, if the volume is composited from back to front, with respect to the eye, the over operator is defined as:

\[ C_{out} = C_{in} \alpha_{in} + (1 - \alpha_{in}) C_{old} \]  
\[ \alpha_{out} = \alpha_{in} + (1 - \alpha_{in}) \alpha_{old} \]  

The color blend operations for both the under and over operators have \( C_{in} \) multiplied by its opacity \( \alpha_{in} \). It is for this reason that many implementations store colors pre-multiplied by their opacities. This is advantageous because it makes the color and alpha blend
operations match. If the sample rates change, the opacities must be scaled appropriately so that over all image quality is maintained. The formula for scaling alpha values from an initial sample rate $sr_{old}$ to a new one $sr_{new}$ is:

$$\alpha_{new} = 1 - (1 - \alpha_{old})^{sr_{old}/sr_{new}}$$ (2.5)

### 2.2 3D Texture Based Volume Rendering

A graphics hardware analog of the volume rendering pipeline proposed by Levoy and Drebin et al., 3D texture volume rendering [65], uses texture mapping hardware to approximate the software method. Quantized scalar data values are downloaded to texture memory, as 3D textures. Proxy geometry, such as view aligned planes or eye centered spherical shells [38], are assigned 3D texture coordinates. Figure 2.6 shows a two dimensional example of view aligned slicing compared to spherical shell slicing. Trilinear interpolation occurs in hardware as the polygons are textured. A pre-classification texture based volume renderer is implemented using a texture look up table prior to interpolation, where the lookup is based exclusively on the scalar data value. The scalar values are replaced with user assigned color and alpha values. The textured proxy geometry is then typically composited from back to front. Engel et al. showed how to implement a post-classification texture based volume renderer that significantly reduces the number of slices needed to adequately sample a scalar volume, while maintaining a high quality rendering, using a mathematical technique of pre-integration and hardware extensions such as dependent textures [15].

Most 3D texture implementations shade using only the ambient light portion of the Blinn-Phong shading model. A number of methods have been developed to utilize the diffuse and specular portions of this model as well. For instance, each scalar voxel can be

![Diagram](image)

(a) View aligned slicing   (b) Spherical shell slicing

**Figure 2.6.** View aligned slicing and spherical shell slicing of volume data.
recoded based on its data value and gradient direction in the preprocessing stage. Diffuse shading occurs during the classification stage via a modified texture look up table [17]. This technique can cause a number of artifacts in the final image including loss of data fidelity and poor shading due to an inadequate number of bitplanes in the texture look up table for the quantization of both gradient direction and data value. Other methods shade using hardware features such as OpenGL pixel textures and the OpenGL color matrix [61]. These methods, however, may limit the user to visualizing only a single diffuse shaded isosurface rather than the typical cloud like volume rendering. They may also require multiple rendering passes which can limit interactivity. Recent advancements in commodity graphics hardware have made a number of per-pixel surface shading options available [9, 22], which enable high quality surface based volume shading without loss of data fidelity at interactive frame rates. A technique proposed for computing volumetric shadows utilizes a secondary volume for storing the light intensity at each voxel [1]. This approach is adapted for graphics hardware from a similar technique proposed by Kajiya and Von Herzen [26]. Volumetric shadows stored using a 3D texture shadow map suffer from an artifact known as attenuation leakage. This is caused by the fact that trilinear interpolation effectively blurs the light contribution, and the net effect is volume surfaces which appear to occlude themselves.

2.3 Related Work

Direct volume rendering has developed to a solid level of maturity since key work was done nearly two decades ago [3, 13, 20, 26, 39, 57]. Although this thesis focuses on texture based volume rendering methods, there are a number of volume rendering implementations and techniques that deserve mention.

2.3.1 Ray Casting

Volumetric ray tracing, or ray casting [13, 39], is a direct extension of ray tracing for surfaces [64]. Rays are cast from the eye point, through the image plane and intersected with the volume elements. Sample values along the ray are trilinearly interpolated from the nearest volume elements, or voxels. Gradient estimation can either be done on a per voxel basis with each vector component trilinearly interpolated at the sample point, or it can be recomputed at each sample point. Classification is typically done for each sample point after interpolation using data value and gradient magnitude. Classification done before interpolation, or pre-classification, can introduce artifacts in the final image.
Methods have been introduced which utilize opacity weighted color interpolation to reduce these pre-classification artifacts [66]. Color is computed for each sample using the normalized gradient and light direction based on the Blinn-Phong shading model [2, 54]. Color and alpha values computed in the classification stage are composited in front to back order using Porter and Duff's under operator [55]. A great deal of effort has been put into this method to improve the reconstruction filter [5, 48], light transport model [58], and pipeline optimization [11].

2.3.2 Shear-Warp

The shear-warp method [37] can be used to accelerate volume rendering. It optimizes the pipeline at several stages. First, samples are only generated orthogonally along the major axes through a sheared version of the volume. This requires three copies of the data to be available in memory, one for each axis, so that the volume elements are always traversed in a cache coherent manner. All sample points are computed in a 2D slice before moving to the next slice. Bilinear interpolation is used within each 2D slice. Gradient estimation is computed as a pre-process for each data value and interpolated in the same manner as the sample points. Classification and shading proceed as in the ray casting method. Compositing is also done in front to back order. An entire slice is composited at a time. The resulting image is a sheared version of the volume. An image space warp is required as an additional step to make the volume appear correctly rotated, rather than sheared. This method also benefits from run-length encoding of the volume data, which allows large portions of the data to be rendered as though it were a single data value. This technique also incorporates a volumetric shadow rendering technique that utilizes a single 2D image buffer. The effectiveness and utility of this technique is limited since slicing directions must correspond to the major axes of the volume.

2.3.3 2D Texture Based Volume Rendering

A hardware implementation similar to the shear warp method uses axis aligned planes which are rendered with 2D textures [4]. 2D interpolation occurs in hardware. Classification is typically done via a texture look up table. Compositing is done in back to front order. This method also requires 3 copies of the data sliced along the major axes. Unfortunately, downloading a new copy of the texture to the graphics hardware when the view direction requires it can be time consuming and interfere with interactivity. For this reason, 3D textures are preferred over 2D, since in the 3D texture case, only one copy of
the data is required. The shading options available in the 3D texture method above are also applicable to this method.

2.3.4 Volume Pro

Another hardware implementation similar to the shear warp method is the VolumePro graphics board [53]. This implementation supports hardware 3D interpolation and gradient estimation. It shades volumes up to $512^3$ using the full Blinn-Phong shading model \textit{i.e.,} including ambient, diffuse and specular highlights. This card uses a second standard graphics card to perform the image space warp. The Volume Pro graphics board can achieve performance of 30 frames per second. Since this implementation is done entirely in hardware, however, it is difficult to make changes to the pipeline that are outside the hardware manufacturer’s specifications.

2.3.5 Splatting

Splatting [63] is another popular volume rendering method. This method treats each voxel as a sample point. Each sample is first classified and shaded; using both data value and the estimated gradient for computing color and alpha values. Since each sample is a voxel, interpolation is not required. Rather, the shaded sample point is convolved with a Gaussian or other reconstruction kernel to generate a contribution extent, or voxel footprint. Since each extent overlaps sample points besides those that are directly behind it, with reference to the eye location, all samples must be ordered correctly to assure proper compositing. Compositing proceeds either back to front or front to back. Since splatting essentially implements a pre-classification volume renderer, it can have arbitrary errors in its renderings. This is because the generation of the contribution extent is symmetric and ignores boundary locations. The use of the estimated gradient in classification is a naive attempt to address this artifact. Much of the current research in splatting is focused on correcting this problem [49, 50].

Texture splats [8] is a hardware implementation of splatting described above. Classification and shading for each voxel occurs in software. A collection of contribution extents is encoded and downloaded to 2D texture memory. Each voxel is rendered as a polygon with a size corresponding to the size of the reconstruction kernel used for the contribution extent. The polygon is textured using the appropriate extent texture. Rather than adding to the polygon’s color, the extent texture modulates the polygon’s color. As a result, each voxel appears blurred and overlaps its neighbors extent by some amount. This method
is easily extended to include texture glyphs for vector volume data visualization.

2.3.6 Parallel Volume Rendering

Parallelizing the volume rendering pipeline usually divides the task in image space or object space. Image space division usually requires complete data duplication for each rendering thread. Data duplication on shared memory architectures is usually not an issue since each thread can access a central data set. On distributed memory architectures, however, data duplication may not be practical or even possible for large data sets. Object space division requires that the data be bricked, rendered in full image space, and later composited to form the final image. Hybrids of image and data parallel volume rendering methods have also been explored [62]. A major bottleneck in any parallel process is inter-node communication [51]. In image parallel implementations, each thread renders its portion of the image space and sends it to an assembly thread. In data parallel implementations, an intermediate image needs to be composited for each pair of subvolumes. A hierarchical binary swap method has demonstrated considerable success as a compositing scheme for data parallel volume rendering on distributed memory architectures [42]. This method divides the compositing task up in image space such that each thread participates in each level of the compositing hierarchy. A number of methods have been introduced to improve data cell(voxel) traversal, data hierarchy, and cache coherence for ray tracing methods [52]. Efficient parallel hardware volume rendering methods exploit data parallelism and either composite in graphics hardware [67] or software [34]. Other platforms with potential for parallel hardware volume rendering use specialized ASICs and FPGAs on distributed systems [24].

2.3.7 General Light Transport

Most volumetric light transport models assume spherical particles with a density distribution that can be represented mathematically as a continuous function. The interaction of light with these particles is determined by optical properties such as absorption, emission, and scattering. Light transport models that accurately take into account these optical properties [26] are closely related to models of radiative transport from astrophysics and nuclear physics [6]. Such models are computationally expensive and may not produce the most appropriate images for scientific visualization, where a clear and informative image is far more important than a physically accurate one. A concise derivation of volumetric light transport equations for computer graphics and
their discrete approximations can be found in Max’s “Optical Models for Direct Volume Rendering” [45]. Most of the approximations presented in Max’s work have unbounded light intensities, which also make them undesirable for scientific visualization. Displaying unbounded light intensities, or high dynamic range images, requires an additional step known as tone mapping [59]. The image must be scanned to identify the maximum light intensity and then all pixels must be non-linearly scaled for display on relatively low dynamic range devices.
CHAPTER 3

PARALLEL HARDWARE VOLUME RENDERING

Hardware accelerated 3D texture based volume rendering is an approach that allows users to achieve interactive display rates for reasonably sized datasets. The size of interactive datasets is a function of the hardware’s available texture memory and fill rate. Current high end hardware implementations place an upper bound on dataset sizes of around 256MB. Consumer graphics cards support less than 128MB. In this chapter, a scalable, pipelined approach for rendering datasets which are larger than the limitations of a single graphics card is presented. This is done by taking advantage of multiple hardware rendering units and parallel software compositing.

The goals of TReX, a system for interactive volume rendering of large datasets, are to provide near interactive display rates for terabyte sized, time-varying, datasets and provide a low latency platform for volume visualization in immersive environments. 5 frames per second (fps) is considered to be near interactive rates for normal viewing environments, where immersive environments have a lower bound frame rate of 10 fps. While this is significantly below most virtual environment update rates, the user can successfully investigate extremely large datasets at this rate. Using TReX for virtual reality environments requires very low latency, around 50ms per frame, or 100ms per view update or stereo pair. To achieve lower latency renderings, either smaller portions of the volume are rendered on more graphics pipes or subsample the volume so that a graphics pipe renders fewer samples per frame.

3.1 TReX System Overview

The implementation presented in this chapter is a hybrid parallel software and hardware volume renderer. It was designed to meet target performance for near interactive display of large scale time-varying scientific datasets. Specifically, the system was designed to render full resolution time-varying data, such as the Raleigh-Taylor fluid flow
Figure 3.1. Time-varying Raleigh-Taylor fluid instability dataset (1024³).
dataset shown in Figure 3.1, at 5 fps on a 128-cpu, 16-pipe SGI Origin 2000 with IR-2 graphics hardware. While the limiting factor is the high performance I/O as described in Section 3.1.3, rendering is not the bottleneck as demonstrated by achieving over 5 fps for a static volume. For immersive environments, 10 frames per second are achieved with stereo pairs, albeit with half the sampling resolution as described in section 3.3. The primary difference from previous parallel hardware volume rendering work is the volume renderer’s interactivity on extremely large datasets. This is achieved by a design that utilizes all available hardware components: streaming time-varying datasets from a carefully designed, very high-performance I/O system, rendering with graphics pipes and compositing the results with processors. Interactively visualizing datasets is a critical first step in the analysis process, allowing users to rapidly gain a detailed understanding of their results. Since the hardware components can work independently, the parallel volume rendering process can be pipelined as shown in Figure 3.2. With overlapped stages, this pipeline allows us to achieve an overall performance which closely matches the performance of an individual graphics pipe. The details of the T Rex pipeline are presented in the following sections.

### 3.1.1 Pre-processing

This implementation requires an off-line preprocessing step in which the data is quantized from its native data type, commonly floating point, to either 8 bit or 12 bit unsigned integer data. The data is then divided, or bricked, into subvolumes with sizes matching the available texture memory on each graphics pipe. Note that most graphics hardware implementations currently require texture dimensions to be a power of two. The original volume may need to be super sampled or padded to match this power of two requirement. It is advisable to perform these operations on the original floating point data prior to bricking to avoid quantization errors and artifacts at subvolume interfaces. Interface artifacts are caused by boundary conditions in super sampling schemes.

### 3.1.2 The T Rex Pipeline

The rendering pipeline in T Rex has four stages. Each stage is a multi-threaded process capable of executing simultaneously with the other stages. A stage consists of two main parts: an event manager which handles communication, and a functional part which implements the task(s) of the thread. For a volume partitioned into $N$ subvolumes, T Rex will create $N$ readers and $N$ renderers. Note that it is not necessary for the
number of compositing threads to equal the number of subvolumes. The ideal number of compositor threads is a function of both image size and the number of images to be composited. A user interface thread is responsible for displaying the final composited image and sending user event messages to the other stages for supporting Immersive TRex and direct manipulation widgets. The details of each stage of the pipeline are discussed in the following sections.

3.1.3 Stage 1: Subvolume Reader

The first stage of the pipeline involves reading a time step from disk. TRex creates a separate reader thread for each of the subvolumes in a time step. Provided the data resides on a well striped RAID and direct I/O is available, the subvolumes can be read from disk in parallel at approximately 140MB/sec. Unfortunately this data rate is not fast enough to sustain the desired throughput of 5 fps for 10243 time-varying datasets but does provide a parallel approach to I/O that will work on most Silicon Graphics systems. For large time-varying datasets, a higher performance I/O design is required.

A method to achieve such high performance I/O is to build a file system that is customized for streaming volume data directly into system memory and then into texture memory. On the SGI Origin 2000 it is possible to do this by first co-locating the I/O controllers and graphics pipes, such that they share a common physical memory within the NUMA architecture. This configuration avoids the overhead associated with routing data through the system’s interconnection network, thus minimizing data transfer latency. In

---

1 Direct I/O avoids kernel interrupt overhead and is a feature available on SGI systems.

---

**Figure 3.2.** The TRex Pipeline
order to maximize performance, the I/O rates must match the approximate 300MB/sec
texture download rate of the InfiniteReality pipes. For 16 pipes operating in parallel
this is equivalent to a sustained rate of approximately 5GB/sec. These rates require
the use of a striped file system built using 64 dual fiber channel controllers and 2,304
individual disks. This is achieved by placing four fiber channel controllers next to each
pipe, with each fiber channel controller capable of 70MB/sec. This gives us a rate of
4 × 70MB/sec= 280MB/sec for each pipe. Assuming the best case performance can be
achieved, this rendering platform is still limited by the 280MB/sec rate of the I/O system.
In addition, it is also necessary to store subvolumes in contiguous blocks on disk to achieve
these data rates as well as minimize the amount of traffic over the system interconnect.
This is accomplished using SGI’s real time filesystem (RTFS). Initial benchmarks have
placed this configuration capable of approximately 4GB/sec. Researchers at the Advanced
Computing Laboratory at Los Alamos National Laboratory are continuing efforts to reach
the desired 5GB/sec rate.

32 Processor Origin 2000
Architecture

Node
Detail

Figure 3.3. SGI Origin 2000 architecture.
3.1.4 Stage 2: Render

The second stage of the pipeline renders subvolumes in parallel. A separate rendering thread manages each graphics pipe. These threads are initially responsible for creating OpenGL rendering windows. A renderer initializes multiple image buffers for simultaneous rendering with the other stages since the compositing stage and UI stage both rely on the image buffers as well. Due to the shared memory architecture of the Origin 2000 allocated memory may be located on any of the nodes within the machine. In order to improve performance, by reducing the latency of remote data transfers, the subvolume buffers used by the reader stages are specifically placed on the same node as the graphics hardware and I/O controllers. Figure 3.3 shows a diagram of the architecture for a 32 processor Origin 2000. The number of raw data buffers is dependent on the amount of time step buffering desired plus one for the simultaneous reading of data and downloading to texture memory.

Renderers receive a render message from the user interface. This message includes information about the current frame’s rotation, translation, scale, and sample rate. The OpenGL model-view matrix is set for the frame then geometry and volume data are rendered. Each renderer supports a simplistic scene graph which orders geometric primitives and subvolumes (if a pipe renders more than one). This volume rendering approach uses 3D textures with either view aligned slicing [65] or an approximation to Lamar’s concentric shells method [38]. The final image’s color and alpha values are read from the frame buffer and stored in memory. Finally, the renderer sends a message to the compositors that a new image is available, along with a pointer to the image buffer, and the subvolume’s distance from the eye point.

A texture lookup table encodes the transfer function, which assigns color and alpha values to the scalar texture elements. The user makes changes to the transfer function

![8 bit Data Values](image)

**Figure 3.4.** Texture Lookup Table. Note: the Alpha Band (top) has been multiplied by the Color Band (bottom) to show the resulting alpha weighted colors.
**Figure 3.5.** Texture Lookup Table with the Semi-Automatic generation of alpha mappings. The top band allows the user to select data values based on their distance from an ideal boundary detected in the volume. The middle band shows the generated alpha mapping multiplied by the color band.

by manipulating control points in color and alpha space as illustrated in Figure 3.4. The transfer function control is extended to allow the user to select a boundary distance function based on Knielmann’s semi-automatic transfer function generation [29]. The user can then select the appropriate portions of this automatic transfer function by manipulating control points in the alpha band (see Figure 3.5). If the transfer function or sample rate has changed since the last frame, the transfer function is updated and redownloaded to the graphics hardware prior to rendering a subvolume.

### 3.1.5 Stage 3: Compositor

Compositing begins once a completion message is received from each of the $N$ renderers. The message includes a pointer to the renderer’s shared memory image buffer and the subvolume’s distance from the eye point. A compositing thread is responsible for compositing $N$ images across a horizontal stripe of the final image. Composite order is determined by comparing the locations of the subvolumes and compositing back to front. As shown in Figure 3.6 when image $A$ is compositing over image $B$, the resulting image resides in $A$’s buffer. This eliminates the need for additional memory in the compositing stage. The first composite thread waits for the other compositors to finish before sending a message to the UI that a new image is ready for display.

The decision to use software compositing over hardware compositing was made because the compositing task is embarrassingly parallel and there was not a custom compositing network attached to this system. When using the existing graphics hardware for com-
posing, the cost of downloading \(N\) images is prohibitively time consuming. Also, the graphics hardware is the critical resource in this system. By utilizing available CPUs to composite the partial results, better scaling can be achieved versus a graphics hardware approach. Employing the available CPUs also allows compositing to overlap with the rendering of the next frame resulting in only a one frame latency rather than the multiple frame latency imposed by the Volumizer and Minnesota hardware based compositing systems.

### 3.1.6 Stage 4: User Interface

The user interface thread is responsible for managing the input from the user and sending messages that trigger other stages of the pipeline. If the user changes a viewing parameter such as rotation, scale, translation, or the transfer function, the UI sends a request to the renderers along with the new view parameters for the frame. When a message is received indicating that a new frame is available, the UI downloads the raw image data from the shared memory image buffer directly to the display's frame buffer.

In addition, the user has access to a quality parameter that adjusts the number of samples through the volume. This parameter is also set automatically. When the user is in an interaction state such as a pending rotation or translation, the sample rate is set lower to increase the frames per second. Once the interaction state is completed, i.e.
the user releases the mouse button, the quality parameter is set higher and the volume is rendered at a higher sample rate allowing automatic progressive refinement. When the window size is changed, the UI will send a resize request to the renderers. The slave display’s window size will be changed as well as the image buffers.

Figure 3.7 shows a schematic of the system. At the top, the volume is divided into subvolumes which are distributed to different renderers. Each renderer generates an image of its subvolume. These are labeled (subvolume, image-stripe). After all subvolumes are rendered, as shown in the middle of Figure 3.7, the compositor threads composite the appropriate subimages. As a final step, each compositor contributes its portion to the final image as shown at the bottom of Figure 3.7.

3.2 Discussion

Applications rendering semi-transparent objects, from back to front, generate new color values by using Equations 3.1 and 3.2 [55].

\[ c_{out} = a_{source} \times c_{source} + (1 - a_{source}) \times c_{target} \]  \hspace{1cm} (3.1)

\[ a_{out} = a_{source} + (1 - a_{source}) \times a_{target} \]  \hspace{1cm} (3.2)

Where \( a_{target} \) and \( c_{target} \) are the alpha and color values currently in the frame buffer. \( a_{source} \) and \( c_{source} \) are the incoming alpha and color values. \( a_{out} \) and \( c_{out} \) are the new alpha and color values to be written to the frame buffer.

Standard hardware implementations only allow the setting of one function which applies to both color and alpha channels. Since the equations for color and alpha are clearly different, color and alpha cannot be bended with Equation 3.1. Doing so would cause errors in the accumulated alpha value. Equations 3.3 and 3.4 demonstrate what happens when alpha compositing is treated the same as color compositing.

\[ c_{out} = a_{source} \times c_{source} + (1 - a_{source}) \times c_{target} \]  \hspace{1cm} (3.3)

\[ a_{out} = a_{source} \times a_{source} + (1 - a_{source}) \times a_{target} \]  \hspace{1cm} (3.4)

Notice that \( a_{source} \) is squared and then added to the complemented \( a_{source} \) times \( a_{target} \). This contrasts with Equation 3.2 and the error can not be easily corrected.
Figure 3.7. The entire system: renders at the top, compositors in the middle, final image at the bottom. The labels are (subvolume, image-stripe).
The solution is to pre-multiply the color values in the texture lookup table by their corresponding alpha values. The resulting Equations 3.5 and 3.6 match Equations 3.1 and 3.2 respectively, since $ca_i$ expands to $c_i \cdot a_i$.

$$c_{out} = ca_{source} + (1 - a_{source}) \times c_{target}$$

(3.5)

$$a_{out} = a_{source} + (1 - a_{source}) \times a_{target}$$

(3.6)

This correction is especially necessary when the alpha values are used at some latter time, such as compositing. For display on a single graphics pipe, the accumulated alpha value is not important, since only the incoming fragment’s alpha value is used in computing the color value, i.e. the alpha value in the frame buffer is never used for computing color.

The use of polygonal objects such as direct manipulation widgets and isosurfaces in conjunction with volume data requires us to take steps to insure that the geometry is composited with the volume correctly. A scene with geometry and volume composited correctly allows geometry to appear embedded in the volume. The type of geometric objects are limited to those that are fully opaque. Geometry is rendered first with depth test and depth write. Next the volume data is rendered from back to front with depth test only. This allows volume data to be rendered over geometric data but not behind. One difficulty of compositing subvolumes with geometric data is handling polygons that reside in two or more subvolumes or only partially in a subvolume. This can be solved by clipping geometry to planes corresponding to the faces of the subvolume which border other subvolumes. This requires at most six clipping planes for a subvolume which is completely surrounded by other subvolumes.

TReX takes advantage of several platform specific optimizations. Benchmarks on the IR graphics subsystem revealed that unsigned shorts were best for frame buffer reads and writes. Topology of the platform is also a concern especially in NUMA platforms like the Origin 2000. Data transfer latency can be significantly reduced by placing processes and their memory nearest the I/O devices that they use. This configuration avoids the overhead associated with routing data through the system’s interconnection network. A significant speed up was realized by placing the renderers and incoming data buffers near the IR pipe which they manage for the same reasons.
3.3 Immersive TRex

Using TRex in an immersive environment adds another level of complexity to the rendering pipeline. First, a stereo pair must be generated for each new viewpoint. This requires twice the fill rate as a monocular viewpoint. Achieving target frame rates requires that the number of samples each graphics pipe must process per-frame be decreased by half. One solution is to increase the number of graphics pipes, rendering threads, and compositors. This requires rebrickling the dataset so that smaller subvolumes are rendered on each pipe. Reducing the overall size of the dataset via subsampling is another option if additional graphics units are not available. Subsampling, however, has the side effect of blurring fine details. The number of compositor threads in either case needs to be increased to match the lower latency of the rendering stage.

Second, tracking devices are essential for creating an immersive environment. Typically, a separate daemon process communicates with the tracking device and reports position and orientation to a shared memory arena. A TRex VR session will create an additional thread for monitoring and reporting head and hand/device data to the UI. Since multiple tracking devices are available, such as Polhemus Fast Track (TM) and Ascension Flock of Birds (TM), and interaction devices may vary, the VR thread is responsible for mapping events from the current tracking and interaction devices to a set of events which the TRex UI understands.

New viewpoints are generated using parallel viewing directions with asymmetrical, or sheared, frustums (see Figure 3.8). Head orientation in semi-immersive environments such as the Responsive Workbench (TM Fakespace) is essentially disregarded. This is because eyes can be treated as points, and the portal to the virtual space (view surface) is fixed in the real space. View direction for semi-immersive environments is determined by the line from the eye point in real space which perpendicularly intersects the view surface plane. For fully immersive environments, such as those achieved with n - Visions Datavisor (TM) head mounted displays, eyes are still treated as points but head orientation is required to specify the virtual portal. View direction for fully immersive environments is specified by head orientation, and eyes are assumed to be looking forward.

View aligned slicing causes artifacts when the volume is close to the user and rendered with perspective projection. Lamar's spherical shell slicing reduces this problem by assuring that the volume is rendered with differential slices which are perpendicular to the line from the center of projection to the volume element being rendered. This
Figure 3.8. View Point construction for Semi-Immersive Environments (A), and Fully-Immersive Environments (B) for an arbitrary head orientation. Notice: View directions are parallel and perpendicularly intersect the view surface. The overlap region in B is variable on many head mounted displays.

approach uses an adaptive tessellation of the spherical shell. While coarse tessellations can cause artifacts, fine tessellations can cause significant latency in the rendering. The user is allowed to select a tessellation that is appropriate for the visualization.

3.4 TRex Widgets

Direct manipulation widgets [7] can improve the quality and productivity of interactive visualization. Widgets also allow the user to have a uniform experience when using either the desktop or an immersive environment.

The TRex widget sets were created using the Brown University widget paradigm. The widgets are object oriented, extendible entities which maintain state similar to a discrete finite automaton. They are based on simple parts such as spheres, bars, and sliders. Complex widgets are constructed from these sub-parts. Each sub-part represents some functionality of the widget. For instance, the bars which make up the boundary of a frame widget, when selected, would translate the whole frame, the spheres which bracket
the corners would scale the frame, sliders attached to the bars would alter some scalar value associated with the functionality of the widget.

To facilitate parallel rendering, a typical widget is broken into $N+1$ lightweight threads, where $N$ is the number of subvolumes in the session. A parent thread is responsible for handling events from the UI and communicating with the child threads. The $N$ child threads are responsible for rendering the widget and performing subvolume specific tasks. Each child thread is associated with a subvolume and is clipped to the half planes corresponding to the faces of the subvolume which border other subvolumes. Three custom widgets have been developed for use with TRex.

The color map widget places the transfer function in the scene with the volume being rendered. This provides a tighter coupling between the actual data values and their representation as a color value in the image. The color map widget is composed of three bands, one for color to data value mapping, one for opacity to data value mapping, and one for the semi-automatic generation of opacity mappings [29]. This widget also includes sliders, as can be seen in Figure 3.9(a), for manipulating the high quality sample rate, interactive sample rate, and opacity scaling.

A data probe widget allows the user to query the visualization for local quantitative information. This widget can be used to point to a region of the volume and automatically query the original data for the values at that location. This widget is particularly useful for studying data from physical simulations where the actual value at a location is of interest. Figure 3.9(c) shows a dataset employing the data probe widget.

The internal structure of volumetric data is often obscured by other portions of the volume. One method for revealing hidden information is to use clipping planes to remove the regions which occlude. For this purpose a widget was developed to allow the user to position and orient an arbitrary clipping plane in the scene. Because of the amorphous quality of some volume renderings, it is necessary to map a slice to the clipping plane with a different transfer function to make the clipped boundary apparent. One useful mapping is a simple data value to gray scale and a linear alpha ramp from low to high (see Figure 3.9(b)). This sort of mapping is particularly useful for radiology datasets where users are more accustomed to viewing slices, rather than the whole volume.
Figure 3.9. TRex Widgets, (a) demonstrates the Color Map and Clip Widgets, (c) demonstrates the Data Probe widget and shows intermediate renderings to the left.
CHAPTER 4

MULTI-DIMENSIONAL TRANSFER FUNCTIONS

Transfer function specification is arguably the most important task in volume visualization. While the transfer function’s role is simply to assign optical properties such as opacity and color to the data being visualized, the value of the resulting visualization will be largely dependent on how well these optical properties capture features of interest. Specifying a good transfer function can be a difficult and tedious task for several reasons. First, it is difficult to uniquely identify features of interest in the transfer function domain. Even though a feature of interest may be easily identifiable in the spatial domain, the range of data values that characterize the feature may be difficult to isolate in the transfer function domain due to the fact that other, uninteresting regions, may contain the same range of data values. Second, transfer functions can have an enormous number of degrees of freedom. Even simple 1D transfer functions using linear ramps require two degrees of freedom per control point, one for the position along the data value axis and another for the opacity or color intensity. Third, typical user interfaces do not guide the user in setting these control points based on dataset specific information. Without this type of information, the user must rely on trial and error. This kind of interaction can be especially frustrating since small changes to the transfer function can result in surprisingly large and unintuitive changes to the volume rendering.

Rather than classifying a sample based on a single scalar value, multi-dimensional transfer functions allow a sample to be classified based on a combination of values. Multiple data values tend increase the probability that a feature can be uniquely isolated in the transfer function domain, effectively providing a larger vocabulary for expressing the differences between structures in the dataset. These values are the axes of a multi-dimensional transfer function. Adding dimensions to the transfer function, however, greatly increases the degrees of freedom necessary for specifying a transfer function and the need for dataset specific guidance.
In the following sections, the application of multi-dimensional transfer functions is demonstrated for two distinct classes of data: scalar data and multivariate data. The scalar data application is focused on locating surface boundaries in a scalar volume. We motivate and describe the axes of the multi-dimensional transfer function for this type data. The use of multi-dimensional transfer functions for multivariate data is then described. Two examples are used, color volumes and meteorological simulations, to demonstrate the effectiveness of this class of transfer function.

### 4.1 Scalar Data

For scalar data, the gradient is a first derivative measure. As a vector, it describes the direction of greatest change. The normalized gradient is often used as the normal for surface based volume shading. The gradient magnitude is a scalar quantity which describes the local rate of change in the scalar field. For notational convenience, $f'$ is used to indicate the magnitude of the gradient of $f$, where $f$ is the scalar function representing the data:

$$f' = \| \nabla f \|$$  \hspace{1cm} (4.1)

This value is useful as an axis of the transfer function since it discriminates between homogeneous regions (low gradient magnitudes) and regions of change (high gradient magnitudes). This effect can be seen in Figure 4.1. Figure 4.1(a) shows a 1D histogram based on data value and identifies the three basic materials in the Chapel Hill CT Head; air (A), soft tissue (B), and bone (C). Figure 4.1(b) shows a log-scale joint histogram of data value versus gradient magnitude. Since materials are relatively homogeneous, their gradient magnitudes are low. They can be seen as the circular regions at the bottom of the histogram. The boundaries between the materials are shown as the arches; air and soft tissue boundary (D), soft tissue and bone boundary (E), and air and bone boundary (F). Each of these materials and boundaries can be isolated using a 2D transfer function based on data value and gradient magnitude. Figure 4.1(c) shows a volume rendering with the corresponding features labeled. The air/bone boundary, (F) in Figure 4.1 is a good example of a surface which cannot be isolated using a simple 1D transfer function. This type of boundary appears in CT datasets as the sinuses and mastoid cells. Figure 4.2 compares attempts at emphasizing the frontal and maxillary sinuses of the Visible Male CT using a 1D transfer function and a 2D transfer function.
Figure 4.1. Material and boundary identification of the Chapel Hill CT Head with data value alone (a) versus data value and gradient magnitude \( (f') \), seen in (b). The basic materials captured by CT, air (A), soft tissue (B), and bone (C) can be identified using a 1D transfer function as seen in (a). 1D transfer functions, however, cannot capture the complex combinations of material boundaries; air and soft tissue boundary (D), soft tissue and bone boundary (E), and air and bone boundary (F) as seen in (b) and (c).
Figure 4.2. The frontal and maxillary sinuses of the Visible Male CT. While a 1D transfer function can show the sinuses along with the skin, it cannot capture them in isolation. Only a higher dimensional transfer function, in this case a 2D transfer function using data value and gradient magnitude, can uniquely classify them.

Often, the arches that define material boundaries in a 2D transfer function overlap. In some cases this overlap prevents a material from being properly isolated in the transfer function. This effect can be seen in the circled region of the 2D data value/gradient magnitude joint histogram of the human tooth CT in Figure 4.3(a). The background/dentin boundary (F) shares the same ranges of data value and gradient magnitude as portions of the pulp/dentin (E) and the background/enamel (H) boundaries. When the background/dentin boundary (F) is emphasized in a 2D transfer function, the boundaries (E) and (H) are erroneously colored in the volume rendering, as seen in Figure 4.3(c). A second derivative measure enables a more precise disambiguation of complex boundary configurations such as this. Some edge detection algorithms (such as Marr-Hildreth [43]) locate the middle of an edge by detecting a zero-crossing in a second derivative measure, such as the Laplacian. A more accurate but computationally expensive measure is computed, the second directional derivative along the gradient direction, which involves the Hessian (H), a matrix of second partial derivatives. $f''$ is used to indicate this second derivative.
\[
    f'' = \frac{1}{\|\nabla f\|^2} (\nabla f)^T \mathbf{H} \nabla f
\]  

(4.2)

More details on these measurements can be found in previous work on semi-automatic transfer function generation [28, 29]. Figure 4.3(b) shows a joint histogram of data value verses this second directional derivative. Notice that the boundaries (E), (F), and (G) no longer overlap. By reducing the opacity assigned to non-zero second derivative values, the background/dentin boundary can be rendered in isolation, as seen in Figure 4.3(d). The relationship between data value, gradient magnitude, and the second directional derivative is made clear in Figure 4.4. Figure 4.4(a) shows the behavior of these values along a line through an idealized boundary between two homogeneous materials (inset). Notice that at the center of the boundary, the gradient magnitude is high and the second derivative is zero. Figure 4.4(b) shows the behavior of the gradient magnitude and second derivative as a function of data value. This shows the curves as they appear in a joint histogram or a transfer function.

**Figure 4.3.** Material and boundary identification of the human tooth CT with data value and gradient magnitude \((f')\), seen in (a), and data value and second derivative \((f'')\), seen in (b). The background/dentin boundary (F) cannot be adequately captured with data value and gradient magnitude alone. (c) shows the results of a 2D transfer function designed to show only the background/dentin (F) and dentin/enamel boundaries (G). The background/enamel (H) and dentin/pulp (E) boundaries are erroneously colored. Adding the second derivative as a third axis to the transfer function disambiguates the boundaries. (d) shows the results of a 3D transfer function that gives lower opacity to non-zero second derivative values.
Figure 4.4. The behavior of primary data value \(f\), gradient magnitude \(f'\), and the second directional derivative \(f''\) as a function of position (a) and as a function of data value (b).

4.2 Multivariate data

Multivariate data contains, at each sample point, multiple scalar values that represent different simulated or measured quantities. Multivariate data can come from numerical simulations that calculate a list of quantities at each timestep, or from medical scanning modalities such as MRI, which can measure a variety of tissue characteristics, or from a combination of different scanning modalities, such as MRI, CT, and PET. Multi-dimensional transfer functions are an obvious choice for volume visualization of multivariate data, since different data values can be assigned to the different axes of the transfer function. It is often the case that a feature of interest in these datasets cannot be properly classified using any single variable by itself. In addition, a kind of first derivative can be computed from the multivariate data in order to create more information about local structure. As with scalar data, the use of a first derivative measure as one axis of the multi-dimensional transfer function can increase the specificity for isolating and visualizing different features in the data.

One example of data that benefits from multi-dimensional transfer functions is volumetric color data. A number of volumetric color datasets are available, such as the Visible Human Project’s RGB data. The process of acquiring color data by cryosection is becoming common for the investigation of anatomy and histology. In these datasets, the differences in materials are expressed by their unique spectral signature. A multi-dimensional transfer function is a natural choice for visualizing this type of data. Opacity can be assigned to different positions in the 3D RGB color space. Figure 4.5(a) shows
(a) Histograms of the Visible Male RGB dataset.

(b) The white (A) and gray (B) matter of the brain.

(c) The muscle and connective tissues (C) of the head and neck, showing skin (D) for reference.

**Figure 4.5.** The Visible Male RGB (color) data. The opacity is set using a 3D transfer function, color is taken directly from the data. The histogram (a) is visualized as projections on the primary planes of the RGB color space.
Figure 4.6. A kidney from the Visible Male RGB dataset. The renal vein is labeled (E). A clipping plane reveals internal structure (right).

A joint histogram of the RGB color data for the Visible Male; regions of this space that correspond to different tissues are identified. Regions (A) and (B) correspond to the fatty tissues of the brain, white and gray matter, as seen in Figure 4.5(b). In this visualization, the transition between white and grey matter is intentionally left out to better emphasize these materials and to demonstrate the expressivity of the multi-dimensional transfer function. Figure 4.5(c) shows a visualization of the color values that represent the muscle structure and connective tissues (C) of the head and neck with the skin surface (D) given a small amount of opacity for context. In both of these figures, a slice of the original data is mapped to the surface of the clipping plane for reference. Figure 4.6 shows a visualization of the kidney from the Visible Male RGB data.

Our choice of RGB for the transfer function axes is rather arbitrary; it is simply the most direct use of the color data. Other natural choices for color representation are the HSV or HLS spaces, or a CIE colorimetric space, if calibration data is available. Any color space is fine as long as it is possible to convert to RGB for display. It is important to note, however, that materials which are indistinguishable in the RGB color space will also be indistinguishable in any other color space. The choice of color space representation for the transfer function should be made on the basis of ease of use. Some color spaces, such as HSV, are better geared for human navigation. Our experience, however, has shown that tissue colors in cryosection are sometimes not what we expect. This can be seen in Figure 4.6, where the color in the renal vein (E) is essentially black, rather than red, as blood is expected to be. For this reason, our exploration of this dataset has been largely guided by probing and dual-domain interaction, which are described in the next chapter. It is impractical to manipulate the transfer function in the full 3D space that it
defines. Instead, the transfer function is only manipulated using two axes at a time. The placement of classified regions is very similar to that shown in Figure 4.5(a), where each classified region is represented as a projection onto two of the transfer function axes.

The kind of first derivative computed in multivariate data is based on previous work in color image segmentation [12, 56, 10]. While the gradient magnitude in scalar data represents the magnitude of local change at a point, an analogous first derivative measure in multivariate data captures the total amount of local change, across all the data components. This derivative has proven useful in color image segmentation because it allows a generalization of gradient-based edge detection. In our system, this first derivative measure is used as one axis in the multi-dimensional transfer function in order to isolate and visualize different regions of a multivariate volume according to the amount of local change, analogous to our use of gradient magnitude for scalar data.

If the dataset is represented as a multivariate function \( f(x, y, z) : \mathbb{R}^d \to \mathbb{R}^p \), so that

\[
f(x, y, z) = (f_1(x, y, z), f_2(x, y, z), \ldots, f_m(x, y, z))
\]

then the derivative \( \mathbf{D}f \) is a matrix of first partial derivatives:

\[
\mathbf{D}f = 
\begin{bmatrix}
\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\
\frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\
\vdots & \vdots & \vdots \\
\frac{\partial f_m}{\partial x} & \frac{\partial f_m}{\partial y} & \frac{\partial f_m}{\partial z}
\end{bmatrix}
\]

By multiplying \( \mathbf{D}f \) with its transpose, a \( 3 \times 3 \) tensor \( \mathbf{G} \) can be formed, which captures the directional dependence of total change:

\[
\mathbf{G} = (\mathbf{D}f)^T \mathbf{D}f
\]  

(4.3)

In the context of color edge detection [12, 56, 10], this matrix (specifically, its two-dimensional analog) is used as the basis of a quadratic function of direction \( \mathbf{n} \), which Cumani [10] terms the squared local contrast in direction \( \mathbf{n} \):

\[
S(\mathbf{n}) = \mathbf{n}^T \mathbf{G} \mathbf{n}
\]

\( S(\mathbf{n}) \) can be analyzed by finding the principal eigenvector (and associated eigenvalue) of \( \mathbf{G} \) to determine the direction \( \mathbf{n} \) of greatest local contrast, or fastest change, and the
magnitude of that change. Our experience, however, has been that in the context of multi-dimensional transfer functions, it is sufficient (and perhaps preferable) to simply take the L2 norm of $G$, $\|G\|$, which is the square root of the sum of the squares of the individual matrix components. As the L2 norm is invariant with respect to rotation, this is the same as the L2 norm of the three eigenvalues of $G$, motivating our use of $\|G\|$ as a directionally independent (and rotationally invariant) measure of local change. Other work on volume rendering of color data has used a non-rotationally invariant measure of $G$ [14]. Since it is sometimes the case that the dynamic range of the individual channels ($f_i$) differ, the ranges of each channel's data value can be normalized to be between zero and one. This allows each channel to have an equal contribution in the derivative calculation.

Meteorological simulations are a good example of datasets with features that can only be identified using a combination of data values [31], and which additionally benefit from using $\|G\|$ as an axis in the multi-dimensional transfer function. Air masses, for instance, are a phenomenon described primarily by differences in both temperature and humidity. The interfaces of these air masses, or fronts, are responsible for the formation of mid-latitude storms. In particular, cold fronts can produce severe weather including showers, thunderstorms, hail, high winds, and tornados. Naturally, the precise identification of these fronts are of interest to meteorologists. Figure 4.7 contains the results from a numerical meteorological simulation that uses a forcing function from measured data. Figure 4.7(a) is a surface map of the simulation domain for reference. Figure 4.7(b) shows the results of an expert frontal analysis using a technique which overlays slices through different scalar values, $(f_1, f_2, etc.)$, of the dataset, similar to those shown in Figures 4.7(c) and 4.7(d). This type of analysis is difficult because the expert must create a mental image of frontal behavior based on these scalar visualizations. The task is greatly simplified by visualizing the data based on its unique combination of data values, in this case temperature and humidity. The frontal regions were identified by probing in the spatial domain, seen as the dotted line in Figure 4.7(f) and visualizing the data values in the transfer function domain, seen in Figure 4.7(e). While the frontal region is identified as (A) in Figure 4.7(e), the visualization is clearer when the regions that correspond to the air masses that meet at these fronts are shown, identified as (B) and (C) in Figures 4.7(e) and 4.7(f). The rendering on the left in Figure 4.7(f) shows the air masses; the image on the right uses a similar transfer function, but excludes regions
with low \( \| G \| \) values. Notice that the interfaces, or frontal regions, of these air masses are emphasized.

4.3 Discussion

Using multi-dimensional transfer functions heightens the importance of densely sampling the voxel data in rendering. With each new axis in the transfer function, there is another dimension along which neighboring voxels can differ. It becomes increasingly likely that the data sample points at the corners of a voxel straddle an important region of the transfer function (such as a region of high opacity) instead of falling within it. Thus, in order for the boundaries to be rendered smoothly, the distance between view-aligned sampling planes through the volume must be very small. Most of the figures in this thesis were generated with sampling rates of about 3 to 6 samples per voxel. At this sample rate, frame updates can take nearly a second for a moderately sized \((256 \times 256 \times 128)\) shaded and shadowed volume. For this reason, we lower the sample rate during interaction, and re-render at the higher sample rate once an action is completed. During interaction, the volume rendered surface will appear coarser, but the surface size and location are usually readily apparent. Thus, even with lower volume sampling rates during interaction, the rendered images are effective feedback for guiding the user in transfer function exploration.

A benefit of using multi-dimensional transfer functions is the ability to use a “default” transfer function which is produced without any user interaction. Given our interest in visualizing the boundaries between materials, this was achieved by assigning opacity to high gradient magnitudes and, in the case of scalar data, low-magnitude second derivatives, regardless of data value, while varying hue along the data value axis. This default transfer function is intended only as a starting point for further modification with the widgets, but often it succeeds in depicting the main structures of the volume, as seen in Figure 4.8. Other application areas for volume rendering may need different variables for multi-dimensional transfer functions, with their own properties governing the choices for default settings.

While multi-dimensional transfer functions are quite effective for visualizing material boundaries, we have also found them to be useful for visualizing the materials themselves. For instance, if we attempt to visualize the dentin of the Human Tooth CT using a 1D transfer function, we erroneously color the background/enamel boundary, seen in Figure 4.9(a). The reason for this can be seen in Figure 4.3(a), where the range of data...
Figure 4.7. Frontal zones of a numerical weather simulation. (a) shows a reference map of the simulation domain. (b) shows the results of an expert analysis using scalar data visualizations similar to (c) and (d). (c) and (d) are slices through the dataset with a spectral color map. (e) shows a 2D log-scale joint histogram of temperature versus humidity. Region (A) shows the ranges of these data values that represent a mid-latitude front, (B) identifies the warm air mass, (C) identifies the cold air mass. (f) shows a volume rendering using a 3D transfer function which emphasizes regions (B) and (C). \(\|G\|\) from Equation 4.3 is used as the third axis of the transfer function for the rendering on the right to emphasize the portions of the air masses near the front. The dotted line shows a path through the dataset, the values along this line are shown in (e).
Figure 4.8. A default transfer function for scalar data applied to the Chapel Hill CT. Hue varies along the data value axis and opacity varies along the gradient magnitude axis. A clipping plane reveals internal structure (right).

values which define the background/enamel boundary overlap with the dentin’s data values. We can easily correct this erroneous coloring with a 2D transfer function that only gives opacity to lower gradient magnitudes. This can be seen in Figure 4.9(b).
Figure 4.9. The dentin of the Human Tooth CT. (a) shows that a 1D transfer function, simulated by assigning opacity to data values regardless of gradient magnitude, will erroneously color the background/enamel boundary. A 2D transfer function, shown in (b) can avoid assigning opacity to the range of gradient magnitudes that define this boundary.
CHAPTER 5

INTERACTION AND TOOLS

While adding dimensions to the transfer function enhances a users ability to isolate features of interest in a dataset, it tends to make the already unintuitive space of the transfer function even more difficult to navigate. This difficulty can be considered in terms of a conceptual gap between the spatial and transfer function domains. The spatial domain is the familiar 3D space for geometry and the volume data being rendered. The transfer function domain, however, is more abstract. Its dimensions are not spatial (i.e. the ranges of data values), and the quantities at each location are not scalar (i.e. opacity and three colors). It can be very difficult to determine the regions of the transfer function that correspond to features of interest, especially when a region is very small. Thus, to close this conceptual gap, new interaction techniques must be developed, which permit interaction in both domains simultaneously. A suite of direct manipulation widgets provide the tools for such interactions. Figure 5.1 shows the various direct manipulation widgets as they appear in a system, known as Simian, which was designed to guide the user toward setting high quality multi-dimensional transfer functions.

In a typical session with this system, the user creates a transfer function using a natural process of exploration, specification, and refinement. Initially, the user is presented with a volume rendering using a pre-determined transfer function that is likely to bring out some features of interest. This can originate with an automated transfer function generation tool [28], or it could be the default transfer function described earlier in Section 4.3. The user would then begin exploring the dataset.

Exploration is the process by which a user familiarizes him or herself with the dataset. A clipping plane can be moved through the volume to reveal internal structures. A slice of the original data can be mapped to the clipping plane, permitting a close inspection of the entire range of data values. Sample positions are probed in the spatial domain and their values, along with values in a neighborhood around that point, are visualized in the transfer function domain. This feedback allows the user to identify the regions of
Figure 5.1. The direct manipulation widgets.
the transfer function that correspond to potential features of interest, made visible by
the default transfer function or the sliced data. Once these regions have been identified,
the user can then begin specifying a custom transfer function.

During the specification stage, the user creates a rough draft of the desired transfer
function. While this can be accomplished by manually adding regions to the transfer
function, a simpler method adds opacity to the regions in the transfer function at and
around locations queried in the spatial domain. That is, the system can track, with a
small region of opacity in the transfer function domain, the data values at the user-selected
locations, while continually updating the volume rendering. This visualizes, in the spatial
domain, all other voxels with similar transfer function values. If the user decides that an
important feature is captured by the current transfer function, he or she can add that
region into the transfer function and continue querying and investigating the volume.

Once these regions have been identified, the user can refine them by manipulating
control points in the transfer function domain to better visualize features of interest.
An important feature of our system is the ability to manipulate portions of the transfer
function as discrete entities. This permits the modification of regions corresponding to a
particular feature without affecting other classified regions.

Finally, this is an iterative process. A user continues the exploration, specification, and
refinement steps until they are satisfied that all features of interest are made visible. In
the remainder of this section we will introduce the interaction modalities used in the ex-
ploration and specification stages and briefly describe the individual direct manipulation
widgets.

5.1 Probing and Dual-Domain Interaction

The concept of probing is simple: the user points at a location in the spatial domain
and visualizes the values at that point in transfer function domain. We have found this
feedback to be essential for making the connection between features seen in the spatial
domain and the ranges of values that identify them in the transfer function domain.
Creating the best transfer function for visualizing a feature of interest is only possible
with an understanding of the behavior of data values at and around that feature. This is
especially true for multi-dimensional transfer functions where a feature is described by a
complex combination of data values. The value of this dataset-specific guidance can be
further enhanced by automatically setting the transfer function based on these queried
values.

In a traditional volume rendering system, setting the transfer function involves moving the control points (in a sequence of linear ramps defining color and opacity), and then observing the resulting rendered image. That is, interaction in the transfer function domain is guided by careful observation of changes in the spatial domain. We prefer a reversal of this process, in which the transfer function is set by direct interaction in the spatial domain, with observation of the transfer function domain. Furthermore, by allowing interaction to happen in both domains simultaneously, the conceptual gap between them is significantly lessened, effectively simplifying the complicated task of specifying a multi-dimensional transfer function to pointing at a feature of interest. We use the term “dual-domain interaction” to describe this approach to transfer function exploration and generation.

The top of Figure 5.2 illustrates the specific steps of dual-domain interaction. When a position inside the volume is queried by the user with the data probe widget (a), the values associated with that position (multivariate values, or the data value, first and second derivative) are graphically represented in the transfer function widget (b). Then, a small region of high opacity (c) is temporarily added to the transfer function at the data values determined by the probe location. The user has now set a multi-dimensional transfer function simply by positioning a data probe within the volume. The resulting rendering (d) depicts (in the spatial domain) all the other locations in the volume which share values (in the transfer function domain) with those at the data probe tip. If the features rendered are of interest, the user can copy the temporary transfer function to the permanent one (e), by, for instance, tapping the keyboard space bar with the free hand. As features of interest are discovered, they can be added to the transfer function quickly and easily with this type of two-handed interaction. Alternately, the probe feedback can be used to manually set other types of classification widgets (f), which are described later. The outcome of dual-domain interaction is an effective multi-dimensional transfer function built up over the course of data exploration. The widget components which participated in this process can be seen in the bottom of Figure 5.2, which shows how dual-domain interaction can help volume render the CT tooth dataset. The remainder of this section describes the individual widgets and provides additional details about dual-domain interaction.
User moves probe in volume

Position is queried, and values displayed in transfer function

Queried region can be permanently set in transfer function

Changes are observed in rendered volume

User sets transfer function by hand

Region is temporarily set around value in transfer function

Figure 5.2. Dual-Domain Interaction
5.2 Data Probe Widget

The data probe widget is responsible for reporting its tip’s position in volume space and its slider sub-widget’s value. Its pencil-like shape is designed to give the user the ability to point at a feature in the volume being rendered. The other end of the widget orients the widget about its tip. When the volume rendering’s position or orientation is modified, the data probe widget’s tip tracks its point in volume space. A natural extension is to link the data probe widget to a haptic device, such as the SensAble PHANTOM, which can provide a direct 3D location and orientation [47].

5.3 Clipping Plane Widget

The clipping plane is responsible for reporting its orientation and position to the volume renderer, which handles the actual clipping when it draws the volume. In addition to clipping, the volume widget will also map a slice of the data to the arbitrary plane defined by the clip widget, and blend it with the volume by a constant opacity value determined by the clip widget’s slider. It is also responsible for reporting the spatial position of a mouse click on its clipping surface. This provides an additional means of querying positions within the volume, distinct from the 3D data probe. The balls at the corners of the clipping plane widget are used to modify its orientation, and the bars on the edges are used to modify its position.

5.4 Transfer Function Widget

The main role of the transfer function widget is to present a graphical representation of the transfer function domain, in which feedback from querying the volume (with the data probe or clipping plane) is displayed, and in which the transfer function itself can be set and altered. The balls at the corners of the transfer function widget are used to resize it, as with a desktop window, and the bars on the edges are used to translate its position. The inner plane of the frame is a polygon texture-mapped with the lookup table containing the transfer function. A joint histogram of data, seen with the images in Chapter 4, can also be blended with the transfer function to provide valuable information about the behavior and relationship of data values in the transfer function domain.

The data values at the position queried in the volume (either via the data probe or clipping plane widgets) are represented with a small ball in the transfer function widget. In addition to the precise location queried, the eight data sample points at the corners of the voxel containing the query location are also represented by balls in the transfer
function domain, and are connected together with edges that reflect the connectivity of the voxel corners in the spatial domain. By “re-projecting” a voxel from the spatial domain to a simple graphical representation in the transfer function domain, the user can learn how the transfer function variables (data values at each sample point) are changing near the probe location. The values for the third, or unseen axis, are indicated by coloring the balls. For instance, with scalar data, second derivative values which are negative, zero, or positive are represented by blue, white, and yellow balls, respectively. When the projected points form an arc, with the color varying through these assigned colors, the probe is at a boundary in the volume as seen in Figure 5.1. When the re-projected data points are clustered together, the probe is in a homogeneous region. As the user gains experience with this representation, he or she can learn to “read” the re-projected voxel as an indicator of the volume characteristics at the probe location.

5.5 Classification Widgets

In addition to the process of dual-domain interaction described above, transfer functions can also be created in a more manual fashion by adding one or more classification widgets to the main transfer function window. Classification widgets are designed to identify regions of the transfer function as discrete entities. Each widget type has control points which modify its position or size. Optical properties, such as opacity and color are modified by selecting the widgets inner surface. The opacity and color contributions from each classification widget are blended together to form the transfer function. We have developed two types of classification widget: triangular and rectangular.

The triangular classification widget, shown in Figures 4.2, 5.1, 5.2, and 6.2, is based on Levoy’s “isovalue contour surface” opacity function [39]. The widget is an inverted triangle with a base point attached to the horizontal data value axis. The triangle’s size and position are adjusted with control points. There are an upper and lower threshold for the gradient magnitude, as well as a shear. Color is constant across the widget; opacity is maximal along the center of the widget, and it linearly ramps down to zero at the left and right edges.

The triangular classification widgets are particularly effective for visualizing surfaces in scalar data. More general transfer functions, for visualizing data which may not have clear boundaries, can be created with the rectangular classification widget. The rectangular region spanned by the widget defines the data values which receive opacity and color.
Like the triangular widget, color is constant, but the opacity is more flexible. It can be constant, or fall off in various ways: quadratically as an ellipsoid with axes corresponding to the rectangle’s aspect ratio, or linearly as a ramp, tent, or pyramid.

As noted in the description of the transfer function widget, even when a transfer function has more than two dimensions, only two dimensions are shown at any one time. For 3D transfer functions, classification widgets are shown as their projections onto the visible axes. In this case, a rectangular classification widget becomes a box in the 3D domain of the transfer function. Its appearance to the user, however, as 2D projections, is identical to the rectangular widget. When the third axis of the transfer function plays a more simplified role, interactions along this axis are tied to sliders seen along the top bar of the transfer function. For instance, since our research on scalar data has focused on visualizing boundaries between material regions, we have consistently used the second derivative to emphasize the regions where the second derivative magnitude is small or zero. Specifically, maximal opacity is always given to zero second derivative, and decreases linearly towards the second derivative extremal values. How much the opacity changes as a function of second derivative magnitude is controlled with a single slider, which we call the “boundary emphasis slider.” With the slider in its left-most position, zero opacity is given to extremal second derivatives; in the right-most position, opacity is constant with respect to the second derivative. We have employed similar techniques for manipulating other types of third axis values using multiple sliders.

While the classification widgets are usually set by hand in the transfer function domain, based on feedback from probing and re-projected voxels, their placement can also be somewhat automated. This further reduces the difficulty of creating an effective higher dimensional transfer function. The classification widget’s location and size in the transfer function domain can be tied to the distribution of the re-projected voxels determined by the data probe location. For instance, the rectangular classification widget can be centered at the transfer function values interpolated at the data probe’s tip, with the size of the rectangle controlled by the data probe’s slider. The triangular classification widget can be located horizontally at the data value queried by the probe, with the width and height determined by the horizontal and vertical variance in the re-projected voxel locations. This technique produced the changes in the transfer function for the sequence of renderings in Figure 5.2.
5.6 Shading Widget

The shading widget is a collection of spheres which can be rendered in the scene to indicate and control the light direction and color. Fixing a few lights in view space is generally effective for renderings, therefore changing the lighting is an infrequent operation.

5.7 Color Picker Widget

The color picker is an embedded widget which is based on the hue-lightness-saturation (HLS) color space. Interacting with this widget can be thought of as manipulating a sphere with hues mapped around the equator, gradually becoming black at the top, and white at the bottom. To select a hue, the user moves the mouse horizontally, rotating the ball around its vertical axis. Vertical mouse motion tips the sphere toward or away from the user, shifting the color towards white or black. Saturation and opacity are selected independently using different mouse buttons with vertical motion. While this color picker can be thought of as manipulating an HLS sphere, no geometry for this is rendered. Rather, the triangular and rectangular classification widgets embed the color picker in the polygonal region which contributes opacity and color to the transfer function domain. The user specifies a color simply by clicking on that object, then moving the mouse horizontally and vertically until the desired hue and lightness are visible. In most cases, the desired color can be selected with a single mouse click and gesture.

5.8 Discussion

A further benefit of dual-domain interaction is the ability to create feature-specific multi-dimensional transfer functions which would be extremely difficult to produce by manual placement of classification widgets. If a feature can be visualized in isolation with only a very small and accurately placed classification widget, the best way to place the widget is via dual-domain interaction. This is the case for visualizing different soft tissues in CT data, such as the white matter of the brain in the Visible Male CT, shown in Figure 5.3.

Dual-domain interaction has utility beyond setting multi-dimensional transfer functions. Dual-domain interaction also helps answer other questions about the limits of direct volume rendering for displaying specific features in the data. For example, the feedback in the transfer function domain can show the user whether a certain feature of interest detected during spatial domain interaction is well-localized in the transfer
Figure 5.3. The brain of the Visible Male CT. The transfer functions were created using dual-domain interaction. A detail region shows how small the region that identifies this subtle feature is in the transfer function domain.

function domain. If re-projected voxels from different positions, in the same feature, map to widely divergent locations in the transfer function domain, then the feature is not well-localized, and it may be hard to create a transfer function which clearly visualizes it. Similarly, if probing inside two distinct features indicates that the re-projected voxels from both features map to the same location in the transfer function domain, then it may be difficult to selectively visualize one or the other feature.
CHAPTER 6

RENDERING AND HARDWARE

The quality of interaction and exploration described in this thesis would not be possible without the use of modern graphics hardware. The implementation of multi-dimensional transfer functions relies heavily on an OpenGL extension known as dependent texture reads. This extension can be used for both classification and shading. In this section, modifications to the classification portion of the traditional 3D texture-based volume rendering pipeline are described. Also described are methods for adding interactive volumetric shading and shadows to the pipeline.

The system supports volumes that are stored as 3D textures with one, two, or four values per texel. This is due to memory alignment restrictions of graphics hardware. Volumes with three values per sample utilize a four value texture, where the fourth value is simply ignored. Volumes with more than four values per sample could be constructed using multiple textures.

6.1 Dependent Texture Reads

Dependent texture reads are a hardware extension that is a similar but more efficient implementation of a previous extension known as pixel texture [15, 23, 46, 61]. Dependent texture reads and pixel texture are names for operations which use color fragments to generate texture coordinates, and replace those color fragments with the corresponding entries from a texture. This operation essentially amounts to an arbitrary function evaluation with up to three variables via a lookup table. If we were to perform this operation on an RGB fragment, each channel value would be scaled between zero and one, and these new values would then be used as texture coordinates of a 3D texture. The color values produced by the 3D texture lookup replace the original RGB values. Nearest neighbor or linear interpolation can be used to generate the replacement values.

The ability to scale and interpolate color channel values is a convenient feature of the hardware. It allows the number of elements along a dimension of the texture containing
the new color values to differ from the dynamic range of the component that generated the texture coordinate. Without this flexibility, the size of a 3D dependent texture would be prohibitively large.

6.2 Classification

Dependent texture reads are used for the transfer function evaluation. Data values stored in the color components of a 3D texture are interpolated across some proxy geometry, a plane for instance. These values are then converted to texture coordinates and used to acquire the color and alpha values in the transfer function texture per-pixel in screen space. For eight bit data, an ideal transfer function texture would have 256 color and alpha values along each axis. For 3D transfer functions, however, the transfer function texture would then be $256^3 \times 4$ bytes. Besides the enormous memory requirements of such a texture, the size also affects how fast the classification widgets can be rasterized, thus affecting the interactivity of transfer function updates. We therefore limit the number of elements along an axis of a 3D transfer function based on its importance. For instance, with scalar data, the primary data value is the most important, the gradient magnitude is secondary, and the second derivative serves an even more tertiary role.

For this type of multi-dimensional transfer function, we commonly use a 3D transfer function texture with dimensions $256 \times 128 \times 8$ for data value, gradient magnitude, and second derivative respectively. 3D transfer functions can also be composed separately as a 2D and 1D transfer function. This means that the total size of the transfer function is $256^2 + 256$. The tradeoff, however, is in expressivity. A transfer function can no longer be specified based on the unique combination of all three data values. Separable transfer functions are still quite powerful. Applying the second derivative as a separable 1D portion of the transfer functions is quite effective for visualizing boundaries between materials. With the separable 3D transfer function for scalar volumes, there is only one boundary emphasis slider which affects all classification widgets as opposed to the general case where each classification widget has its own boundary emphasis slider. A similar approach is employed for multi-variate data visualization. The meteorological example used a separable 3D transfer function. Temperature and humidity were classified using a 2D transfer function and the multi-derivative of these values was classified using a 1D transfer function. Since our specific goal was to show only regions with high values of $\|G\|$, we only needed two sliders to specify the beginning and ending points of a linear
ramp along this axis of the transfer function.

6.3 Surface Shading

Shading is a fundamental component of volume rendering because it is a natural and efficient way to express information about the shape of structures in the volume. However, much previous work with texture-memory based volume rendering lacks shading. Many modern graphics hardware platforms support multi-texture and a number of user defined operations for blending these textures per-pixel. These operations, which we will refer to as fragment shading, can be leveraged to compute a surface shading model.

The technique originally proposed by Rezk-Salama et al. [9] is an efficient way to compute the Blinn-Phong shading model on a per-pixel basis for volumes. This approach, however, can suffer from artifacts caused by denormalization during interpolation. While future generations of graphics hardware should support the square root operation needed to renormalize on a per-pixel basis, we can utilize cube map dependent texture reads to evaluate the shading model. This type of dependent texture read allows an RGB color component to be treated as a vector and used as the texture coordinates for a cube map. Conceptually, a cube map can be thought of as a collection of six textures that make up the faces of a cube centered about the origin. Texels are accessed with a 3D texture coordinate \((s, t, r)\) representing a direction vector. The accessed texel is the point corresponding to the intersection of a line through the origin in the direction of \((s, t, r)\) and a cube face. The color values at this position represent incoming diffuse radiance if the vector \((s, t, r)\) is a surface normal or specular radiance if \((s, t, r)\) is a reflection vector. The advantages of using a cube map dependent texture read is that the vector \((s, t, r)\) does not need to be normalized, and the cube map can encode an arbitrary number of lights or a full environment map. This approach, however, comes at the cost of reduced performance. A per-pixel cube map evaluation can be as much as three times slower than evaluating the dot products for a limited number of light sources in the fragment shader stage.

Surface based shading methods are well suited for visualizing the boundaries between materials. However, since the surface normal is approximated by the normalized gradient of a scalar field, these methods are not robust for shading homogeneous regions, where the gradient magnitude is very low or zero and its measurement is sensitive to noise. Gradient based surface shading is also unsuitable for shading volume renderings of multivariate
fields. While we can assign the direction of greatest change for a point in a multivariate field to the eigenvector \(e_1\) corresponding to the largest eigenvalue \(\lambda_1\) of the tensor \(G\) from Equation 4.3, \(e_1\) is only a valid representation of orientation, not the absolute direction. This means that the sign of \(e_1\) can flip in neighboring regions even though their orientations may not differ. Therefore, the vector \(e_1\) does not interpolate, making it a poor choice of surface normal. Furthermore, this orientation may not even correspond to the surface normal of a classified region in a multivariate field.

6.4 Shadows

Shadows provide important visual cues relating to the depth and placement of objects in a scene. Since the computation of shadows does not depend on a surface normal, they provide a robust method for shading homogeneous regions and multivariate volumes. Adding shadows to the volume lighting model means that light gets attenuated through the volume before being reflected back to the eye.

The approach presented in this section differs from previous work using 3D volume light maps [1] in two ways. First, rather than creating a volumetric shadow map, an off screen render buffer is utilized to accumulate the amount of light attenuated from the light's point of view. Second, the slice axis is modified to be the direction halfway between the view and light directions. This allows the same slice to be rendered from both the eye and light points of view. This approach is similar to the 2D shadow buffer described in [36]. However, the algorithm described below is general with respect to view and light orientation.

Consider the situation for computing shadows when the view and light directions are the same, as seen in Figure 6.1(a). Since the slices for both the eye and light have a one-to-one correspondence, it is not necessary to precompute a volumetric shadow map. The amount of light arriving at a particular slice is equal to one minus the accumulated opacity of the slices rendered before it. Naturally if the projection matrices for the eye and light differ, we need to maintain a separate buffer for the attenuation from the light's point of view. When the eye and light directions differ, the volume would be sliced along each direction independently. The worst case scenario happens when the view and light directions are perpendicular, as seen in Figure 6.1(b). In the case, it would seem necessary to save a full volumetric shadow map which can be re-sliced with the data volume from the eye's point of view providing shadows. This approach, however, suffers from an artifact
referred to as attenuation leakage. The visual consequences of this are blurry shadows and surfaces which appear much darker than they should due to the image space high frequencies introduced by the transfer function. The attenuation at a given sample point is blurred when light intensity is stored at a coarse resolution and interpolated during the observer rendering phase.

Rather than slice along the vector defined by the view direction or the light direction, the slice axis is modified to allow the same slice to be rendered from both points of view. When the dot product of the light and view directions is positive, the slice axis is the vector halfway between the light and view directions, seen in Figure 6.1(c). In this case, the volume is rendered in front to back order with respect to the observer. When the dot product is negative, the slice axis is the vector halfway between the light and the inverted view directions, seen in Figure 6.1(d). In this case, the volume is rendered in back to front order with respect to the observer. In both cases the volume is rendered in front to back order with respect to the light. Care must be taken to insure that the slice spacing along the view and light directions are maintained when the light or eye positions change. If the desired slice spacing along the view direction is $d_v$ and the angle between $v$ and $l$ is $\theta$ then the slice spacing along the slice direction is

$$d_s = \cos \left( \frac{\theta}{2} \right) d_v. \quad (6.1)$$

This is a multi-pass approach. Each slice is first rendered from the observers point of view using the results of the previous pass from the light’s point of view, which modulates the brightness of samples in the current slice. The same slice is then rendered from light’s point of view to calculate the intensity of the light arriving at the next layer.

Since the amount of light attenuated at each slice must be maintained, an off screen render buffer is utilized, known as a pixel buffer. This buffer is initialized to $1 - light\ intensity$. It can also be initialized using an arbitrary image to create effects such as spotlights. The projection matrix for the light’s point of view need not be orthographic; a perspective projection matrix can be used for point light sources. However, the entire volume must fit in the light’s view frustum. Light is attenuated by simply accumulating the opacity for each sample using the over operator. The results are then copied to a texture which is multiplied with the next slice from the eye’s point of view before it is blended into the frame buffer. While this copy to texture operation has been highly optimized on the current generation of graphics hardware, we have achieved a dramatic increase in performance using a hardware extension known as render to texture. This
extension allows us to directly bind a pixel buffer as a texture, avoiding the unnecessary copy operation.

This approach has a number of advantages over previous volume shadow methods. First, attenuation leakage is no longer a concern because the computation of the light transport (slicing density) is decoupled from the resolution of the data volume. Computing light attenuation in image space allows us to match the sampling frequency of the light transport with that of the final volume rendering. Second, this approach makes far more efficient use of memory resources than those which require a volumetric shadow map. Only a single additional 2D buffer is required as opposed to a potentially large 3D volume. One disadvantage of this approach is that due to the image space sampling, artifacts may appear at shadow boundaries when the opacity makes a sharp jump from low to high. This can be overcome by using a higher resolution for the light buffer than for the frame buffer. We have found that 30 to 50 percent additional resolution is adequate.

As noted at the end of the previous section, surface based shading models are inappropriate for homogeneous regions in a volume. However, it is often useful to have both surface shaded and shadowed renderings regardless of whether or not homogeneous regions are being visualized. To insure that homogeneous regions are not surface shaded, we simply interpolate between surface shaded and unshaded using the gradient magnitude.
Naturally, regardless of whether or not a particular sample is surface shaded, it is still modulated by the light attenuation providing shadows. In practice we have found that interpolating based on $1 - (1 - \|\nabla f\|)^2$ produces better results, since mid-range gradient magnitudes can still be interpreted as surface features. Figure 6.2 shows a rendering which combines surface shading and shadows in such a way. Figure 4.1 shows a volume rendering using shadows with the light buffer initialized to simulate a spotlight. Figures 4.2 and 4.3 show volume renderings using only surface based shading. Figures 4.5, 4.6, and 4.7 only use shadows for illumination.
Figure 6.2. Volume renderings of the Visible Male CT (frozen) demonstrating combined surface shading and shadows.
CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Parallel Volume Rendering

Hardware volume rendering is a highly effective interactive visualization modality. Unfortunately, it imposes limits on the size of volumetric datasets which can be rendered with adequate update rates. This thesis presents a scalable solution for the near interactive visualization of dataset time steps which are potentially an order of magnitude larger than the capabilities of a modern graphics card alone. The implementation is also flexible enough to support advanced interaction tools and serve as a platform for future volume rendering and visualization research. The results of this research have been integrated in a production quality application for use by scientists at Los Alamos National Laboratory.

With the recent performance gains in the commodity graphics card market, PC clusters are an attractive replacement for the Origin 2000 system presented in this thesis. While TRex has been successfully ported to a PC cluster, this task still presents several significant challenges. The limitations present in both the internal PC architecture and interconnection networks make it difficult to maintain near interactive rendering rates. The I/O systems on PCs are not concurrent and synchronizing frames add additional latency to the system. There are several areas in which the current algorithms need to be modified so they operate efficiently in a distributed memory environment, such as the compositing phase and compression for optimal data transfer rates.

TRex’s VR capabilities can be enhanced with intuitive interaction devices and new widgets. Future research will explore optimizations to TRex’s pipeline, such as predictive tracking, that take advantage of the known inter-frame latency. Direct manipulation widgets have proven to be an indispensable tool for interactive visualization and immersive environments. New widgets for performing various operations on volumetric data such as classification, segmentation, annotation, editing, and vector volume visualization should
be explored. Other considerations are the addition of visualization modalities such as haptic feedback and auralization.

7.2 Multi-Dimensional Transfer Functions

This thesis also demonstrates the importance of multi-dimensional transfer functions for direct volume rendering applications. Several examples for both scalar and more general multivariate datasets are presented. The importance of interactive techniques is also identified. New interaction modalities and tools make the process of specifying a high quality transfer function efficient and effective. These tools guide the user based on dataset specific information.

Direct manipulation widgets and spatial interaction techniques lend themselves well to immersive environments. Recent efforts have focused on experimenting with multi-dimensional transfer functions and dual-domain interaction in a stereo, tracked, environment. An immersive environment could make interacting with a 3D transfer function more natural and intuitive. Specifying a multi-dimensional transfer function, however, requires skill, experience, and patience even with well designed interaction modalities and tools. One solution for making virtual reality volume rendering accessible to an inexperienced user is to provide a transfer function in a simplified form. Once regions have been classified, the most important parameters for investigation are opacity and color, which can be set with a virtual knob or slider. Figure 7.1 shows an example of such an interface. The menu in the upper-right is the simplified transfer function interface. Each knob modifies the opacity of one or more classification widgets in a multi-dimensional transfer function. Rather than manipulating the transfer function directly, these knobs offer the user a more meaningful representation of a potentially complicated and unintuitive transfer function domain.

The use of direct manipulation widgets for classifying discrete regions of the transfer function domain has proven useful for efficiently manipulating multi-dimensional transfer functions. The transfer function interface could be further improved by permitting a user to group widgets and specify portions of the transfer function as layers. Layers permit an additional level of organization. Each classified feature could be specified using any number of classification widgets, but still be identified and manipulated as a single entity using its layer.

Visualizing multivariate datasets using multi-dimensional transfer functions has the
potential to greatly improve the expressivity of volume rendering applications. One major problem that arises with multivariate datasets is that they may have been acquired at different times with different orientations and fields of view. Registration of these datasets is important. A subtle mis-registration may dramatically affect the quality and value of a visualization. An interactive volume rendering application, such as the one proposed in this thesis, could be improved with the capability of performing automatic and semi-automatic registration of multiple data channels as an integrated part of the volume rendering and exploration process.

7.3 Shading

This thesis presents a number of methods for volume shading, including a novel technique for generating volumetric shadows. Using shadows for illumination is advantageous since it does not require a surface normal, thus eliminating the need for a normal volume. New shading models based on this approach might have the potential to create even more realistic and informative imagery. As future generations of graphics hardware provide even richer feature sets, it should soon be possible to create and implement better approximations of light transport through volumetric media at interactive frame rates.

Figure 7.1. A virtual reality interface for manipulating multi-dimensional transfer functions.
Another area of future research would be to explore methods of surface normal generation using on-the-fly post-classification gradient estimation. This is a non-trivial problem since the transfer function can introduce very high frequencies or discontinuities, which can be problematic for creating normals that produce smooth shading. Such a method would have the potential to provide robust normals for surface shading multivariate volume visualizations.

7.4 Limitations

While direct volume rendering is an effective technique for visualizing and exploring spatially varying data, there are a number of disadvantages to this technique. Extracting geometry from volumetric data can be inefficient for generating imagery, however it is vital for a number of other tasks. For instance, once an isosurface has been extracted from a volume, extraneous features can be culled from the visualization using a connected components analysis. Numerical simulations also require an explicit geometric representation of features, which direct volume rendering techniques simply do not provide. The use of a transfer function in direct volume rendering can also be problematic when a sense of absolute precision is required. It is commonly used to specify fuzzy regions which are likely to be a feature of interest. When this feature is identified by a very narrow range of data values, it may be difficult to accurately render the feature. A sharp, high frequency, peak in the transfer function can introduce discontinuities in the rendering, which can only be avoided by increasing the number of slices used to render the volume. Most current hardware implementations only support limited precision color and opacity representations that place an upper limit on the number of slices that can be composited together without causing significant artifacts.

7.5 Summary

Interactive volume rendering is an effective method for visualizing and exploring volume data. The process of volume exploration can be improved by addressing three key factors: rendering speed, maximum dataset size, and the quality of interaction and imagery. This work contributes to the field of volume rendering in several ways. First, a hybrid parallel volume rendering platform is presented, which is suitable for visualizing extremely large datasets and virtual reality volume rendering. The importance of multi-dimensional transfer functions for direct volume rendering applications is
demonstrated and applied to both scalar and more general multivariate datasets. A suite of new interaction modalities and tools enables effective volume visualization through a natural process of exploration, specification, and refinement. Dual-domain interaction is introduced as a new mechanism for specifying a multi-dimensional transfer function. Techniques for volumetric surface shading and shadows are explored. Finally, it is shown that modern graphics hardware can used to generate high-quality volume visualizations of large datasets at interactive frame rates.
REFERENCES


[14] Ebert, D., Morris, C., Rheingans, P., and Yoo, T. Designing Effective


(http://www.cs.utah.edu/~gk/MS).

tion (1998), pp. 79–86.


[41] Lorensen, W., and Cline, H. Marching cubes: A high resolution 3D surface


