REMOTE USER-DRIVEN EXPLORATION OF LARGE SCALE VOLUME DATA

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of The Ohio State University

By

Naeem O. Shareef, B.S., M.S.

*****

The Ohio State University
2005

ABSTRACT

No rendering pipeline exists to explore very large volume data, for example on the order of terabytes or more. In the extreme case, the data is essentially “stuck” at the site of creation. The consequence is that the utility of the dataset is greatly diminished since only a very small number of users are able to explore the rich information contained within. We address the challenge of how to provide access to such datasets to remote user’s equipped with low-cost computational and display technology. Our work proposes a novel end-to-end rendering pipeline that allows for effective data exploration. Our paradigm couples view-dependent and image-based data structures along with novel rendering algorithms that allow for fast spatial and transfer function browsing on the client side.

The view-dependent data structure, called a Pixel Ray Image (PRI), holds scalar information on projection rays through the volume. The representations for the scalar data are determined from the requirements of the particular projection equation. We present compact representations that may be stored in texture form for interactive rendering on todays PC graphics hardware. During spatial browsing, when the transfer functions do not change, the Layered Slab Image (LSI) data structure holds pre-computed projections of the data that are then used to quickly compute approximate renderings at nearby viewpoints.

The PRI represents a compression of the volume along a single dimension. Sampling planes placed orthogonal to the sampling direction of the PRI can be computed easily in graphics hardware. Using an approach similar to the shear-warp algorithm, the volume may be rendered directly from this compressed format at views within a 45 degree neighborhood. The volume may be rendered from any view by placing three PRI at orthogonal sampling directions to each other. We present a novel rendering algorithm to render the volume directly from the compressed format on PC graphics hardware.

Dedicated to my parents, Ghouse and Sameena,
and to my brothers, Saleem and Matheen.
Their support was invaluable towards the completion of this work.

ACKNOWLEDGMENTS

In my overextended stay at The Ohio State University, there are many colleagues and friends to thank. Please forgive me if I neglect to mention anyone. I wish to thank all of the reviewing committee members, Roger Crawfis, Richard Parent, and Han-Wei Shen, who offered their time and wisdom. Dr. Roni Yagel, technically my second advisor but my first advisor in computer graphics, was the main impetus behind my persuit into the field. I would not have been able to enter the dissertation program without his backing. My collaboration with Professor DeLiang Wang helped to shape my ability to perform research. I was able to win a paper award and publish a journal paper in the area of Artifical Intelligence under his tutelage. Professor Roger Crawfis became my second advisor in computer graphics and introduced me to the problems that are addressed in this dissertation. Professor Han-Wei Shen was a great motivator, provided invaluable advice, and helped me through some tough times. Thanks go to the department administrative staff. Most notably are Kathryn ("Kitty") Reeves, who helped me to find teaching positions so that this dissertation could be completed, Tom Fletcher, and Marty Marlatt. Thanks also go to Dr. Brian Shelbourne at Wittenberg University, who allowed me to pay my living expenses as an adjunct professor and the opportunity to teach his class, COMP 370 Computer Graphics, during the Spring semester.

The clique of former students who shared our mentor, Roni Yagel, provided lasting support and an ongoing friendship. Our fraternity will hopefully never disappear: Yair Kurzion, Asish Law, Raghu Machiraju, Torsten Moeller, Klaus Mueller, David Reed, Po-Wen Shih, and Edward J Swan II. Everyone in this group has achieved success in their careers since their graduations and it is now my turn being the last of the group,.

I would also like to thank current and former students at OSU with whom I have had the pleasure to create a great friendship: Tahmina Ansari, Praveen Bhaniramka, Udeepta Bordoloi, Jason Bryan, Shannon Campbell, Matthew Camuto, Erdogan Cesmeli, Sean Chang, Eugene Fratkin, Jinzhu Gao, Antonio Garcia, Joachim Giesen, Samrat Goswami, Brent Haley, Christa Hayes, Jian Huang, Ming Jiang, Yootai Kim, Scott King, Domin Lee, Jinho Lee, Naveen Leekha, Guo-Shi Li, Liya Li, Sameep Mehta, Ethan Metsger, Luke Molnar, Kovalan Muniandy, Johan Oestman, Tathagata Ray, John Rusnak, Wendy Rusnak, Nivedita Sahasrabudhe, Dennis Sessanna, Richard P. Sharp, Arunachalam Somasundaram, Madhu Soundararajan, Don Stredney, Jian Sun, Shivaraj Tenginakai, Jenafer Thomas, Chaoli Wang, Dr. Gregory Wiet, Jonathan Woodring, Lining Yang, and Amar Zumkhawala.

I would also like to thank those outside the OSU sphere of influence: Dirk Bartz, Lance Burton, Carrie Casillas, Perry Chappano, Joseph Heafitz, and Stephanie Kitchen.

VITA

July 16, 1968	Born - New Rochelle, NY
1990	B.S. Applied Mathematics/Compuer Science, Carnegie Mellon University, Pittsburgh, PA
1992	M.S. Computer And Information Science, The Ohio State University, Columbus, OH
1991 - 1993	Graduate Teaching Assistant, The Ohio State University
1993 - 1994	Graduate Research Assistant, The Ohio State University
1994	Graduate Teaching Assistant, The Ohio State University
1994 - 2000	Graduate Research Assistant, The Ohio State University
2001 - 2004	Graduate Teaching Assistant, The Ohio State University
2004 - present	Adjunct Professor, Wittenberg University, Springfield, OH

PUBLICATIONS

Research Publication

1. N. Shareef and R. Crawfis, “A View-dependent Approach to MIP for Very Large Data,” *Proc. SPIE EI ’02*, Vol. 4665, pp. 13-21, 2002.

2. J. Huang, K. Mueller, N. Shareef, and R. Crawfis, “FastSplats: Optimized Splatting on Rectilinear Grids,” *IEEE Visualization ’00*, Salt Lake City, Utah.

3. J. Huang, N. Shareef, R. Crawfis, P. Sadayappan, and Klaus Mueller, “A Parallel Splatting Algorithm with Occlusion Culling,” *3rd Eurographics Workshop on Parallel Graphics and Visualization*, 2000.

4. N. Shareef, “An Image-Based Approach to Scientific Visualization Using Layers,” *First Annual Biomedical Engineering Research Symposium*, The Ohio State University, OH, 2000.

5. D. Stredney, A. Agarwal, D. Barber, R. Crawfis, WC Feng, J. Hou, DK Panda, P. Sadayappan, K. Powell, P. Schmalbrock, D. Sessanna, GJ Wiet, N. Shareef, J. Bryan, “Interactive Medical Data on Demand: A High-Performance Image-based Approach Across Heterogeneous Environments,” *Proc. of Medicine Meets Virtual Reality 8*, Westwood et. al., (Eds), IOS Press Amsterdam, pp. 327-333, 2000.

6. K. Mueller, J. Huang, N. Shareef, and R. Crawfis, “High-Quality Splatting on Rectilinear Grids With Efficient Culling of Occluded Voxels,” *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, No. 2, pp. 116-143, 1999.

7. J. Huang, K. Mueller, N. Shareef, and R. Crawfis, “VOXBLT : An Efficient and High Quality Splat Primitive,” *IEEE Visualization ’99: Late Breaking Hot Topics*, 1999.

8. K. Mueller, N. Shareef, J. Huang, and R. Crawfis, “IBR-Assisted Volume Rendering,” *IEEE Visualization ’99: Late Breaking Hot Topics*, 1999.

9. D. Stredney, R. Crawfis, GJ Wiet, D.Sessanna, N. Shareef, and J. Bryan, “Interactive Volume Visualization for Synchronous and Asynchronous Remote Collaboration,” Proc. of Medicine Meets Virtual Reality 7, Westwood et. al., (Eds), IOS Press Amsterdam, pp. 344-350, 1999.

10. J. E. Swan II, K. Mueller, T. Moller, N. Shareef, R. Crawfis, R. Yagel, “Perspective Splatting,” *Naval Research Laboratory Memorandum Report, NRL/MR/5580--99-8355*, 1999.

11. R. A. Crawfis, Po-Wen Shih, N. Shareef, “Visualizing Dual 3D Vector Fields Using a Splatting Technique,” *IEEE Visualization ’98 (Vector Field Visualization Workshop),* Chapel Hill, NC, 1998.

12. K. Mueller, T. Moller, J. E. Swan II, R. Crawfis, N. Shareef, R. Yagel, “Splatting Errors and Aliasing,” *IEEE Transactions on Visualization and Computer Graphics ITVCG,* Vol. 4, No. 2, IEEE Computer Society: April-June, pp. 178-191, 1998.

13. C. Edmond, D. Heskamp, G. Mesaros, S. Weghorst, C. Airola, D. Stredney, D. Sessanna, R. Crawfis, N. Shareef, G. Wiet, and J. Miller, “ENT SURGICAL SIMULATOR,” (Final Report - Cooperative Agreement No. DAMD17-95-2-5023 - HITL Technical Report R-99-14). Fort Detrick, MD 21702-5012: U.S. Army Medical Research and Materiel Command.

14. N. Shareef, D. L. Wang, R. Yagel, “Segmentation of Medical Images Using LEGION,” *IEEE Transactions on Medical Imaging*, Vol. 18, No. 1, January 1999.

15. D. Stredney, G.J. Wiet, R. Yagel, D. Sessanna, Y. Kurzion, M. Fontana, N. Shareef, M. Levin, K. Martin, A. Okamura, “A Comparative Analysis of Integrating Visual Representations with Haptic Displays,” *Proc. of Medicine Meets Virtual Reality 6, Art, Science, Technology: Healthcare (R)evolution*, San Diego, CA, pp. 20-26, 1998.

16. N. Shareef, D. L. Wang, R. Yagel, “Segmentation of Medical Images Using LEGION,” *OSU-CISRC-4/97-TR26*, 1997.

17. J. E. Swan II, K. Mueller, T. Mo"ller, N. Shareef, R. A. Crawfis, R. Yagel, “An Anti-Aliasing Technique for Splatting,” IEEE Visualization ’97, pp. 197-204, 1997.

18. N. Shareef, “Segmentation of MRI and CT Data Using Locally Excitatory Globally Inhibitory Oscillator Networks,” *Midwest Artificial Intelligence and Cognitive Science Conference*, 1996.

19. N. Shareef, D. Wang, and R. Yagel, “Segmentation of Medical Data Using Locally Excitatory Globally Inhibitory Oscillator Networks,” *World Congress on Neural Networks WCNN ’96*, 1996.

20. R. Yagel, D. M. Reed, A. Law, P. Shih, and N. Shareef, “Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing,” *Proc. 1996 Symposium on Volume Visualization*, 1996.

21. R. Yagel, D. Stredney, G.J. Wiet, P. Schmalbrock, D.J. Sessanna, Y. Kurzion, E. Swan, N. Shareef, J. Smith, and D.E. Schuller, “Cranial Base Tumor Visualization through Multimodal Imaging Integration and Interactive Display,” *4th Annual Meeting of the Society of Magnetic Resonance*, 1996.

22. G.J. Wiet, D. Stredney, R. Yagel, E. Swan, N. Shareef, P. Schmalbrock, K. Right, J. Smith, and D.E. Schuller, “Cranial Base Tumor Visualization through High Performance Computing,” *Proc. of Medicine Meets Virtual Reality 4, Health Care in the Information Age*, pp. 43-59, 1996.

23. N. Shareef and R. Yagel, “Rapid Previewing via Volume-based Solid Modeling,” *The 3rd Symposium on Solid Modeling and Applications, SOLID MODELING ‘95*, pp. 281-292, 1995.

24. N. Shareef and R. Yagel, “Rapid Previewing via Volume-based Solid Modeling,” *OSU-CISRC-11/94-TR58*, 1994.

FIELDS OF STUDY

Major Field: Computer Science and Engineering

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A pervasive problem in scientific visualization is that compute power has always been outpaced by an ever increasing data size. This is especially true for volume visualization of scalar data, the focus of our work. At a minimum, large scale volume datasets hold billions of samples. New technology is already creating datasets on the order of tera- and petabytes. State-of-the-art massively parallel supercomputers are now able to reduce the time to create gigantic datasets of simulations from weeks to only days and hours. Also, technological advances in sampling devices are facilitating high data resolutions, and therefore, unprecedented detail. Many such datasets are already or will be available in the near future. The Visible Human Project [1] provided the first high resolution scans of the entire human body. The Ohio State University houses the strongest MRI magnet for medical applications in the world at 8-Tesla, and will be able to image the human body with slice resolutions exceeding 1K x 1K pixels. The challenges that need to be addressed to extract information from data of this magnitude is outlined by the DOE Accelerated Strategic Computing Initiative (ASCI) [1], which has identified visualization as a key tool to understand these datasets. Important datasets already exist but the infrastructure and tools are not available to effectively extract the wealth of information contained within. For data

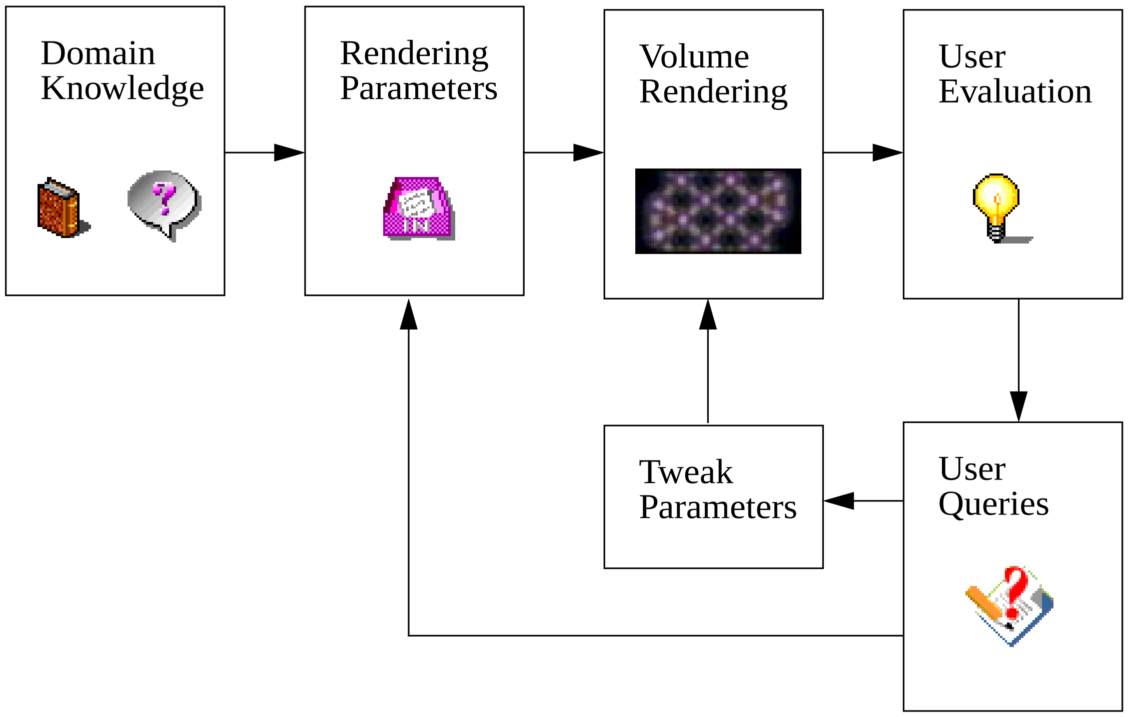


Figure 1. A user-driven data exploration framework for volume rendering. On-the-fly rendering is embedded into a feedback loop where the user successively changes visualization parameters and analyzes successive renderings.

magnitudes at terabytes or more, an “end-to-end” rendering pipeline is nonexistent due to insufficient technological infrastructure, including high speed data retrieval and management, networks, and computational power.

The utility of any dataset is truly maximized when more users are able to study it. Many of these users are located at remote sites, which are equipped with only modest computing technology. Large scale datasets reside at well-funded facilities (usually government funding) with the technology and infrastructure to store, manage, and locally move the data. How to provide remote access to these datasets is a challenge that has not been adequately addressed by the research community. There are bottlenecks all along the data pipeline between the location where the data is stored and the remote site. Network bandwidth limitations between the sites are a severe bottleneck. The time to download a volume dataset on the order of gigabytes to a remote disk can take tens of minutes or more. Local disk storage will be able to hold only a handful of datasets and lack adequate data access speeds. Only parallel computing platforms will be able to provide the computational power needed to render the data. For gigantic datasets, the remote user will have to try to visit the research facility since the data is essentially “stuck” at the site.

The purpose of volume visualization is not to simply compute a “pretty picture.” Rather, it is a tool that could be used by researchers and scientists to help in understanding the data and to glean new insight. Our work pursues a user-driven approach to data discovery, which is both an effective means to data exploration and has already been proven to be valuable in practice. This approach is touted by Will Schroeder in the “Transfer Function Bake-Off” [73] for transfer function design, where the transfer function is an important visualization parameter. A key underpinning of this approach is that human cognition is tightly coupled with the human visual system, and so we are extremely good at detecting and interpreting patterns and features. In fact, the participation of the end user is invaluable in any data understanding framework. We illustrate the essential steps of a user-driven framework in Figure 1. It has two feedback loops where visualization is a key component. The user comes to the task equipped with a set of personalized goals and initial queries about the data. In many cases, the data was created by the user to study a particular problem. First, an initial rendering is computed from chosen visualization parameters, e.g viewpoint, transfer function, and shading, which may be set using a priori domain knowledge or even with automated or semi-automated methods. The rendering provides a single view and interpretation of the data which the user can then analyze. The information interpreted from the rendering may not be sufficient to satisfy all the user's queries and may even spawn new ones. The user can then enter the inner loop where the visualization parameters are changed incrementally from the initial parameters to compute successive renderings for further analysis. Alternatively, the user may enter the outer loop where a new set of visualization parameters are chosen to explore a different set of queries. In this manner, one is able to construct a mental model of the data, which is strengthened by a "hands-on" approach.

Three essential components of a manual approach to volume exploration are interactive feedback in terms of high frame rates, acceptable image quality, and the ability to express parameter changes through the GUI interface. Recent work has focused on achieving high frame rates by leveraging hardware acceleration. Kniss [41] uses hardware acceleration in the form of volume rendering with 3D texture mapping in order to quickly compute renderings using multi-dimensional transfer functions for classification. The user uses a suite of widgets via the GUI to express the functions. Similarily, König [44] uses the commercial product called VolumePro©, which is a volume renderer implemented in specialized hardware. These approaches can only render from rectilinear volumes and have been shown to be useful for medical datasets, which contain surface boundaries between material. A main limitation of these hardware platforms is that multiple frame rates are only possible for small to medium size volume datasets. How to render large scale volume data is an open area of research.

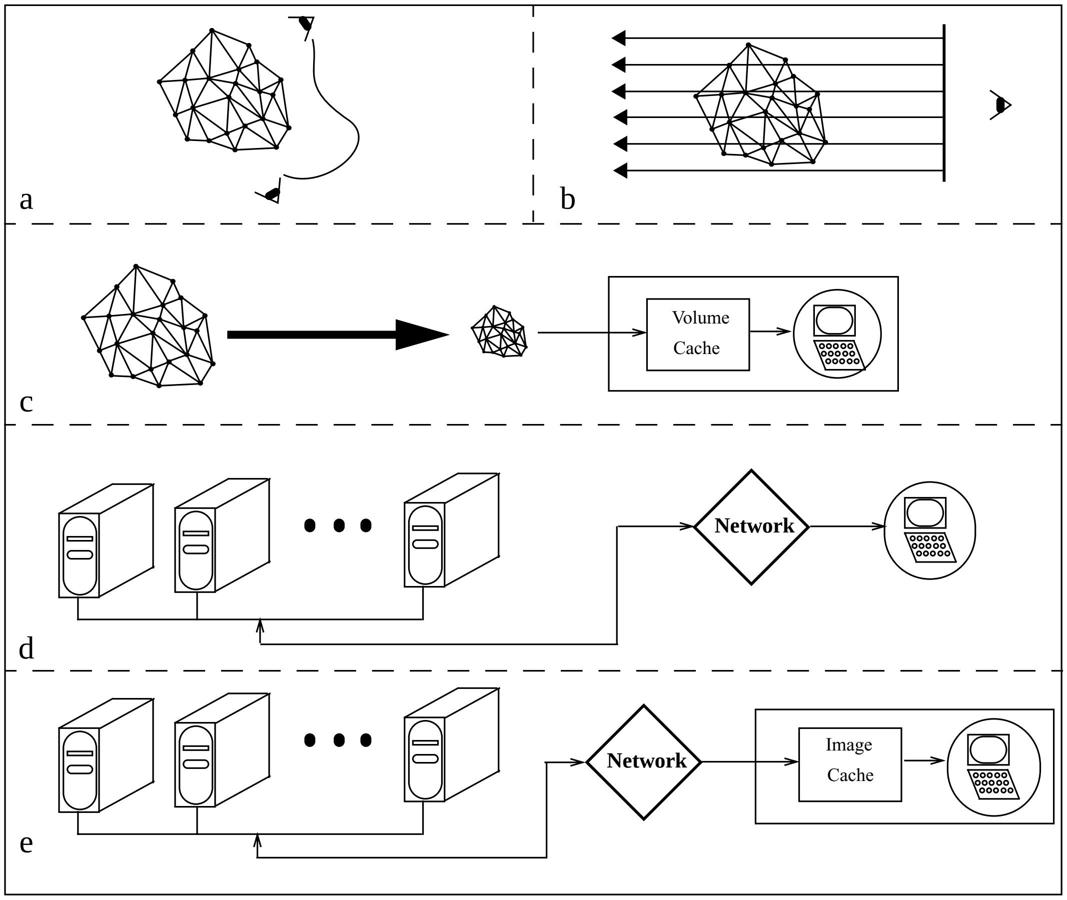


Figure 2. Five approaches to render large scale volume data in a remote visualization scenario. The remote user can view a pre-computed movie (a). A view-dependent construction holding scalar information, shown in (b), allows the user to change the transfer function on-the-fly. The approach shown in (c) reduces the size of the volume so that it can be rendered on the user's desktop. A state-of-the-art renderer can compute renderings at the host facility (d) and then send the final image to the user for display. An image cache is used in (e) to store retrieved images from a state-of-the-art renderer, where images are re-used to compute approximate renderings at novel views.

A more challenging problem we address in this work is how a remote user can explore large volume data in a user-driven paradigm. Providing access to the data at remote sites adds another bottleneck with its own unique challenges. This includes increased limitations in network bandwidth, memory and storage space, and compute power of the low-end client on the remote side. We illustrate in Figure 2 five possible approaches that could be used in a remote user scenario. The first approach, shown in Figure 2a, is to pre-compute a movie showing a fly-through animation of the data. The movie is easily downloaded or streamed to the user's desktop and played in realtime. The remote user does not need to access the cumbersome large volume dataset at all. Instead, expensive rendering using a high-end computing platform is done as a separate pre-process decoupled from display. The problem is that the user is a passive participant and cannot change any of the visualization parameters, except to forward and reverse the movie. A view-dependent approach is shown in Figure 2b, where scalar information is pre-computed and stored at a chosen view. The user's view is kept fixed while other visualization parameters, such as classification and shading, can be changed on-the-fly to compute new renderings. As with the first approach, access to the original volume dataset is not needed during display after the view is computed. A compression scheme may be needed to reduce the size of the representation. Many such view samples may be pre-computed, though the user is restricted to only these camera locations.

The next three approaches render directly from the volume and thus allow changes to any visualization parameter. If the volume is rendered on the user's desktop, it must be reduced in size, as shown in Figure 2c. An inevitable consequence is that much of the data must be thrown away and/or smoothed out in order to fit into memory or local disk, and thus image quality noticeably suffers. To ameliorate this, the volume may have to be rendered out-of-core, which adds an additional computational bottleneck. Hardware accelerated rendering methods on the PC may be utilized to render these reduced volumes at high frame rates for rectilinear volumes [11][72] that fit in video memory or moderate sized unstructured grid volumes [77][26]. Alternatively, the volume may be rendered at the host facility equipped with state-of-the art computing resources, for example supercomputer technology or clusters of workstations, as shown in Figures 2d and 2e. The approach illustrated in Figure 2d shows a scenario where the back-end acts as a rendering server and the remote user's machine is simply a display client to show the resultant rendered image. When the user changes any visualization parameter, a round-trip is required where the new parameter is sent to the back-end and the client waits for a new rendering to be sent back for display. The latency to render an image and transmit it over the network may be substantial. To overcome this, the approach shown in Figure 2e, augments the client-server framework of Figure 2d by caching images and then re-using them using an image-based rendering method. While the back-end is busy computing the next rendering, the client is able to display an approximate rendering at interactive frame rates by re-projecting the cached imagery. Bethel [6] implements this framework by caching layers of images at a particular view and then uses 2D texture hardware blend the layers to display to the user. The image-based construction was first presented by Brady [9] and extended by Mueller [67]. These implementations show that image quality is not appreciably degraded in the approximate rendering for a limited range of novel views.

We identify four key criteria that should be satisfied in a working remote visualization framework for rendering large volume data that supports effective user-driven data exploration :

1. Quick visual feedback when changing any visualization parameter
2. Reduce or eliminate the dependence on expensive infrastructure
3. Display at high screen resolution on the remote user's desktop
4. Preserve image quality

All of the approaches presented in Figure 2 fall short on one or more of these criteria. A pre-computed movie does not allow the user to change any of the visualization parameters. A pre-computed view-dependent scalar construction allows the user to easily change parameters, such as the transfer function and lighting, but the user's view remains fixed. Methods to reduce the volume size usually result in poor image quality. A state-of-the-art renderer at a host facility would only be accessible to a limited number of remote users. To overcome the severe network bottleneck between the host facility and the remote user, it is beneficial to try to move more of the rendering tasks to the client side. This is becoming a realizable goal with advances in desktop technology in recent years including : faster CPUs by an order of magnitude, larger and cheaper L1 and L2 cache, main memory, and disk capacity, and PC graphics cards that have surpassed the speed of their graphics workstations predecessors. High resolution displays, e.g. plasma displays and even caves, are becoming affordable for individual user's and small research groups.

We present a novel end-to-end graphics pipeline where most of the rendering tasks are done on the client side by combining the methods introduced in Figures 2b and 2e. We present the Pixel Ray Image (PRI), a view-dependent construction holding per-pixel scalar information. The representations are compact and in compressed form to efficiently solve the different volume projection equations. These view samples are cached on the client side. The view-dependent data is organized into 2D texture maps, which are used to quickly render the data by leveraging PC graphics hardware.

CHAPTER 2

VOLUME RENDERING

The problem of how to visualize a field of values in a volume of space is called volume rendering. Each field value may represent a variety of phenomena, such as temperature, pressure, Hounsfield unit, direction vector, and tensor. We focus on the problem of how to render a field of scalar values defined in 3D space. More formally, the field is defined as a function $f(x, y, z)$ that maps $\Re^3 \rightarrow \Re$. The field may be defined analytically for all locations in the volume or in a discrete manner as a collection of data sample values. In the former case, the volume is usually the result of a simulation, for example computational fluid dynamics. In the latter case, the volume is sampled on a grid either with a finite element method or acquired with a sampling device. The grid samples are organized so that they are vertices to volume cells, which tessellate the volume. The two types of structures are *structured* and *unstructured* grids. The structured grid consists of hexagonal cells that are defined implicitly and includes uniform, rectilinear, and curvilinear grids. Medical data is defined on structured grids obtained from sources such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI), and Positron Emission Tomography (PET). Unstructured grids are usually constructed from tetrahedral cells, but can also include a mixture of cell types. Thus, neighborhood information is explicitly defined in the *unstructured grid*.

There are three popular approaches to render a volume: isosurfaces, maximum projection, and volume integration. In isosurface rendering, an embedded surface is defined by a particular scalar value, called an isovalue. Indirect approaches [55][13] construct a polygonal mesh from the volume and render the mesh using traditional polygon rendering. A popular algorithm is called Marching Cubes [55], which operates on rectilinear grids. Parker [70] uses a raycasting approach to directly render the isosurface. The maximum projection approach displays the largest classified scalar value along a viewing ray. This is a "winner-take-all" scheme and requires a search along the viewing ray. Volume integration (also called volume summation) includes all scalar values in the volume projection to visualize the entire volume. Our work focuses on the the latter two approaches, which are considered to be direct volume rendering methods. The rendering pipeline consists of four key steps: resampling, reconstruction, classification, and projection. Direct volume rendering algorithms can be classified into the following methods: raycasting [48][10], splatting [99][12], shear-warp [45][18], cell projection [100], and slice-based methods [30][91][103]. Recent research has focused on implementations for the PC platform due to the fast advances in graphics hardware technology. Earlier methods used standard polygon-based and texture mapping graphics acceleration, such as[89][37][77][78], slice-based methods [103], and 3D texture mapping [11][95][98][19]. Specialized graphics hardware provides another rendering platform on the PC, such as Vizard [43] and VolumePro [72].

2.1 Maximum Projection

Maximum projection is a simple and yet very useful volume rendering approach. The maximum classified scalar value along a pixel ray is assigned as the final pixel value. Being a “winner-take-all” decision, the projection equation is usually solved as a search along the pixel ray. The expensive operations of shading and compositing are not performed here. The utility of this approach is its ability to filter the data in a view-dependent manner in order to display values of highest importance. In this capacity, it has primarily been used to visualize particular structures of interest in medical data, such as bone and vasculature, which may be highlighted with a nuclear tracer in CT and MRI data. For example, bone in CT data is usually associated with the high Hounsfield unit values. In addition, surrounding soft tissue is still displayed and serves as a frame of reference, as shown in a rendering of theVisible Female CT dataset in Figure 11. The most common maximum projection method used in practice is Maximum Intensity Projection (MIP), shown in Figure 3a, which linearly maps each scalar value to a grey level intensity. A drawback of the MIP approach is that a MIP image lacks depth information. Both depth-shaded and local maximum intensity MIP approaches try to overcome this deficiency. The Maximum Opacity Projection (MOP) method maps scalars to color and opacity in order to allow for better discrimination between different materials using color without the need to compute the expensive volume integration equations.

2.1.1 MIP

In MIP rendering scalar values are mapped to grey level intensities in a classification

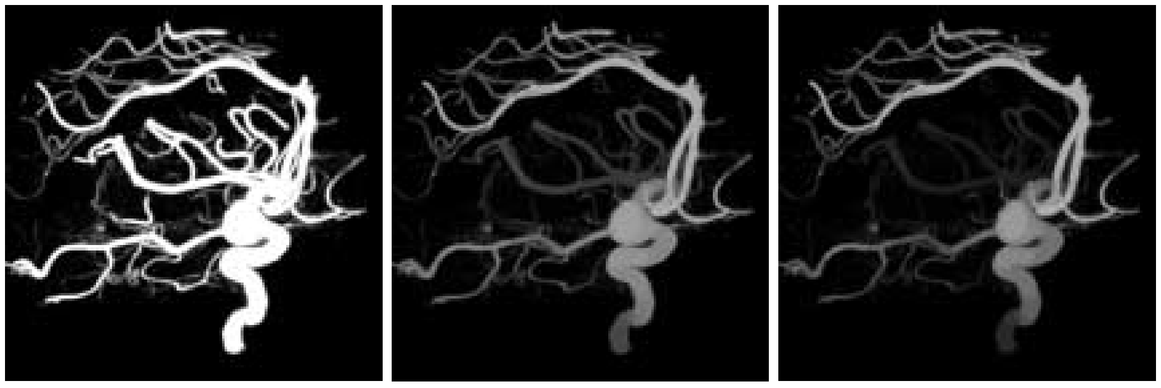


Figure 3. MIP renderings of an Aneurism dataset. (a) is a MIP image. (b) and (c) are depth shaded rendeirngs.

step. In practice, the transfer function that is defined is a linear ramp where higher scalar values map to brighter intensities, as shown in equation 1, where the function maps scalar $s \in [s_{min}, s_{max}]$ to a normalized intensity value.

$$T(s) = (s - s_{min}) / (s_{max} - s_{min}) \tag{1}$$

The maximum operator disregards depth information, which can make it difficult to interpret MIP images. As an example, consider the image in Figure 3a that shows a MIP rendering of an MRA dataset containing a network of vessels from an aneurism. Since structures that have higher intensity values are projected over closer structures with lower intensity, it is difficult to determine the connectedness of the vessel network. Other ambiguities are due to little or no intensity variation between different structures or within the same structure where it may be difficult to discern its shape. How to resolve such ambiguities in MIP images has been an active area of research. We identify five avenues that have been taken to address these issues: 1) animate about the data, 2) re-classification, 3) color-opacity classification, 4) volume clipping, 5) depth shading, and 6) local maximum intensity projection. These approaches require the user to interact with visualization parameters in order to improve interpretation of the maximum projection.

2.1.2 Acceleration Approaches

Animating about the volume can help to resolve some of the ambiguities due to the maximum projection operator. Other views may be able to disambiguate depth ordering between structures that a static MIP image is unable to show. This requires rendering at interactive frame rates and so fast rendering algorithms. An early approach by Heidrich [37] chooses a priori a limited number of isovalues, computes polygonal isosurfaces using Marching Cubes, and renders the maximum value with standard polygon rendering hardware using the *z*-test. Each mesh, which represents a particular isosurface, is mapped to a unique *z*-value. Sakas [80] accelerates raycasting using a fast DDA approach to quickly identify voxels along the sampling ray. Since tri-linear interpolation is expensive, they reconstruct a sample using either nearest neighbor interpolation or a fast approximation to the maximum value of a volume cell. Mroz [66] removes voxels that will not contribute to the MIP as a pre-processing step to rendering. The remaining voxels are stored in a sorted list by value, and then projected to the screen using splatting or shear-warp.

2.1.3 MIP Variants

The general form for the maximum projection operator applied to a pixel ray at pixel

(x, y) is shown in equation 2. The transfer function T is defined as in equation 1 for MIP as it is commonly used in practice. Transfer function mappings other than the linear mapping to grey level can provide for different and useful renderings. The simple scheme by Mroz [66] allows for “windowed” transfer functions so that only voxels with isovalues in a chosen scalar range are displayed. Other mappings for function T can allow for improved contrast-enhancement and picking a subset of material to display from the classification space. The Sliding-Thin-Slab [104] approach clips the volume between the near and far clipping planes and only applies the maximum operator to a slab portion of the volume. The user is allowed to determine the location and size of the slab by repositioning the near and far clip planes. The MOP approach [96] uses a color-opacity transfer function. The maximum operator is applied to the opacity transfer function and the associated color is assigned as the final pixel value.

$$color(x, y) = \max_{0 \le z \le D} (T(f(z))) \tag{2}$$

Depth shading weights the classified sample by its distance from the image plane. This provides depth cues without the need for illumination. The images in Figure 3b and Figure 3c show depth-shading applied to the Aneurism dataset, which should be compared with the MIP rendering shown in Figure 3a. It is clear to see that the depth-shaded renderings show a better disambiguation of the vessel network. For example, portions of the network that look to be connected in the MIP image are in fact disconnected. The depth function may be defined arbitrarily by the user. The image in Figure 3b was computed with a linear ramp depth function. The image in Figure 3c was computed with an exponential depth

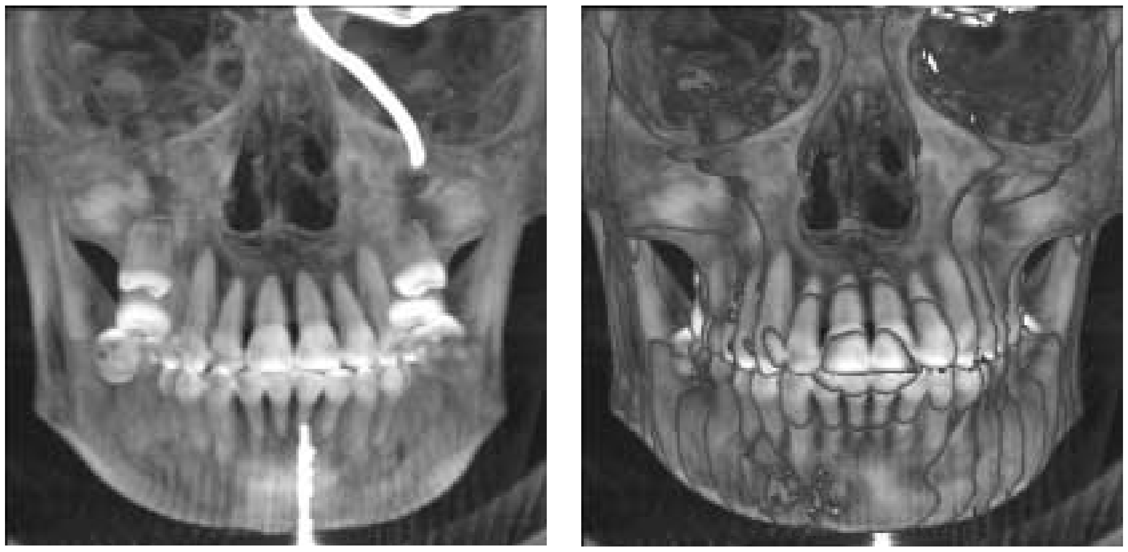

(a) (b)

Figure 4. (a) A MIP image of a CT scan of a human head shows bone structures and tubes highlighted with a tracer. (b) An LMIP image is able to show material closer to the viewer.

function, which emphasizes even more the vessels closer to the viewer. The image was rendered with a hardware-accelerated approach by Shareef [88] using OpenGL’s **glFog** function.

The local maximum intensity projection (LMIP) approach [81] searches for the first intensity value along the pixel ray above a user-defined threshold that is a local maximum. If no such local maximum is found, then the global maximum intensity is assigned as the final pixel value. Figure 4 shows a comparison between MIP rendering (Figure 4a) and the LMIP approach (Figure 4b) on a CT scan of the human head. Closer structures are shown (Figure 4b) over higher intensities towards the back, such as the bright tubular structures shown in Figure 4a.

2.2 Volume Integration

Volume integration is able to incorporate all scalar values on a pixel ray into a final pixel value, as opposed to isosurface or maximum projection rendering. Classification is a mapping, called a transfer function, from scalar properties to optical properties. The most commonly defined mapping is between the scalar value to color and opacity. The scalar values are classified according to the chosen optical model, which describes how light interacts with the material. The choice of an optical model is a key component because it determines the nature of the material in the volume. A cloud model views the volume as a collection of particles in space while a material model assumes a collection of objects each with material properties. An example of the latter case is when scalar values represent Hounsfield units, as in CT imaging, so that materials are easily identified by scalar ranges [22]. The classified samples on the pixel ray are combined into a final pixel value with an alpha compositing operator [74].

2.2.1 Optical Models

A number of optical models [7][39][79][100] have been proposed that describe how light interacts with the scalar field. Sabella's model, in its low-albedo form, is commonly used in practice where the scalar field is assumed to be a cloud of particles that is both self-emitting and light absorbing. Max [61] gives a comprehensive description for three cases of the model: emission only, absorption only, and absorption plus emission. In the emission only model, sometimes called summation rendering, the glow from transparent particles along the pixel ray is accumulated towards the eye, as defined in the following equation:

$$I_\lambda(p) = \tau_\lambda \int_0^D c_\lambda(f(z))\rho_\lambda(f(z))dz \tag{3}$$

The transfer function $c_\lambda(s)$ is the color emitted by the field and $\rho_\lambda(s)$ maps scalars to particle density. The light intensity can be separated into its component wavelengths, denoted by λ. In the absorption only model, also called the x-ray model, light originating behind the volume is absorbed as its passes through the volume on its way to the eye, as illustrated in the following equation:

$$I_\lambda(p) = I_{\lambda,0} e^{-\tau_\lambda \int_0^D \rho_\lambda(f(z))dz} \tag{4}$$

Light intensity from the background $I_{\lambda,0}$ is attenuated by the accumulation of $\rho_\lambda(s)$, called the extinction coefficent, on the pixel ray. Both models are combined into the absorption plus emission model, shown in equation 6. The second term describes the accumulation of emitted light, but attenuated on its way to the eye.

$$I_\lambda(p) = I_{\lambda,0} e^{-\tau_\lambda \int_0^D \rho_\lambda(f(z))dz} + \int_0^D c_\lambda(f(z))\rho(f(z)) e^{-\tau_\lambda \int_0^z \rho_\lambda(f(u))du} dz \tag{5}$$

There is no closed form solution to this equation for arbitrary forms for function *f* and transfer functions $c_\lambda(s)$ and $\rho_\lambda(s)$.

2.2.2 Riemann Sum Integration

When the function *f* is piecewise constant, which is computed by most volume rendering approaches, the integral equations can be solved with a Riemann Sum approximation [48][61]. If resample locations are chosen at a constant sample spacing, Δs, on the pixel ray, the absorption plus emission equation is solved with the following discrete solution:

$$I_\lambda(p) = I_{\lambda,0}\prod_{i=0}^{n}\alpha(i\Delta s) + \sum_{i=0}^{n} C_\lambda(i\Delta s)\alpha(i\Delta s)\cdot\prod_{j=0}^{i-1}(1-\alpha(j\Delta s)) \tag{6}$$

The opacity for the *i*th segment is approximated using a Taylor's series expansion $\alpha(i\Delta s) = 1 - \exp(-\tau_\lambda\rho_\lambda(i\Delta s)\Delta s) = \tau_\lambda\rho_\lambda(i\Delta s)(\Delta s)$, where τ_λ is a constant.

2.2.3 Higher Order Integration

When the function *f* has higher order than piecewise constant, then the volume integral equation is more difficult to solve. Previous work has solved the volume integral equations when function *f* has a piecewise linear form. The integral is solved by computing the integral independently over each linear segment and then combining the solutions. Each linear segment is described by three parameter values (s_f, s_b, l). The value s_f is the scalar value at the front end of the segment, value s_b is the scalar value at the back end of the segment, and l is the length of the segment. When volume reconstruction is taken to be linear interpolation, as it usually is in practice, the intersection between the pixel ray and a volume grid defines function *f* such that the endpoints of the linear segments are at the cell faces. Figure 5 shows a pixel ray intersecting a tetrahedral grid. Cell projection rendering algorithms [89][100] use this formulation by projecting volume cells in depth sorted order.

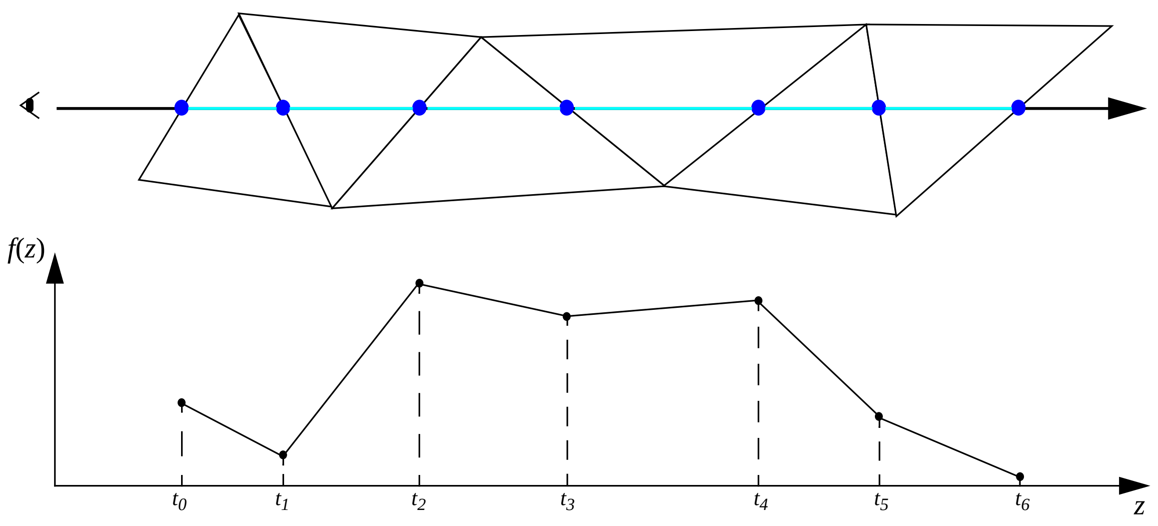


Figure 5. When the scalar field varies linearly within each volume cell, the scalar function on a pixel ray, $f(z)$, can be piecewise linear. The endpoints of the segments are at the ray-cell face intersections (top) shown at the bottom as distances t_0, ..., and t_6, from the eye.

Max [62] simplifies the volume integral of equation 6 by assuming constant color across the segment. If $\rho_\lambda(s)$ is an integrable and non-negative function, then the approximation $l\int_0^1 \rho((1-t)s_f + ts_b)dt = w \cdot S'(s_f, s_b)$ is used to compute alpha, where factor $w = l / |s_b - s_f|$ and S' is the integral of $\rho_\lambda(s)$ on the interval $[s_f, s_b]$. If $\rho_\lambda(s)$ is piecewise linear, then S' can be computed analytically using the trapezoid area formula [101]. If the indefinite integral $S(t) = \int_0^t \rho(u)du$ is computable, then $\rho_\lambda(s)$ can have a more general form and $S'(s_f, s_b) = S(s_b) - S(s_f)$.

The Projected Tetrahedra (PT) algorithm [89] projects each tetrahedral cell to the screen and classifies the projection in cases. A segment color and opacity is computed at the “thick” vertex, the projected vertex where the tetrahedron is thickest with respect to the view, and then graphics hardware interpolates color and alpha across the triangles. Two simple schemes are used to compute the integral at the thick vertex. The segment color is computed as: 1) evaluated at the front endpoint only or 2) the average at both endpoints of the segment. Alpha is approximated by the average of the extinction coefficients evaluated at both endpoints. A drawback with Shirley's approach is that linearly interpolating alpha computed at the triangle vertices causes Mach Banding artifacts. Stein [94] improves on this by computing alpha more accurately per-pixel across the triangles. The expression $1 - \exp(-\rho l)$ is stored in a texture and texturing mapping is used to lookup alpha as the segment length is linearly interpolated across the triangles. If ρ is assumed to be constant in the cell, then a 1D texture is constructed and l is interpolated linearly on the triangles by setting the texture coordinate of the thick vertex to ρl and zero at the thin vertices. If ρ varies linearly, a 2D texture is constructed and indexed by ρ and l. Stein also improves on the color approximation by assuming linear color in the cell and computes the color at the thick vertex using the trapezoid area formula on a piecewise linear transfer function [101]. Roettger [77] allows for arbitrary transfer functions on programmable PC graphics hardware. They construct a complete table for all possible linear segments that is indexed by (s_f, s_b, l) using a 3D texture lookup table. Since texture memory size is still limited on graphics boards, it is difficult to construct a high resolution table. Alternatively, they construct a 2D texture table, indexed by (s_f, s_b), which holds color and alpha for a normalized segment length $\tilde{l}$. The retrieved texel value is multiplied by $l/\tilde{l}$, which is assigned as the polygon color and interpolated across the triangle. This approximation may suffer from banding artifacts since alpha is interpolated linearly. Guthe [34] also uses programmable

PC graphics hardware and improves on Roettger's 2D texture solution. Alpha is computed with two texture tables. A 2D texture holds the average density for each possible pair of indices (s_f, s_b). This value is used in a dependent lookup on a 1D texture holding $1 - \exp(-x)$. Chromaticity is computed with a quadratic approximation. Engl [26] applies this formulation using 3D texture mapping and dependent texturing on programmable PC graphics hardware. They compute segment color and alpha in two ways. The first method stores color and alpha in a 2D texture indexed by (s_f, s_b) that is computed for a constant segment length l representing the average of all possible cell widths in the volume grid. The second approach stores the indefinite integrals $\rho_\lambda(s)$ and $c_\lambda(s)$ in a 2D texture indexed by (s_f, s_b). The factor w is applied in the register combiners. The function $c_\lambda(s)$ is allowed to be any integrable function except that the segment color does not include attenuation within the segment.

The volume integral equations become more difficult to solve when function f has degree greater than one because more expensive methods are required. Williams [101] solves the integral equations when both function f and the transfer functions are linear splines. Williams et al [102] solve the integral equations when both function f and the transfer functions are quadratic. The integration is computed over consecutive pairs of knots in the transfer functions. Williams [102] uses a five point Gaussian quadrature rule to approximate the integral.

2.3 Classification

The transfer function maps properties of the scalar field to optical properties. The most commonly used mappings in practice associate scalar values to color and opacity, which are the glow and extinction functions from Sabella's particle model. Other properties of the scalar field may be used in a mapping, such as the first derivative [47][41], to enhance perceived surfaces in the volume. Mappings may define shading properties, including ambient, diffuse, and specular.

The importance of classification is that transfer function instances can result in very different and relevant visualizations of the data. Choosing one or more appropriate transfer functions is not an easy task. In fact, it was called the "Holy Grail" of volume rendering by Blinn well over a decade ago. The problem is that the space of transfer functions is infinite. Searching this space for appropriate mappings that fit each user's differing needs is difficult. The approaches to search for transfer functions can be identified into three broad classes: automatic [4][36], semi-automatic [60][27], and manual methods [8][41][44]. The Transfer Function Bake-Off [73] surveys methods from classification. Automatic methods presume certain properties of the scalar field to be important. Semi-automatic methods present to the user a candidate set of mappings. The user judges the useful of the suggested mappings. The Volume Design Gallery [60] displays in "thumb-nail" format images of volume renderings from fixed viewpoints for automatically generated transfer functions. The user then chooses from the visualizations in the collection. Manual methods give freedom to the user to set and tweak transfer functions on-the-fly. This requires user-friendly interfaces and interactive rendering feedback. We present a user-driven framework in our work, though automatic and semi-automatic approaches are useful to guide the user in the search.

CHAPTER 3

VIEW-DEPENDENT RENDERING

View-dependent approaches have been employed to try to reduce the workload when rendering large data. These approaches are characterized by operations that reduce or simplify the data with respect to the user's current view parameters. Some approaches take advantage of incremental changes in the user's view such as zooming and frame-to-frame coherence. At a particular viewpoint, much of the data may not be visible and so view frustum and occlusion culling are key operations that may considerably reduce data size. When the user is far away or zooms out from the data, then a more simplified version of the data may be rendered with a much reduced cost. View-dependent approaches to render polygonal scenes have used mesh simplification [38] and Level-of-Detail (LOD) representations of the polygonal mesh to choose an appropriate mesh size to reduce render time.

3.1 Approaches to Volume Rendering

View-dependent approaches have been used in the context of volume rendering. Much focus has concentrated on the acceleration of isosurface rendering. Livnat [53] extracts a polygonal isosurface by only processing volume cells containing the visible isosurface. Both Livnat [53] and Gao [29] identify occluded portions of the isosurface. Parker [70] identifies the visible isosurface using early ray termination via raycasting. In direct volume rendering, Weiler [97] constructs an LOD representation of the volume to render using 3D texturing. Meredith [64] renders irregular volume data using a multiresolution approach and uses view-dependency to determine splat size. Gao [28] removes volume cells from consideration due to occlusion for their parallel volume renderer apriori to rendering using a collection of view samples that define a Plenoptic Opacity Function given the opacity transfer function. The view samples are chosen to cover the view space well.

View-dependent approaches have been presented to allow the user to change classification on-the-fly while keeping the user's viewpoint at a chosen fixed location. These approaches store per-pixel scalar information reconstructed on pixel rays. Botha [8] pre-computes a histogram of the scalar values along a pixel ray. The user then picks a voxel-thick image-aligned reconstructed slice of the volume. Given a transfer function instance, simple segmentation is performed in the per-pixel histograms and the slice is color classified with a single homogeneous material and then blended with the grey level slice reconstruction. The resulting image serves as a preview to full raycasting. Kaneda [40] represents the scalar function along a pixel ray as a Fourier Series expansion. Per-pixel basis functions are pre-computed and accumulated into an array of basis images. User-specified color maps (changing opacity is not supported) determine the series coefficients. The final image is computed by adding up the weighted basis images. Srivastava [92] pre-compute equally spaced scalar samples along the pixel rays, as in volume raycasting. Samples are then classified, shaded, and alpha composited together using a fast CPU. Furthermore, scalars below a threshold, e.g. representing air, are thrown away and a depth is stored with each sample. Early ray termination is possible if the transfer function values accumulate to full opacity.

3.2 Image-based Rendering

IBR encompasses both specialized data structures and methods that render from pre-acquired scene projections at novel views. These methods may be used to render large and complex scenes where traditional rendering directly from the scene primitives is too expensive, e.g. rendering millions of polygons per frame. Instead, IBR methods use image data structures as the main rendering primitive where rendering speedups by orders of magnitude over conventional rendering have been reported in most instances. The basis for this new rendering paradigm is that an image captures much detail using a simple data structure. Early examples of image-based rendering are pre-computed movies and an extension called Movie Maps [51], which allowed more freedom for user movement. The rendering paradigm has its roots in texture mapping, which provides an inexpensive way to add unmatched realism to objects without the need for complex geometric modelling. An example is billboarding, which was used in flight simulators, games, and architectural walkthrough applications. The projection of a complex object that is considered to be a background object, e.g. trees, shrubs, furniture, etc., is texture mapped onto a quadrilateral that is positioned at the center of the object's location in the scene. The quad is rotated so that it is always facing the viewer. The alpha component of the texture may be used to show objects lying behind the billboard.

Image-based data structures have a number of advantages. Algorithms that operate on these data structures are computationally bounded by image size and not by scene complexity. Expensive geometric transformations and shading operations can be avoided altogether. Projection information contained within the image data may be mapped to novel views as texture primitives where standard texture mapping on todays graphics hardware on the PC is extremely fast. Only a small subset of the entire pre-acquired image set is needed to render a novel view. In addition, images are amenable to compression and transmission over a network.

Each IBR method can be described by how it approximates a theoretical framework called the Plenoptic Function [65], $P(x, y, z, \theta, \sigma, \lambda, t)$, a 7-dimensional description of the flow of light in a scene. The equation describes the light impinging on a point (x, y, z) in space in the direction (θ, σ) with wavelength λ at time t. IBR is formulated as a two-step problem. The first problem is how to sample the Plenoptic Function, also called *Plenoptic Modelling* or *View Sampling*. Plenoptic samples, also called *view samples*, are located and constructed to adequately cover the view space. A *complete* plenoptic sample is a full spherical map about the view sample. An *incomplete* plenoptic sample captures a solid angle subset of a spherical map. The environment map is a good example of a plenoptic sample. The cube map [33][32][31] is an example of a construction for a complete plenoptic sample and a panoramic defined on a cylinder [15][65] is a construction for an incomplete plenoptic sample. A view sample has been also been called a *reference view* or *key view* in the literature. The second problem is how to render at a novel view, also called an off-sample view, from one or more view samples. This problem is concerned with how to reconstruct the Plenoptic Function, also called *view reconstruction*. View reconstruction algorithms typically employ pixel warping followed by image reconstruction on the target pixel plane.

At a minimum, the information stored at a view sample is color information along with the parameters of the camera. When the scene is assumed to consist of opaque objects placed in free space, the color projected to a view sample represents a surface location. Levoy [49] identifies augmented representations that may include depth (in the form of range images) and opacity information. The following surveys [90][69][107] provide a classification of most image-based representations and associated techniques. Most IBR techniques assume a scene with opaque objects placed in free space. Early image-based representations captured only color at a plenoptic sample. QuickTime VR [15] and Image Mosaicking [65] warp pixels from panorama so that the user is immersed in the scene while fixed at the center of projection. They allow only limited user view changes. Lumigraph [31] and Light Field Rendering [50] reduce the Plenoptic Function to 4D using a parameterzation of the light rays on two 2D slices.

Depth information has been used to allow for motion parallax effects in the view reconstruction. Image correspondences have been used to extract depth information using optical flow [16], epipolar geometry [65], and view morphing [85]. Image-based representations that store depth explicitly [63][59][75] must capture this information using a range scanner or extract it from a Z-buffer. Methods that use these representations need to address the problem of occlusion/disocclusion artifacts. The Layered Depth Image (LDI) [86] stores multiple samples per pixel sorted in depth order and is able to overcome these artifacts. These methods must be able to fill holes at the target novel view since pixel warping is a many-to-one mapping.

A collection of image-based representations extend billboarding to representations called *image imposters* [57][82] where image-based representations are constructed for single or clusters of objects in the scene. Maciel [57] assumes that the user's view is far away from the object and pre-computes imposters at view directions on the spherical view space surrounding the object. One set of imposters is acquired at the six faces of the object's bounding box. These are texture mapped onto the bounding box faces or used as billboards. Another set is acquired at the intersections of longitude and latitude lines on the sphere. Work in [82][3] construct imposters for groups of objects, called texture clusters. Schaufler [82] places the imposter at the object or cluster center and rotates it to face the viewer at successive viewpoints. An error metric is used to re-computed an imposter on-the-fly when the user's view changes too much. A multi-layered imposter [63][83][21] is an array of billboards that hold projections of an object at different depth. They have been used to provide improved motion parallax so that novel views further away from a view sample may be reconstructed. The main idea is that pixels at similar depth move together in the warp. Shade [86] augment a billboard with depth per-pixel to define a depth sprite. Debevec [20] use projective texture mapping to texture proxy geometry that is a much simplified approximation to the original geometry.

Two other issues that are important to image-based rendering systems are image caching and compression. Acquired imagery must be cached and re-used during rendering. A number of image caching schemes [84][87][46]have been presented to hold image imposters. The cache is refreshed with new imposters as the user's view changes. Pre-acquired imagery can be quite large, on the order of gigabytes or more. Compression schemes have been presented to reduce the size of the image database on disk.

3.2.1 Image-based Volume Rendering

Image-based data structures and methods have been developed for volume rendering. Choi and Shin [17] store axis-aligned base plane projection images that result from the shear-warp factorization at view samples. View reconstruction requires a warp from the nearest view sample. Chen [14] determine proxy geometry by computing an outer shell surface from the volume using the opacity transfer function. A volume rendering at each view sample is texture mapped onto the proxy and projected to novel views. Raycasting is used to fill holes. Brady [9] and Mueller [67] present a layered imposter representation that stores a pre-integrated slab of the volume at each layer, which holds both color and opacity. The opacity stored at each pixel layer is used to alpha blend the layers to novel views for an approximate volume rendering. Brady allows for perspective projection for inside views of the volume by constructing image layers with raycasting. Mueller computes orthogonal projections for outside views of the volume using splatting and use non-planar billboard geometry for improved motion parallax. Yoon [105] stores per-pixel scalar information, called a “volume isomap”, where the isovalue is searched for in the reference image when rendering an isosurface at a novel view. Shareef [88] stores the minimum and maximum scalar values per-pixel at a view sample to compute MIP with the ability to render depth shaded MIP and arbitrary transfer functions. LaMar [46] and Yoon [106] use an image cache to hold reference images.

CHAPTER 4

REMOTE VOLUME RENDERING

The problem of how to render large scale volume data at interactive frame rates when the volume contains on the order of billions, or more, of voxels has been a hot topic of research for a number of years now. How to do this in a remote visualization setting has been an active area of research only recently due to recent advances in technology related to infrastructure and the advent of the internet. The client-server paradigm introduces unique challenges to the problem where the main issue is how to assign the different tasks in the rendering pipeline to both sides in order to optimally utilize available resources.

A classification of remote rendering frameworks is given by Luke [56] that is separated into four possible scenarios. In the first, also referred to as *render-remote*, a thin client simply displays the image received from the server. The volume data is stored and rendered on the server side with state-of-the-art hardware. This may come in the form of a supercomputer or a cluster of workstations for software rendering [70], or a more cheaper alternative using a PC cluster equipped with graphics hardware [58][68]. A change in any visualization parameter requires a round-trip to the server to compute a new rendering. Some form of parallelization is necessary for one or more of the three main bottlenecks in the rendering pipeline, i.e. compute power, I/O throughput, and network bandwith. Both network and I/O still pose the most serious bottlenecks. A promising research direction that is being pursued today is to build a system from a PC cluster equipped with 3D texture hardware due its speedup and scalability advantages coupled with its low cost [42]. Though, it should be kept in mind that algorithms based on 3D texture hardware are limited to rendering rectilinear volume grids. The direct opposite of this scenario is to have the client perform all the rendering tasks. The server functions to only store and send volume data to the client. This framework, also called *render-local*, comes in two flavors. One is to download the entire volume dataset to the client, which is infeasible when dealing with large scale data. The other has the server calculate over the data to send representations of the volume, e.g. a mipmap volume, octree subdivision, or wavelet representation, that won't overwhelm the client. Guthe [35] uses wavelets and an octree subdivision to compute a level of detail representation of the volume. It is compressed on the server and then decompressed in blocks and rendered on the client side using view-dependent heuristics. In order to avoid extensive network traffic to transmit sizable amounts of data, reported methods inevitably compress the volume in a lossy manner.

The remaining scenario is to divide the rendering tasks between the client and server. Todays PC has experienced an order of magnitude increase in terms of processor speed, memory size, disk storage, and commodity graphics hardware speed and functionality. In addition, large display technology is available at an affordable price, e.g. plasma displays, tiled displays, and virtual workbenches. Luke [56] describes a framework for isosurface visualization where the server renders from a viewpoint with depth buffer writing turned on. The server reads out the depth buffer and creates a mesh, similar to a height field. The color buffer is also read out, and both the mesh and image are sent to the client. The client texture maps the image onto the mesh, called *ZTex* rendering, using 2D texture mapping hardware for final display. The depth information encoded in the mesh provides motion parallax information. Using the property of frame-to-frame coherence, the construction can be re-projected to nearby viewpoints for fast approximate renderings. Bethel [5] also leverages this coherence property for direct volume rendering with their Image-based Rendering Assisted Volume Rendering (IBRAVR) framework. Using the Visipult [5][6] rendering architecture on the server side, Bethel combines the capabilities of a state-of-the-art parallel renderer with off-the-shelf PC texture mapping hardware found on the client, as illustrated in Figure 6. The server side renderer partitions the volume into depth-sorted view-aligned slabs and renders each slab into a separate image. These images are sent through parallel I/O pipes to the client, which performs the remaining color-alpha compositing step using 2D texture mapping hardware. The depth sorted images, a construction presented by Brady [9] and Mueller [67], are re-projected quickly to nearby views. Both the *ZTex* and *IBRAVR* methods hide rendering latency due to the back end only when the user's viewpoint changes. When non-viewpoint parameters change, a round-trip is required to the server to update the meshes and textures.

Yoon [106] implements an image cache, an idea introduced by Shade [87], on the client side to hold one or more computed view samples in their isosurface visualization system. With many view samples in the cache, the user may browse a larger space of vantage points at high frame rates without costly round-trip requests to the server. A view sample holds both a color and depth value per pixel, which are the color and distance to the nearest isosurface, respectively. A rendering is computed from a cached view sample to the

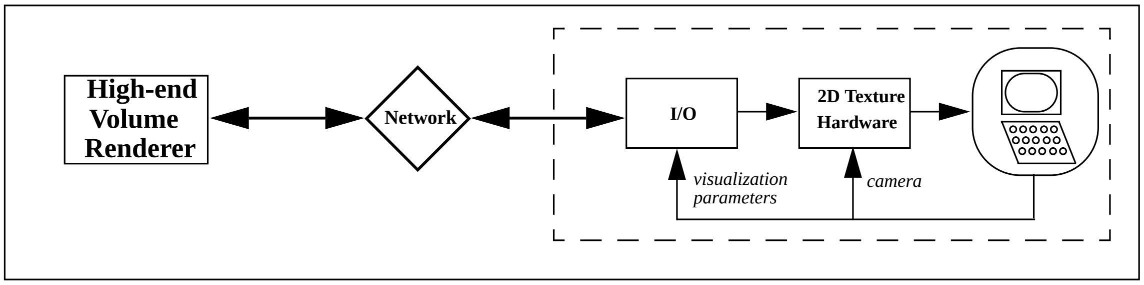


Figure 6. The remote visualization framework by Bethel called IBRAVR.

nearby users view warping the pixel samples. A fast algorithm is used to search for the isosurface in the view sample for every pixel in the final image. As is the case with most image warping techniques, holes can occur in the final image. Volume raycasting is performed at these pixels to fill them, which presumably will be a small number. If the user decides to change non-viewpoint parameters, such as the isovalue, the images in the cache are unusable. Yoon [105] introduces the *isomap* data structure to overcome this problem. Each pixel in the isomap holds a piecewise linear approximation of the scalar function along the pixel ray for only the potentially visible isovalues. The segments are monotonically increasing and so a binary search finds the distance to a user chosen isovalue along the ray on-the-fly.

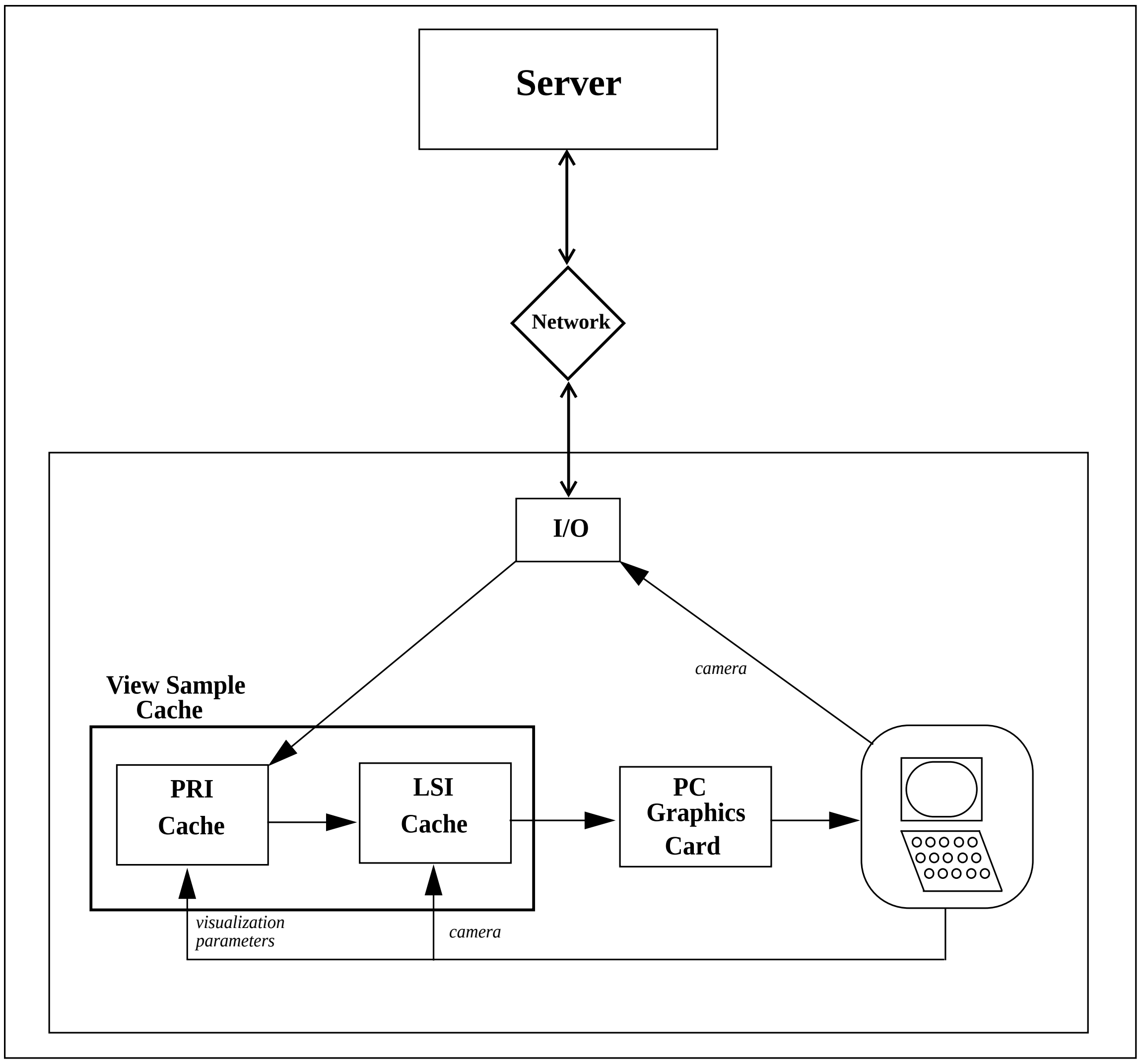


Figure 7. A new remote visualization framework for direct volume rendering of large scale data.

CHAPTER 5

A NOVEL END-TO-END PIPELINE

We present a novel end-to-end rendering pipeline for direct volume rendering that builds upon previous work. Our framework, shown in Figure 7, allows for fast browsing, both spatially and when changing visualization parameters. The client runs a volume browser that maintains a cache of view samples and computes renderings on-the-fly. A view sample holds its camera information and two view-dependent data structures. The first data structure is a Pixel Ray Image (PRI), presented in chapter 6, which holds per-pixel scalar information. As in raycasting, the intersection of a projected ray with the volume defines a 1D function. Associated with each PRI is a Layered Slab Image (LSI), which is a depth-sorted array of images [9][67]. The view frustum of the PRI is divided into image-aligned slabs and each array image holds the rendering of a single slab. Whenever the viewpoint changes only, these images are projected to the nearby views using an image-based approach that utilizes simple 2D texture mapping with alpha blending to quickly display an approximate rendering of the data. When another visualization parameter changes, such the transfer function, the cached LSI are invalid, so a new LSI set needs to be computed, starting with the one nearest to the user's viewpoint in the cache. In the following chapters, we present the data structures for the PRI and LSI and associated rendering algorithms.

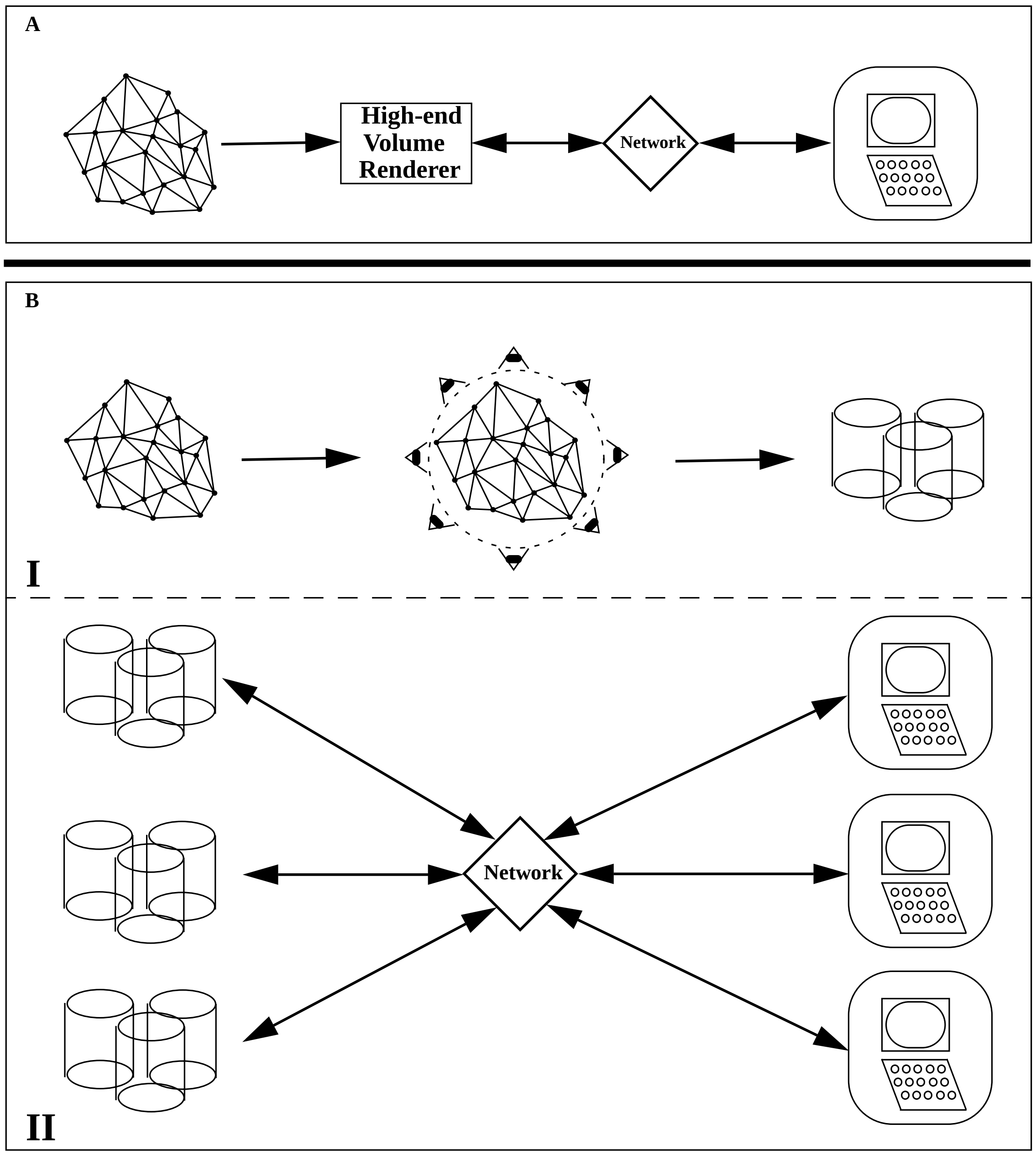


Figure 8. Two schemes to supply PRI to our volume browser. The server may run a state-of-the-art volume renderer, above, which can typically support a single user viewing a single dataset. A two-stage approach, bottom, pre-computes PRI using state-of-the-art technology off-line and stores them in a database. This scheme supports multiple datasets and multiple users simultaneously.

It is the job of the server to replenish the view sample cache with PRI’s that are at or close to the user’s current viewpoint. Two factors that will determine whether the cache contains the most relevant view samples are the latency due to the back-end, coupled with the user’s viewpoint changes. We propose two situations on how the server can supply PRI, shown in Figure 8. In the first, the server is a high-end volume renderer that computes PRI on-the-fly, shown in Figure 8a. Since the PRI only holds scalar information, the renderer is configured to perform only volume resampling and reconstruction.

A second option is to decouple the volume renderer entirely from the volume browser. A two step process is illustrated in Figure 8b. First, all PRI’s to be used in the rendering are pre-computed off-line with a state-of-the-art volume renderer and stored in a database placed on a fast data server. This may take days or even weeks for datasets on the order of terabytes. During volume browsing, the PRI closest to the user’s current view are retrieved from the server (bottom of Figure 8b). If the entire database can fit in secondary storage on the client side, it can be downloaded first. By removing the dependence of the rendering on both the original volume data and a state-of-the-art renderer, this scheme easily supports volume browsing by multiple users for multiple datasets simultaneously, see Figure 8b bottom.

The situation in Figure 8a allows the user to move freely in the view space. The stipulation is that the back-end renderer is able to respond quick enough to replenish the cache with nearby view samples. The situation in Figure 8b has computed view samples a priori, so the workload on the server side is greatly reduced. The problem is how to sufficiently populate the view space with view samples. This is not a straightforward problem and

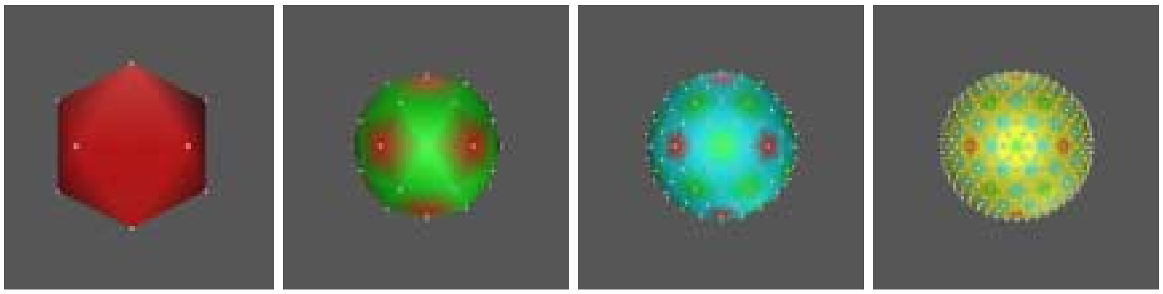


Figure 9. First three iterations of a midpoint subvision algorithm to compute a polygonal mesh of same size equilateral triangles whose vertex locations are used to locate view samples. The subdivision starts from the icosahedron, left, and progresses to the right.

must be addressed by an image-based rendering solution. Since the user's view will be restricted to the portion of the view space where samples are found, it is clear that samples should be available at the interesting portions of the scene. In a static and well known scene, this is a much easier task. In a volume browsing application where transfer functions are allowed to change, no information is known beforehand.

One of the most common ways to view and explore volume data is in an object-centered viewing mode. The user is allowed to reside on a sphere surrounding the volume and to rotate and zoom, similar to a trackball interface. The problem becomes how to place view samples on the sphere. One scheme is to locate samples at the intersections between longitude and latitude lines defined on the sphere. The distribution of the samples is usually uneven and crowded at the poles. A geodesic sphere construction [24] locates vertices evenly in a hierarchical triangle tessellation of the sphere. It is a triangle subdivision approach that creates a tessellation using equilateral triangles of the same size. The algorithm starts the subdivision from one of the first three platonic solids, i.e. the tetrahedron, octahedron, or icosahedron, since these are constructed from the same size equilateral triangles. At each iteration, each triangle is subdivided into four smaller ones by inserting new vertices at the midpoint of every edge. The vertices of the resulting mesh are projected to the sphere in a radial fashion. Figure 9 illustrates the first three subdivision steps on the icosahedron. Vertex locations are denoted by white dots and are color coded with red, green, blue, and and then yellow, respectively, to denote the iteration that the vertex was inserted. At each iteration, the vertices tessellate the entire sphere and each successive iteration adds more vertices. This provides a coarse to fine hierarchy of vertices that populate the entire view space so that the sample count can vary. A particular hierarchy of PRI that fit on the client side may be downloaded to the client.

CHAPTER 6

PRI: A VIEW-DEPENDENT PER-PIXEL SCALAR REPRESENTATION

The Pixel Ray Image (PRI) is a view sample that holds view-dependent scalar information at each pixel. It is defined with a virtual camera parameterization: projection point, projection direction (we assume orthogonal projection), orientation, frustum, and image plane. The image plane is pixelized where pixel locations lie on a cartesian grid formed by two orthonormal basis vectors $\vec{i}$ and $\vec{j}$, i.e. $p(x, y) = x\vec{i} + y\vec{j}$, where x and y are integers. An infinite ray, called a sampling ray, is projected from every pixel center towards the projection direction. The intersection of a ray at pixel (x, y) with the volume defines a 1D scalar function, call it $f_{(x, y)}(t)$. The parameter t is the distance along the ray from the image plane in the projection direction, where $0 \le t_{in} \le t \le t_{out} \le D$. At $t = 0$ the function is evaluated at the pixel center. The values t_{in} and t_{out} are the distances where the ray enters and leaves the volume, respectively. The distance D is placed behind the volume to terminate all the infinite sampling rays. Only the scalar information defined on the portion of the ray intersecting the volume is included in the PRI. Figure 10 illustrates the PRI and a 1D function associated with a pixel's sampling ray. This is a parameterization similar to the Plenoptic function. The main difference is that we store scene information per ray rather than illumination information.

We need to choose a representation for function $f_{(x, y)}(t)$ that can be stored and computed over, call it $\tilde{f}_{(x,y)}(t)$. One possible representation for $\tilde{f}_{(x,y)}(t)$ is to define equidistant spaced resample points along the sampling ray, as is done in volume raycasting. The main drawback of this representation is that the sample distance needs to be chosen very small to accurately represent the scalar field. The result is a revoxelization of the volume at every PRI sample. We define two representations in the following sections based upon the following two criteria: representation size and functional approximation accuracy. Our goal is to render the data from the PRI samples. In order to accomplish this, the scalar function $\tilde{f}_{(x,y)}(t)$ is applied to a projection function P along with other visualization parameters denoted by $\lambda(s)$, as shown in equation (8). This function defines both the classification and projection steps in volume rendering. If the user's camera is defined to coincide with the PRI, then a color can be computed for each pixel by an evaluation of this equation. To address the first criterion we use the fact that the size of a PRI sample depends upon both the number of pixels in the sample and the representation size of each per-pixel scalar function. Since we are caching PRI in main memory, which may contain hundreds to thousands of samples, a PRI should have a small memory footprint size. In addition, a PRI will have to be transmitted over low speed networks to the client side. The size of the representation of the PRI also determines how fast a rendering can be computed during browsing. The second criterion dictates the image quality of the rendering because the accuracy when computing function P depends upon the accuracy of the input function $\tilde{f}_{(x,y)}(t)$.

$$color(x, y) = P(\tilde{f}_{(x,y)}(t), \lambda(s)) \tag{7}$$

The representations for function $\tilde{f}_{(x,y)}(t)$, which we present in the following sections,

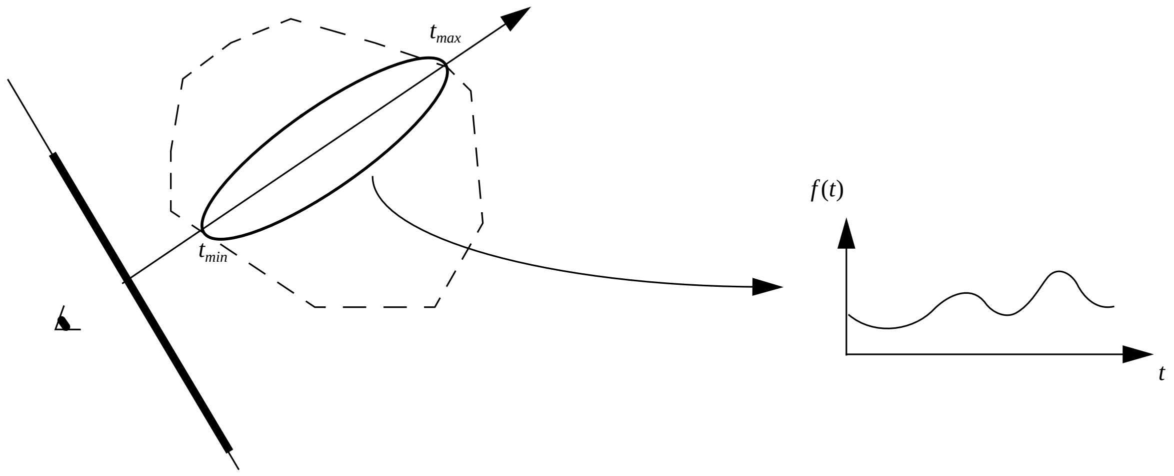


Figure 10. A PRI is a view sample that holds a representation of a 1D scalar function per-pixel on the image plane. Each function, an example is shown at the right, represents the scalar values that project to a particular pixel on the pixelized image plane.

that we use satisfy both of these criteria. The representations are chosen based upon the requirements of each of the projection equations. Our first representation is used to compute the maximum projection equations. It stores a scalar value pair that represents the minimum and maximum of function $f_{(x, y)}(t)$, call the pair (f_{min}, f_{max}). The other projection equations require a complete description of the scalar function along the sampling ray. Our second representation is a linear spline representation for function $\tilde{f}_{(x, y)}(t)$.

6.1 PRI Scalar Reconstruction

Constructing either PRI type, MM or LS, requires resampling the volume along each sampling ray. We identify two ways to collect samples in a volume that is defined either analytically or on a sampling grid. The first approach is to define equidistant spaced resample points along the sampling ray where the sample spacing ensures that the volume is sampled above the Nyquist limit. The second approach is applied to volumes defined on a sampling grid (structured or unstructured) where scalar interpolation within a volume cell is assumed to be linear. In this case, resample points are located at ray-cell face intersections. Given a list of samples retrieved from either approach, we will assume that the scalar function is linear between adjacent samples. The list of samples is a knot list that defines a linear spline.

6.2 Transfer Function Representations

We compute a rendering from a PRI at each pixel from the stored function $\tilde{f}_{(x,y)}$ and the transfer function $T(s)$, where s is a scalar value. The task becomes how to efficiently evaluate the function $T(\tilde{f}_{(x,y)})$ in the projection equation. There are two representations to store the transfer function in tabular form that are commonly used in practice. The first representation stores equidistant spaced samples on the transfer function domain in a 1D array, call it T_{const}. A transfer function value is evaluated at a given scalar value using either nearest neighbor or linear interpolation between two consecutive samples. A second representation stores unevenly spaced samples on the transfer function domain, which are knots in a linear spline approximation of the function, called T_{lin}. The list of knots are stored in a 1D array that is sorted according to location of the knot on the transfer function domain. Each table value holds a value pair, (t_i, I_i), where t_i is the location of the knot on the transfer function domain and I_i is the function value. The function is evaluated at a given scalar value with a binary search on the locations of the knots on the function domain. The T_{lin} representation will usually have much fewer samples than T_{const}. Both representations may be used to store a pre-integrated transfer function.

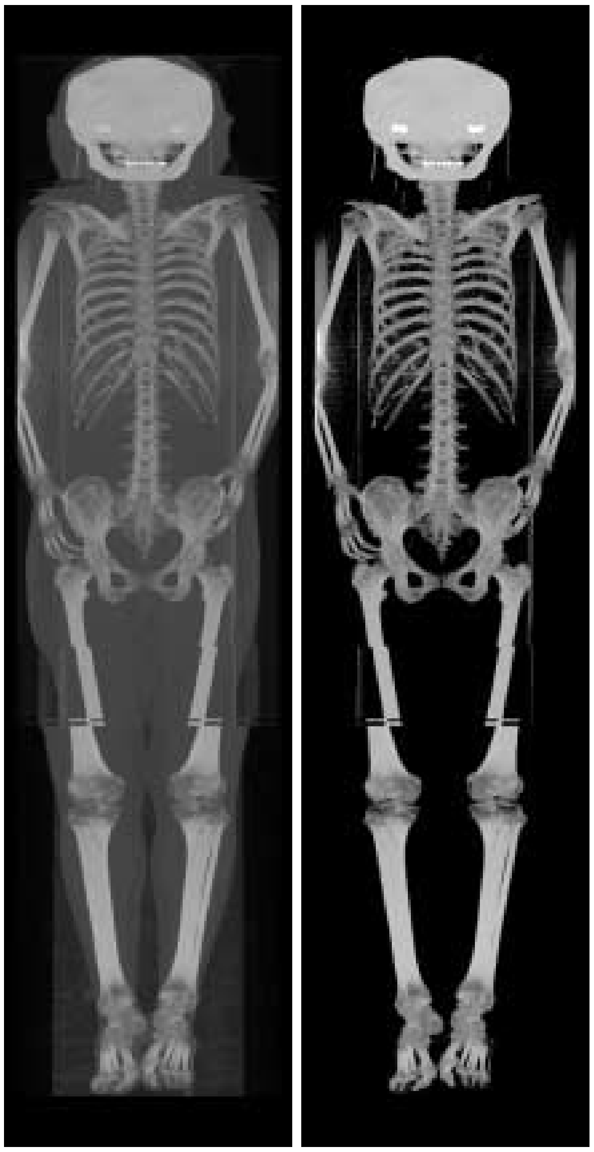
CHAPTER 7

MM-PRI

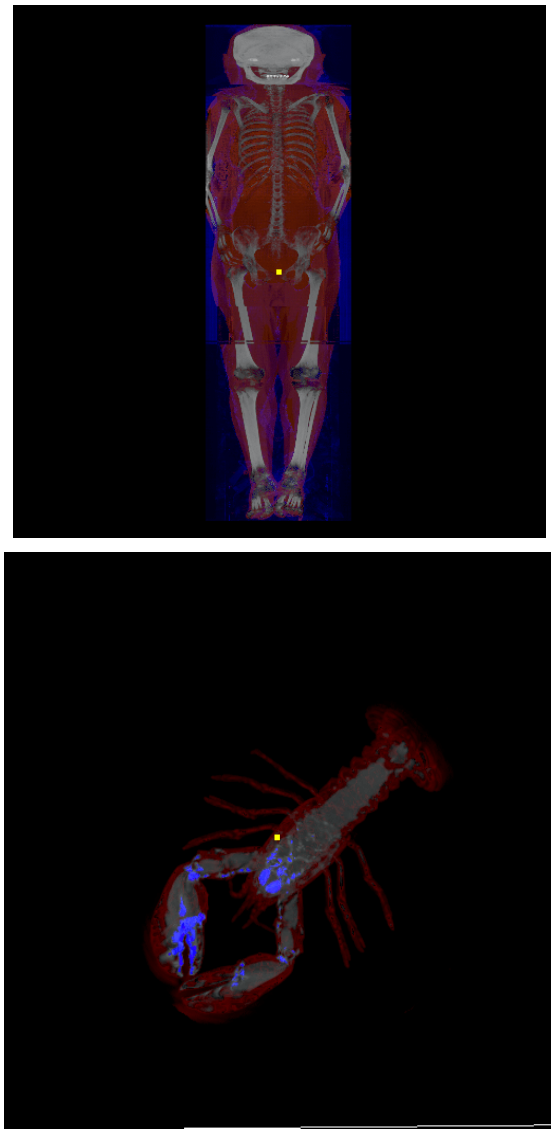
The solution to the maximum projection equation requires a search for the maximal value along the pixel ray, as shown in equation (2) (Section 2.1.3). This search is performed on the portion of the ray generated at pixel (x, y) that intersects the volume:

$$color(x, y) = \max_{t_{in} \leq t \leq t_{out}} (T(f_{(x, y)}(t))) \qquad (8)$$

where t_{in} and t_{out} are the distances along the pixel ray where it enters and leaves the volume. In MIP rendering, the transfer function T is defined as in equation (1) (Section 2.1.1) and returns a grey level intensity. Other mapping functions can provide useful visualizations in a maximum projection. For instance, Drebin [22] uses a maximum likelihood classifier on CT data to distinguish material in a solution to volume integration. Figure 11a shows a comparison between a MIP rendering and a MIP rendering using a different classification mapping that shows bone material from the CT data of the Visible Female dataset. In Maximum Opacity Projection (MOP) rendering, the transfer function returns a color that can provide an extra degree of freedom to distinguish material, as shown in Figure 11b.



(a)

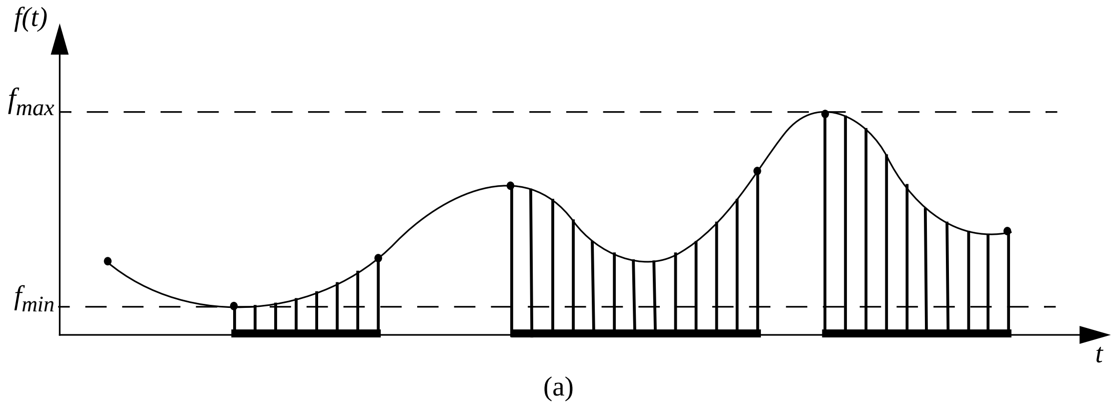


(b)

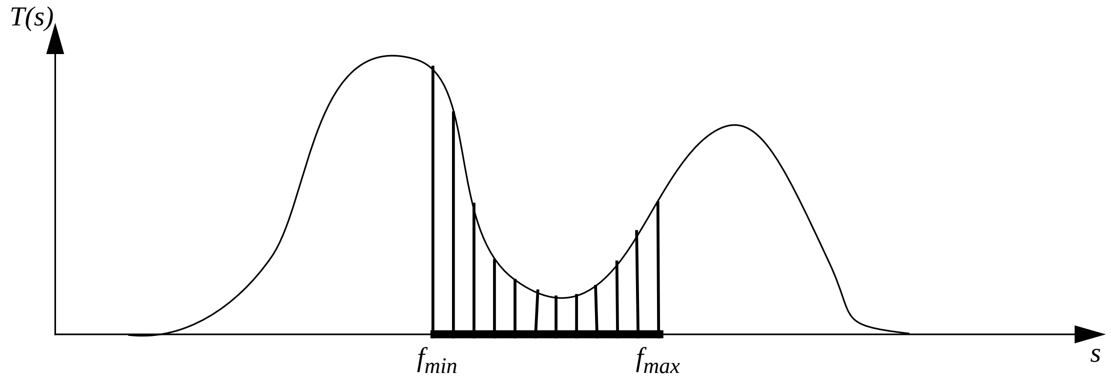
Figure 11. (a) A comparison between MIP rendering (left) and MIP rendering using a different transfer function (right) that shows bone material from the Visible Female dataset. (b) MOP rendering is able to assign color material.

We derive a compact representation for function $f_{(x,y)}(t)$ from equation (9). A straightforward way to solve this equation is to search on the pixel ray for the maximum classified value. For each location *t* on the ray, where $t_{in} \le t \le t_{out}$, the scalar function is evaluated first followed by the transfer function since the final function value applied to the maximum operator is the function $H(t) = T(f(t))$, mathematically a composite function. There are two drawbacks to this procedure: 1) It requires a complete description of the 1D function on the pixel ray and 2) the transfer function is evaluated multiple times for the same scalar value. The consequence of the latter drawback is inefficiency because redundant values are applied to the maximum operator. We illustrate these two drawbacks in Figure 12a. Consider a 1D scalar function and traverse the function from left to right. If we remove the subintervals of the function in its domain, denoted by the filled areas, that evaluate to the same scalar values we encountered earlier in the traversal, only the remaining function values are needed to solve equation (9). These function values define an interval of scalar values between the global minimum and maximum function values, i.e. $[f_{min}, f_{max}]$. Equation (9) can be re-written using the observation that it can be alternatively solved by a search in the transfer function space, as shown in the equation (10). A search in the transfer function space is performed on the domain subinterval defined by the scalar interval $[f_{min}, f_{max}]$, as illustrated in Figure 12b. The MM-PRI only needs to store this value pair per-pixel as the representation $\tilde{f}_{(x,y)}(t)$ in order to solve the projection equation.

$$\max_{t_{in} \le t \le t_{out}} (H(t)) = \max_{f_{min} \le s \le f_{max}} (T(s)) \tag{9}$$



(a)



(b)

Figure 12. We re-formulate the solution to the maximum projection equation by applying searching in classification space rather than along the pixel ray. Figure (a) illustrates the redundant evaluations (red portions) on the transfer function when searching on the pixel ray. It is most efficient to search in classification space (Figure (b)) on the domain subinterval $[f_{min}, f_{max}]$ obtained from the 1D scalar function.

7.1 MM-PRI Construction

The value pair $[f_{min}, f_{max}]$ at each pixel is found by searching for the function extrema on the knot list that is computed on the sampling ray using raycasting, cell projection, or a depth peeling approach. The function extrema will necessarily lie on knot locations with linear interpolation.

7.2 Fast Maximum Projection

Our re-formulation of the maximum projection equation solves the projection equation as a search in classification space. The search is restricted to the function domain subinterval $[f_{min}, f_{max}]$. We call this value pair an *interval index* because it will be used to evaluate the maximum classified value from the transfer function table. The search procedure should be done efficiently because the projection equation is solved at a million pixels or more. We construct a transfer function table, call it $T_{max}(s_1, s_2)$, that returns the maximum transfer function value on the function domain subinterval $[s_1, s_2]$, where $s_1 \le s_2$.

7.2.1 0-degree Spline Transfer Functions

Consider the case in which the transfer function is represented as the form T_{const}. In the following sections, we present three ways to represent the transfer function table $T_{max}(s_1, s_2)$., called the *sample table, interval table,* and the *interval tree table*. The tables are compared with respect to table size and search time.

7.2.1.1 Sample Table

The sample table stores two arrays of values. The first array holds the *n* function samples in a 1D array, called the *sample array,* which is arranged by increasing scalar value. The function maximum may be found with a linear search on those entries in the sample array that overlap the interval. The average search time may be reduced for large sample sizes and interval ranges with a second 1D array, called the *maxima array,* that holds all occurrences of local maxima arranged by increasing scalar value. This array will typically be much smaller than the sample array.

The transfer function maximum value, given the interval index, is found with the following procedure. The interval is first applied to the maxima array. If the interval overlaps at least one of the pre-computed local maxima found in this array, then the maximum transfer function value is the largest between these overlapping maxima and the function values at f_{min} and f_{max}. Otherwise, the maximum transfer function value is the larger of the function values stored in the sample array at f_{min} and f_{max}. We identify local maxima from the discrete samples in the sample array using the following three conditions. If any one of these conditions holds, then the sample is a local maxima and inserted into the maxima array. A sample *i* in the sample array is a local maxima if: 1) $T[i-1] < T[i]$ and $T[i+1] < T[i]$, 2) $T[i-1] < T[i]$ and $T[i] = T[i+1]$, or 3) $T[i-1] = T[i]$ and $T[i] > T[i+1]$, where *T* is the transfer function value found in the sample array. The following procedure efficiently finds the maximum transfer function value:

```
1. $l_{max}$ = local maximum associated with smallest scalar value greater or equal to $f_{min}$.
2. If (no such local maximum exists or the scalar location of lmax is greater than $f_{max}$)
      return max($T[f_{min}]$, $T[f_{max}]$);
3. return the maximum of $T[f_{min}]$, $T[f_{max}]$, and all local maxima between $f_{min}$ and $f_{max}$.
```

The local maximum l_{max} is found with a binary search on the maxima array using f_{min} as the search key. In step 2, if no local maximum lies in the interval (f_{min}, f_{max}) then the larger of the transfer function evaluated at f_{min} and at f_{max} is returned. If a local maximum is found, then the largest local maximum in this scalar range is found by a linear search in the maxima array in step 3.

7.2.1.2 Interval Table

The local maximum can be found in constant time with a construction we call the *interval table*. This table will hold the maximum function values for every possible subinterval on the discretized transfer function domain. A subinterval is denoted by a scalar pair (s_{min}, s_{max}) where $s_{min} \leq s_{max}$ and its length is the number of scalar samples that fall within the interval. The interval table is a 1D array where each entry represents a single subinterval. The subintervals are arranged in sorted order in the array by interval length and then by increasing s_{min} value. The value stored in an array entry is the maximum transfer function value over the represented scalar interval. To retrieve the maximum transfer function value, a table index is constructed from the interval index using the interval's length and minimum scalar value, i.e. $(f_{max} - f_{min}, f_{min})$. When the number of scalar samples in the transfer function domain is large, say for high precision datasets, the interval table grows in size quickly. For example, if the scalar precision is 16-bits, the interval table will require over 2 billion entries.

7.2.1.3 Interval Tree Table

A compromise between the sample and interval tables for a large number of samples of the transfer function is to use a binary tree data structure, called an *interval tree* [25]. The samples on the domain of the transfer function are successively split into half intervals at the middle. A binary tree is constructed from this subdivision. The root node holds the maximum function value over the entire function domain. An internal or leaf node holds the maximum function value on a half interval of the interval represented in its parent node. Every tree node holds the quadruple (l, r, m, c), where the interval $[l, r]$ represents a function domain interval and $l \leq r$, m is the midpoint of this interval, and c is the maximum transfer function value over the interval. The left child of an internal node represents the interval (l, m) and the right child represents interval (m, r). A leaf node has $l = r = m$.

The maximum transfer function value is found with a top-down tree traversal given an interval. A list, call it *R*, is maintained during the search that holds retrieved function values and is intially the empty list. At each node visited in the search, if the interval is equal to the interval represented at the node, then *c* is added to list *R* and continued traversal ends down this tree branch. If the interval is contained entirely within one of the two half intervals of the node, then traversal continues down the corresponding left or right branch with this interval. If the interval overlaps the midpoint, then the index is split at *m* and traversal continues down both tree branches with two new interval indexes that result from a split of the interval index at the midpoint. When all subtree traversals are terminated, the function maximum is found in list *R* by a linear search. The value retrieved from list R is the maximum function value over the original interval index at the start of the tree search because the concatenation of the subintervals from which the values in list R were retrieved is equal to the original the interval index. Table 1 shows the search time and table sizes for each of the three types of transfer function tables and that the interval tree table is a good compromise in terms of both search time and table size.

Type	**Search Cost**	**Size**
sample	$O(n)$	n
interval	$O(1)$	$(n^2 + n)/2$
tree	$O(\log n)$	$2n - 1$

Table 1 Search costs and table sizes for piecewise constant transfer functions.

7.2.2 1-degree Spline Transfer Functions

Consider the case in which the transfer function has a linear spline form, which is described by a list of knot values or T_{lin} format. The maximum transfer function value over any domain subinterval is found at a knot or function interpolation between neighboring knots. Given an interval index (f_{min}, f_{max}), the maximum function value on the interval is found with a binary search on the knots. Similar to the sample table, a sample array and maxima array may be used here. The former holds the complete linear spline. That latter holds local maxima, which will be a subset of the knots found in the sample array. A binary search is done on the maxima array with f_{min}. In the sample array, the scalars f_{min} and f_{max} will lie on two segments of the spline, call them seg_{min} and seg_{max}, which are found with two binary searches. The transfer function is evaluated at f_{min} and f_{max} using linear interpolation of the corresponding segment endpoints. Call these temporary knots $k_{f_{min}}$ and $k_{f_{max}}$, respectively. If seg_{min} and seg_{max} are the same segment, then the maximum function value is the greater of $k_{f_{min}}$ and $k_{f_{max}}$. Otherwise, the function maximum is found at knot $k_{f_{min}}$, knot $k_{f_{max}}$, or one of the knots in between these two. A linear search is performed on all knots between $k_{f_{min}}$ and $k_{f_{max}}$ in the maxima array. Unnecessary interpolations can be avoided and the linear search made more efficient as follows. If seg_{min} and seg_{max} are the same segment, then only a single linear interpolation is required dependent upon the slope of the segment. If the slope is positive, then only $k_{f_{max}}$ is computed, otherwise only $k_{f_{min}}$ is computed. If seg_{min} and seg_{max} are different segments, then we can restrict the linear search as follows. If seg_{min} has positive slope then $k_{f_{min}}$ is not computed and the linear search starts from the knot at the right endpoint of this segment. If seg_{max} has negative slope, then $k_{f_{max}}$ is not computed and the linear search ends at the left endpoint of this segment.

7.2.3 Transfer Function Update

The user will change the transfer function on-the-fly during the exploration phase in our visualization pipeline via a GUI. At each change, the user will redefine a domain subinterval of the function. The transfer function table has to reflect this change for future visualizations. Those table entries that represent values which overlap the indicated subinterval need to be identified and re-assigned with the new maximum function values.

The sample table is updated easily by first identifying those entries that overlap the interval index and then reassigning these values to the new transfer function values. The two interval tables are updated incrementally in a bottom-up fashion by propagating the new function values. First, all entries that overlap the interval index and represent intervals of size one are updated with the new function values. Note that these are the leaf nodes in the interval tree table. The entries that represent intervals of size two are then reassigned using the values stored at the entries representing the intervals of size one. At each step the entries that represent the intervals of the next interval length are updated using the entries that represent intervals of smaller length. In the case of the interval table, an entry that represents an interval of size k is assigned the maximum of the values at the two entries with sizes one and $k - 1$, which when concatenated together span the interval of size k. In the case of the interval tree, a value at an internal node is assigned the maximum of the values found at its two child nodes, which contain up-to-date values.

7.3 Maximum Projection Rendering

7.3.1 On-sample Rendering

Let’s assume that the user’s view is coincident with the MM-PRI view sample. The volume is rendered at or near an MM-PRI view sample location using current generation PC graphics hardware. The 2D array of per-pixel scalar pairs that is associated with an MM-PRI is stored in a 2D texture, call it MM_{tex2D}, where each scalar pair occupies two of the color channels. Float textures provide high precision and a texel format that holds exactly two color channels. The transfer function table is stored in one or more 1D textures and function values may be stored with high precision in a float format as well. The functionality of todays GPU allows for the implementation of all of the search procedures on the transfer function tables presented in the previous section. Dependent texturing and multitexturing are used to compute a rendering. A view-aligned quadrilateral is placed in front of the viewer and rendered with the texture MM_{tex2D} and the transfer function texture(s) mapped to it. The corners of the MM_{tex2D} texture are mapped to the quad vertices. A fragment program retrieves the scalar pair (f_{min}, f_{max}) from the MM_{tex2D} texture and performs the transfer function lookup on the transfer function table textures texture using the pair as an interval index. The table sizes are passed to the fragment program as object colors.

A different fragment program is executed to search on each table type. The procedures presented in the last section are implemented in the fragment programs. The sample table representation will consist of two 1D textures, i.e. one for each of the sample array and the maxima array. The size of the tables are passed to the fragment program as primary and secondary object colors. The entries of the interval table are stored in sorted order by increasing interval length and then by increasing minimum scalar value in a single 1D texture. The table index is computed from f_{min}, the length of the interval represented by the interval index, l, and the total number of entries in the table, n, as shown in the right-hand side of equation 11. The values f_{min} and f_{max} are assumed to be normalized, $\hat{f}_{min} = f_{min} \cdot n$, $\hat{f}_{max} = f_{max} \cdot n$, and $l = \hat{f}_{max} + \hat{f}_{min} - 1$. The entries of the interval tree table are stored in breadth-first order in a single 1D texture. The texture has an RGBA format, which can hold the four-tuple of values stored per tree node. The search procedure starts from the root node, stored at the first texel. The left and right children of an internal node at texel i is easily found at $2i + 1$ and $2i + 2$, respectively. The linear spline form of the transfer function table is represented by two 1D texures (sample array and maxima array). Linear interpolations are applied in the sample array when needed.

$$s = \sum_{i=0}^{(l-2)} (n - i) + \hat{f}_{min} = (l-1)\left(n + \frac{1}{2}l - 1\right) + \hat{f}_{min} \tag{10}$$

The transfer function tables must store both color and opacity to compute a MOP projection. The transfer function tables need to be augmented to store three color and one opacity value instead of just one grey level intensity. The sample table will consist of three 1D textures. The first texture holds the sample array with an RGBA format. The second texture holds just the locations of each local maxima in sorted order. The third texture holds color and opacity values of the local maxima at each corresponding location specified in the second texture. The interval table is represented by a single 1D texture with an RGBA format. The interval tree table is represented by two 1D textures. The first texture holds the tree node values *l*, *r*, and *m* values per node. The second texture holds the RGBA values per tree node. The search is performed on the first texture and the color and opacity are retrieved from the second texture. The transfer function with the linear spline form must hold the distance of the knot in addition to color and opacity. Here, we use four 1D textures. The sample array is represented by two 1D textures where the first holds knot locations and the second is a texture of the same size that holds corresponding color and opacity values. The maxima array is represented by two 1D textures which hold knot locations and color and opacity as is with the sample array textures.

There is an ambiguous case in MOP rendering when two or more scalars map to the same maximum opacity and map to different colors. The ambiguity may be resolved by choosing either the color associated with the smaller or the larger scalar value in the interval. We allow the user to make this choice.

7.3.2 Off-sample Rendering

The MM-PRI may be represented by a layered slab construction to allow for approximate off-sample rendering with an image-based rendering approach. The view frustum is partitioned into *k* image-aligned slabs. For each slab, a 2D array of per-pixel scalars is computed. The pixel rays are paritioned at the boundaries of the slabs and each min/max pair in a 2D array represents the range of scalar values within a slab only. Each 2D array is represented as an MM_{tex2D} texture holding min/max scalar values. These are each texture mapped onto one of *k* view-aligned quads, each placed at the center of a slab. The slabs are projected onto a render target and the fragement program is coded to write the maximum value between the classified value assigned to the fragment and the framebuffer pixel value.

CHAPTER 8

LS-PRI

A full description of the function $f(t)$ on the pixel ray is needed to solve the projection equations described in this chapter. A number of function representations are possible, as we know from Numerial Analysis. We chose a linear spline representation for $\tilde{f}$ based on the following criteria: 1) approximation accuracy to the 1D scalar function, 2) efficient evaluation of the transfer function and solution to the projection equation, 3) representation size, and 4) ease of implementation on graphics hardware.

Higher order polynomial and spline functions can more accurately approximate the scalar function with a more compact representation than a linear spline form, but are more difficult to fit the function and to solve the projection equations. A polynomial is completely described by its coefficients and a spline by its knot sequence. An example is to solve volume integration with a higher order function representation. Max [62] and Williams [101] show how to solve the integral equation when both the 1D pixel ray scalar function and the transfer functions have linear spline forms, which requires a search in the transfer function space. Williams [102] shows how to solve the integral when the 1D pixel ray scalar function has a linear spline form and the transfer functions are quadratic with an expensive 5-point Guassian quadrature rule.

The wavelet provides a level of detail representation and should be further explored further. The 0-degree spline function, which is calculated by most volume renderers in its uniform sampled form, requires too many knot samples to accurately approximate the scalar function over the Nyquist frequency. The result is a revoxelization of the volume where at least one sample is needed per voxel. To ensure high image quality, Engel [26] states that the sampling rate should exceed the product of the Nyquist frequencies of the scalar field and the maximum frequency between the transfer functions. The value of a constant spline segment in the 0-degree spline is used to evaluate the transfer function whose value is applied to the entire segment. Thus, the frequency of the transfer function must be manageable. Previous work [62][26] has shown that evaluating the transfer function over each linear spline segment using pre-integrated transfer functions considerably improves image quality. The question of whether the 0-degree spline with uniform sample spacing may be represented more compactly has been addressed before. Consecutive runs of same valued samples may be merged into longer constant pieces to create a 0-degree spline with variable spacing. Though, this does not yield a considerable reduction in the number of samples, as shown by Srivastava [92] in their run-lengh encoding scheme. The reason is that a constant functional piece is a poor approximating primitive. Srivastava [93] approximate consecutive point samples that are nearly colinear with a single line segment to construct a linear spline representation that reduces the knot count for storage purposes. The linear spline is transformed back to a constant spline during volume integration using a Riemann Sum solution.

8.1 Linear Spline Data Structure

A linear spline is defined by a knot list where the function is linear between consecutive pairs of knots. Thus, each linear segment has a pair of knots as its endpoints, $[(t_k, a), (t_{k+1}, b)]$, where t_k and t_{k+1} are the locations of the knots on the pixel ray, and a and b are the respective scalar values. There are two possible ways to store the knot list. One way stores the value pair (t_k, s) for each knot, where t_k is the distance of the knot from the image plane and s is the scalar value at this knot location. Another way is to store the value pair (l, s), where $l = t_{k+1} - t_k$ is the length of the linear segment along the pixel ray and s is the scalar value at the knot location t_k. In order to retrieve a particular knot, the knot list must be traversed from the front and the segment lengths must be summed together.

8.2 Variations To MIP

8.2.1 Simple Depth-shaded MIP

A simple way to apply depth shading to the MIP approach is to weight the maximum intensity by its distance from the image plane on the pixel ray, as shown in equation (12). The depth function $d(t)$ may be an arbitrary function. There is an ambiguous case when there are multiple occurences of the maximum intensity on the pixel ray. The convention used is to set the pixel color to the the maximum intensity associated with the largest weight factor. The solution to this equation requires a search for the maximum intensity and then the application of the weight factor as the final pixel color.

$$color(i, j) = d(t) \cdot \max_{t_{min} \le t \le t_{max}} (T(f(t))) \tag{11}$$

The location of the maximum intensity on the pixel ray is found by a search on the linear spline segments. The local maximum intensity over a spline segment, denoted by its endpoints (t_k, a_k) and (t_{k+1}, a_{k+1}), is found by using the scalar values of the endpoints as indices to a transfer function $T_{max}(s_1, s_2)$. The location, call it t_{max}, of the returned local maximum intensity, call it a_{max}, is easily found by linear interpolation. A problem that the search procedure must handle is the possibility that the maximum intensity may occur at mulitple locations along the same segment and on different segments along the pixel ray. We use a two-pass approach that quickly excludes segments from the search that do not contain the global maximum intensity and thus avoid expensive linear interpolation calculations. The first pass uses the min/max per-pixel scalar value pairs of the MM-PRI to quickly find the maximum intensity along the pixel ray, call it I_{global}. In the second pass, the linear spline segments are traversed in order (the direction does not matter). A running maximum weighted intensity value, call it $I_{max,}$ is maintained and initialized to zero intensity. For each spline segment, the local maximum intensity is retrieved from $T_{max}(s_1, s_2)$ and compared with the global maximum intensity found in the first pass. If it is less than this value, then we move on to the next segment. If the segment contains the global maximum, then its location is found on the segment, t_{max}, and if $d(t_{max}) \cdot I_{global} > I_{max}$ then value I_{max} is replaced with this new weighted intensity. A requirement is that the transfer function $T_{max}(s_1, s_2)$ return a list of scalar values that represent multiple occurences of I_{global} within the same segment. All these values are compared with I_{global} and I_{max} one-by-one in our procedure.

The search over the linear spline segments may be terminated early if the function $d(t)$ is restricted to be a decreasing function. An example is a ramp function with negative slope, which is a commonly used weighting function in practice. The weighted global maximum intensity along the ray is necessarily the one closest to the image plane and so a front-to-back traversal is used on the spline segments. The weighted value of the first occurence of the global maximum is assigned as the pixel value.

8.2.2 Depth-shaded MIP

Heidrich [37] defines depth shaded MIP as first modulating the classified samples by their associated depth weights and then applying the maximum operator on these values. Heidrich uses a ramp depth weighting function with negative slope, whose slope and vertical position can be changed by the user. Shareef [88] uses ramp and exponential depth functions, which are applied in hardware using OpenGL's **glFog** equations. General forms of the depth weighting function can provide meaningful visualizations and can be used as a tool for data exploration. The maximum projection equation for an arbitrary depth function is shown in equation (12). The difficulty in solving this equation is that the maximum operator is applied on the product of two functions. If the functions d or T take general forms, then the maximum weighted intensity must be searched on the pixel ray at point samples chosen according to the frequency characteristics of both functions. We restrict the depth function, $d(t)$, to be a linear spline. In the following two sections, we restrict the form of the transfer function and present efficient search algorithms to solve the projection

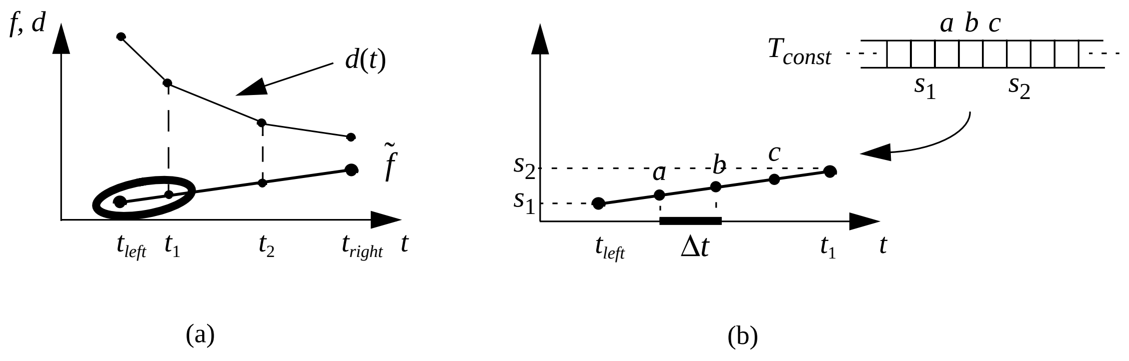


Figure 13. (a) A linear segment of $\tilde{f}$ is partitioned into three intervals, or subsegments, $[(t_{left}, t_1), (t_1, t_2), (t2, t_{right})]$ by the overlapping knots of the depth function, t_1 and t_2. Additional knot locations are found on a subsegment according to the number of transfer function table entries that overlap the subsegment's scalar interval (s_1, s_2).(b) shows the new knot values on the subsegment inside the ellipse shown in (a) given that there are three transfer function table values within the subsegment's scalar range.The product of the depth function and the transfer function are computed at these locations.

equation.

$$color(i, j) = \max_{t_{min} \le t \le t_{max}} (d(t) \cdot T(f(t))) \tag{12}$$

8.2.2.1 0-degree Spline Transfer Function

First, we take the transfer function to be a 0-degree spline function and stored as the table type T_{const}. The projection equation is solved by a traversal on the linear segments of $\tilde{f}$. The maximum weighted intensity is found for each segment and the maximum of these values is assigned as the pixel value. To find the maximum weighted intensity at any linear segment of $\tilde{f}$, the segment is subdivided into subsegments according to the knots from function $d(t)$ that overlap the segment. Figure 13a illustrates this subdivision on a linear segment in $\tilde{f}$ with endpoint locations (t_{left}, t_{right}). The portion of the depth function that overlaps this pixel ray interval contains two overlapping knots located at t_1 and t_2 to define three subsegments. The product of the depth function and the transfer function is computed on each subsegment by locating transfer function samples that overlap the subsegment and then evaluating both functions at these sample locations. If n is the number of entries in the transfer function table that overlap a subsegment's scalar bounds (s_1, s_2), then new knots are located at a sample spacing of $\Delta t = (t_{j+1} - t_j)/(n+1)$ starting from the subsegment's left endpoint, where t_j and t_{j+1} are the subsegment's left and right endpoints, respectively. In the following pseudocode, the linear spline segments are traversed and a special maximum operator is called for each subsegment:

```
maxCol= 0;

For $k = 0$ to numsegs
{
    $m_f = (\tilde{f}(t_{k+1}) - \tilde{f}(t_k))/(t_{k+1} - t_k)$;
    Partition the segment $(t_k, t_{k+1})$ into $m$ subsegments with end-
        points at $(t_{k_0} = t_k), t_{k_1}, \ldots, t_{k_{m-1}}$;

    For $p = 0$ to $(m-1)$
    {
        $s_1 = \tilde{f}(t_{k_p})$;
        $d_1 = d(t_{k_p})$;
        $s_2 = \tilde{f}(t_{k_p})$;
        $d_2 = d(t_{k_p})$;
        $length = t_{k_{p+1}} - t_{k_p}$;

        $m_d = (d(t_{k_{p+1}}) - d(t_{k_p}))/(length)$;

        $n$ = number of entries in table $T$ between $T(\tilde{f}(t_{k_p}))$ and
                $T(\tilde{f}(t_{k_{p+1}}))$;
        $\Delta t = (length)/(n-1)$;

        $\Delta s = m_f \cdot \Delta t$;
        $\Delta d = m_d \cdot \Delta t$;

        $col =$ $getmax(s_1, d_1, s_2, d_2, \Delta s, \Delta d, d, T, n)$;
        **if** $(col > max\mathrm{Col})$
            $maxCol = col$;
    }
}
$color(i, j) = maxCol$;
```

The function *getmax* returns the maximum intensity found on a subsegment. The spacing of the new knots, Δt, is computed from the slope of the subsegment and *n*, the number of overlapping table entries. The locations of the new knots are computed incrementally starting from an endpoint and the product of the functions *d* and *T* is computed at the endpoints and these new knots. The maximum of these values is the maximum intensity over the subsegment. The pseudocode for the algorithm follows:

```
        $getmax(s_l, d_l, s_r, d_r, \Delta s, \Delta d, d, T, n)$
        {
                $maxProd = 0$;
                $maxIntens = 0$;

                if ( $\Delta d < 0$ )
                {
                    s = $s_l$;
                    d = $d_l$;
                    **For** $i = 1$ **to** $n$
                    {
                        $I = T(s)$;
                        **if** $(I > maxIntens)$
                        {
                            $maxIntens = I$;
                            $col = d \cdot I$;
                            **if** $(col > maxProd)$
                                $maxProd = col$;
                        }
                        $d = d + \Delta d$;
                        $s = s + \Delta s$;
                    }
                }
                else
                {
                    s = $s_r$;
                    d = $d_r$;
                    **For** $i$ = n **to** 1
                    {
                        $I = T(s)$;
                        **if** ($I > maxIntens$)
                        {
                            $maxIntens = I$;
                            $col = d \cdot I$;
                            **if** ($col > maxProd$)
                                $maxProd = col$;
                        }
                        $d = d + \Delta d$;
                        $s = s + \Delta s$;
                    }
                }

                **return** $I$;
        }
```

The function will avoid extraneous product computations using the slope of the linear segment from the depth function. If the slope is negative, then products with intensities less than the current maximum intensity, *maxIntens*, are avoided. If the slope is positive, then the subsegment knots are traversed in the opposite order.

8.2.2.2 1-degree Spline Transfer Function

Now we take function T to be a linear spline and stored in the table type T_{lin}. We use a similar approach to compute the projection equation in that the knots of the transfer function that overlap a linear segment of $\tilde{f}$ are also used to partition the segment into subsegments in addition to the overlapping knots from the depth function. The scalar values at the endpoints of the linear segment define a domain subinterval in the transfer function

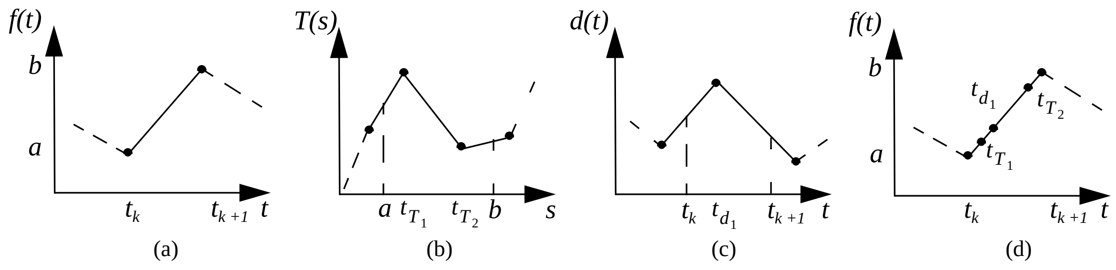


Figure 14. A linear spline segment (a) is partitioned by overlapping knots from the transfer function (b) and depth function (c). The resulting knot list, $(t_{T_1}, t_{d_1}, t_{T_2})$, partitions the segment into subsegments (d). The maximum intensity of the entire segment is maximum intenstity of the maxima computed for each subsegment.

space and the knots that lie within this interval are overlapping. The locations of these knots on the pixel ray are easily found on the linear segment by linear interpolation. Figure 14 shows an example of a partition on a linear segment of $\tilde{f}$ (Figure 14a) using knots from both the transfer and depth functions. If the number of overlapping knots in the transfer function in the scalar range (a, b) is two (Figure 14b) and the number of overlapping knots in the depth function along the pixel ray in (t_k, t_{k+1}) is one (Figure 14c), then the linear segment is partitioned into four subsegments (Figure 14d). Along the extent of each subsegment, the functions d and T are linear. If the line segments cross each other, then a new knot located at the intersection point is inserted into the knot list to further partition the linear segment.

To compute the maximum product on a subsegment of $\tilde{f}$, we consider the four possible ways that the line segments from functions T and d may be oriented with respect to each other. The first two cases are when the slope of both line segments have the same sign. The maximum product over the subsegment is trivial to compute here because it is simply the product of the function values at the two endpoints. If the lines are downward sloping, then the product is computed at the left endpoint (Figure 15a). Otherwise, it is computed at the right endpoint (Figuire 15b). When the sign of the slopes is different, then the maximum product may lie anywhere along the subsegment. We construct a quadratic function, which is the product of the two line segments, and solve for the maximum of this function. Let the transfer function evaluated at the left and right endpoints be T_1 and T_2 and those of the depth function be d_1 and d_2, respectively. We describe the line segments of both the transfer function and the depth function on the subsegment within the same parametric and normalized space, $t \in [0, 1]$. The product of the two functions is the quadratic equation $D(t) = at^2 + bt + c$, where $a = (T_2 - T_1)(d_2 - d_1)$, $b = (T_2 - T_1)d_1 + (d_2 - d_1)T_1$, and $c = d_1 T_1$, and is a parabola graphically. The maximum of the function on the subsegment can be calculated either at the line endpoints or at the vertex of the parabola, which is computed as $v = (-b)/(2a)$. We use the following conditions to determine how to compute the function maximum. If the parabola is open downward, i.e. $a < 0$, then the function maximum may be computed at one of three locations, as shown in Figure 16a. If $v < 0$, then the maximum is computed at the left endpoints. If $v > 1$, then it is computed at the right endpoints. Otherwise, the maximum lies at the vertex, i.e. $D(v)$. If the parabola is open upward, i.e. $a \geq 0$, then the maximum is computed at the left or right endpoints. If $v < 1/2$ then it is computed at the right endpoints, otherwise it is computed at the left endpoints (Figure 16b).

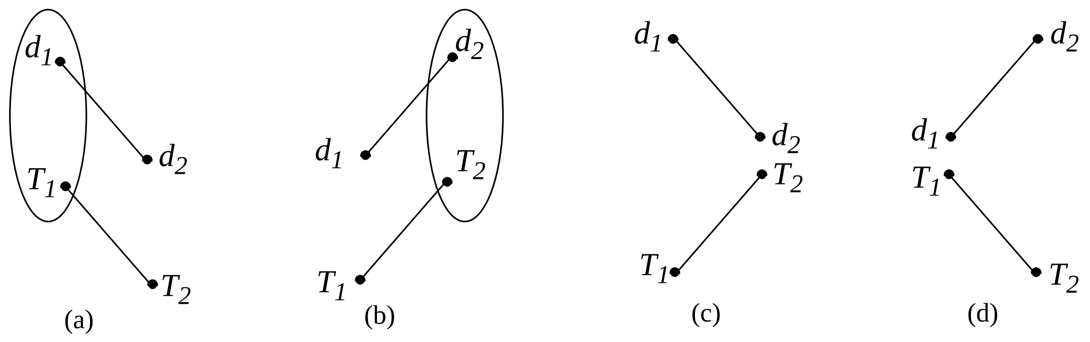


Figure 15. There are four possible line segment orientations from the transfer and depth functions on a subsegment of $\tilde{f}$. The maximum of the product is computed at the line endpoints for the two trivial cases, as illustrated by the red ellipses in (a) and (b). When the line slopes are different, as shown in (c) and (d), the maximum product is computed with a quadratic function.

8.2.3 Local MIP

The local MIP projects the first local maximum intensity above a user-defined threshold. If no such local maximum exists, then the global maximum intensity on the pixel ray is assigned as the pixel color. This projection equation can be computed with our per-pixel linear spline data structure with a front-to-back traversal of the linear segments of $\tilde{f}$ and a search in the transfer function space for each segment. The search is terminated early when the first local maximum intensity above the threshold is found. Our approach has two phases. First, the global maximum intensity is computed for each pixel using the MM-PRI and the transfer function table type T_{max}, presented in the previous chapter. Next, a local maximum intensity is searched for in the transfer function space for each linear segment on a subinterval of the domain of the transfer function, which is defined by the scalar values at the endpoints of a linear segment. Take the scalar pair (s_l, s_r) to be the scalar values at the left and right segment endpoints. If the $s_l < s_r$, then a local maximum is searched for on the domain subinterval $[s_l, s_r]$ in the direction from s_l to s_r. Otherwise, the local maximum is searched on this interval from s_r to s_l. This ensures that the local maximum closest to the image plane is found. If a local maximum intensity is found, then the front-to-back traversal is terminated and this value replaces the global maximum value assigned to the pixel in the first phase.

8.2.4 RLS-PRI

The LS-PRI construction can be reduced in size when solving the local MIP equation or when solving the depth shaded equations for the case that the depth function is assumed to be decreasing. The scalar values on intervals along a pixel ray that represent redundant scalar values encountered earlier in a front-to-back traversal of the scalar function (the shaded portions shown in Figure 12a) will not contribute to the solution of the projection equations in these two cases. Thus, they can be removed from the LS-PRI construction to define a reduced version called the RLS-PRI. The per-pixel linear segments contained in the RLS-PRI will be a subset of those found in the LS-PRI.

A linear spline at a pixel in the RLS-PRI is constructed from the linear spline found at the corresponding pixel in the LS-PRI. The segments are traversed in front-to-back order and the current range of scalar values encountered so far is updated when each segment is processed. For each segment, if the scalar range denoted by the scalar values at its end-points is does not overlap the current scalar range, then this segment is inserted into the RLS-PRI. If the segment's scalar range is completely contained within the bounds of the

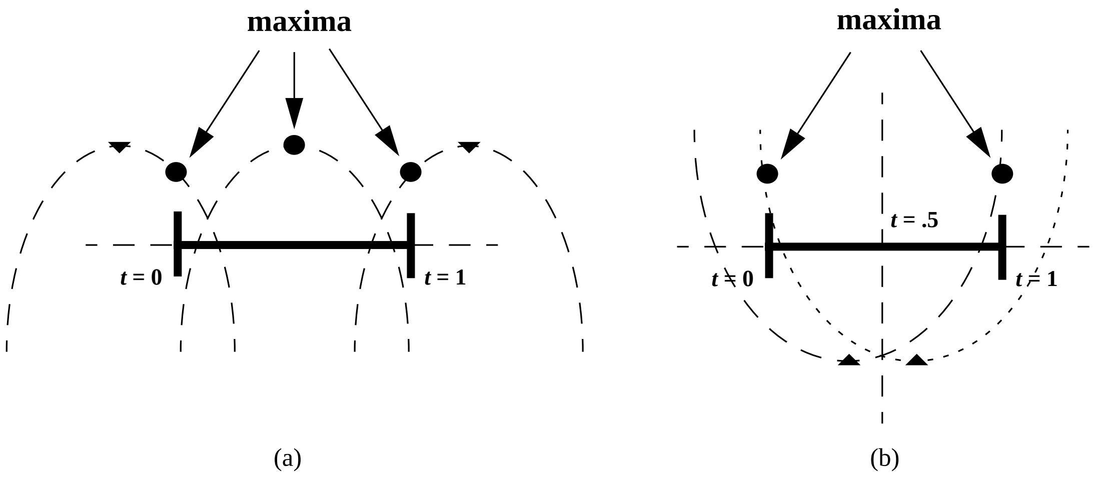


Figure 16. The maximum product for the two non-trivial cases of Figure 15 are computed by solving a quadratic parametric equation. The direction of the openness of the parabola and the location of its vertex with respect to the normalized interval of the subsegment determines where to compute the maximum. If the parabola is open downward (a), then the maximum is computed at the endpoints of the line segments or at the vertex of the parabola. Otherwise, the maximum is computed at the endpoints given the location of the vertex with respect to the midpoint of the interval.

current scalar range, then the segment is not included into the RLS-PRI. If the segment's scalar range partially overlaps the current scalar range, then the segment is clipped, the portion outside the clipped range is added to the RLS-PRI, and the current scalar range is updated to incorporate this portion. The segments inserted into the RLS-PRI at a pixel are collected into runs of contiguous segments. Thus the final construction contains a sorted list of segment runs, where each run is disjoint from the other contiguous segments.

8.3 Volume Integration

Volume rendering by integration is a powerful visualization approach, where Sabella's low-albedo density emitter model [79] is widely used in practice. The absorption plus emission form of the model leads to high quality renderings, but the simpler absorption only and emission only models [61] also lead to useful renderings.

8.3.1 The Integral Equations

The integral equations for the density emmitter model were presented in Chapter 2.2, where the transfer functions were the glow term $c_{\lambda}(s)$ and the extinction coefficient $\rho_{\lambda}(s)$.The integral equations are solved by computing a partial integration on each linear segment of $\tilde{f}$. A segment is described by the scalar values at its endpoints, s_f and s_b, and its length, l, on the pixel ray. The three-tuple (s_f, s_b, l) is input to the integral equation to compute a color and possibly an opacity for the segment. The final integration combines all the per-segment values from the linear spline.

The emission only, absorption only, and simplified forms of the absorption plus emission model [62][26] require an integration of the appropriate transfer function over a linear segment. Equation 14 shows the generalized equation we need to solve and a solution that only depends upon the parameters of the linear segment. Let the function $g(u) = (|s_b - s_f| / l) \times u + s_f$ describe the scalar function defined on a single segment of the linear spline, where $0 \le u \le l$. The last formula in equation 13 shows that the summation can be computed in classification space using the scalar bounds as indices on a transfer function σ. This equation is easily solved if the transfer function is sampled with uniform spacing with a pre-integrated transfer function [62][26] as long as the transfer function σ is non-negative and integrable, i.e. $\int_{s_f}^{s_b} \sigma(v) dv = |\sigma(s_b) - \sigma(s_f)|$. If the transfer function has form T_{lin}, then the summation is solved analytically using the trapezoid rule. First, the linear segments that the scalar s_f and s_b overlap are found with a binary search on the linear segments of the transfer function. The areas under the linear segments that lie between these two values on the function domain are summed together.

$$\int_0^l \sigma(g(u))du = \int_0^l \sigma((|s_b - s_f|/l) \times u + s_f)du = \int_{(|s_b - s_f|/l) \times 0 + s_f}^{(|s_b - s_f|/l) \times l + s_f} \sigma(v)dv$$

$$= (l/|s_b - s_f|)\int_{s_f}^{s_b} \sigma(v)dv = w_{s_f, s_b, l}\int_{s_f}^{s_b} \sigma(v)dv \qquad (13)$$

8.3.1.1 Emission Only

The emission only model sums the glow of emitting particles along the ray to the eye. If p is the total number of segments in the spline $\tilde{f}$ and I_0 is the background color intensity, this equation computes the sum of the glow term for all the spline segments. The variable λ represent wavelength and variable q is an index for each spline segment.

$$I_\lambda = I_{\lambda, 0} + \sum_{q=1}^{p} w_{s_{f,q}, s_{b,q}, l_q} \cdot c_\lambda(s_{f,q}, s_{b,q}) \qquad (14)$$

8.3.1.2 Absorption Only

The absorption only model modulates the background intensity by the sum of the extinction coefficient on the ray applied to an exponential function.

$$I_\lambda = I_{\lambda, 0} \cdot e^{\left(-\sum_{q=1}^{p} w_{s_{f,q}, s_{b,q}, l_q} \cdot \rho_\lambda(s_{f,q}, s_{b,q})\right)} \qquad (15)$$

The summation is computed first and then the exponential is applied to it.

```
$sum$ = 0;
$q$ = 1;

**For** ($q < p$) **do**
    $sum$ += $w_{s_{f,q}, s_{b,q}, l_q} * \rho_\lambda(s_{f,q}, s_{b,q})$;
    $q$++;

$color = I_0 * exp(-sum)$;
```

8.3.1.3 Absorption Plus Emission

The absorption plus emission equation models not only the glow of the material but also the attenuation of the glow as it passes through material on the way towards the eye. A number of methods have been presented to solve this equation when the scalar function on the pixel ray is a linear spline. The solutions presented in the literature stem from approaches addressing the problem of volume rendering of unstructured grids. A main difficulty encountered is how to compute self-attenuation within a linear segment. These solutions make one or more simplifications to the absorption plus emission equation in order to compute an approximate solution. Shirley [89], using graphics hardware available over a decade ago that had only limited texturing capabilities, take the average of the transfer functions evaluated at both ends of a linear segment at the thick vertex of a tetrahedral volume cell as the color and opacity of the segment. These values are interpolated across the triangle in a Gouraud shading style interpolation. Stein et al [94] improve on this with dependent texturing by computing color and alpha more accurately across the tetra on a per-pixel basis. Max [62] simplifies the model by restricting the glow term to be constant (or piecewise constant) and allows the extinction coefficient to be any integrable function, thereby preserving self-attenuation. Engel [26] allows both the emission and extinction functions to be arbitrary, but neglects self-attenuation within the segment. Color-alpha compositing [74] is used to combine the color and alpha values computed at each segment. Recent hardware accelerated approaches have tried to address the self-attenuation calculation problem. Guthe [34] store integrations with self-attenuation over all possible segments, (s_f, s_b, l) in a 3D texture table. Though, table size and re-computation of the table during transfer function changes are two problems to overcome. Guthe [35] use a 2D table indexed by a pair of scalar values that holds pre-integrated values for a representative, e.g. average, segment length. To compute a segment's color and alpha, a retrieved pre-integrated value from the table is scaled to the segment's actual length.

The integral equation may be solved with a linear spline form using the Riemann Sum, which is most commonly used in most volume rendering approaches. Srivastava [93] compute evenly spaced point samples on the spline segments to compute a Riemann Sum over their per-pixel linear splines.

8.4 Mapping Linear Splines to Textures

The projection equations may be computed fast using today's graphics hardware equipped with programmable GPU and dependent texturing technology. If the viewer is coincident with an LS-PRI view sample, then a rendering is computed by projecting the per-pixel splines to the viewer's image plane. The spline knots are stored as texture data and accessed by a fragment shader program, which solves the projection equations. An image-aligned quad that spans the area of the image plane area exactly is rendered to the screen. Texture coordinates are assigned so that they may be used as indices into the *x* and *y* directions of the LS-PRI image plane by the fragment shader. We discuss the problem of how to map the per-pixel splines of the LS-PRI into textures. A straightforward approach is to copy the knots into a 3D texture map where two of the dimensions coincide with the pixel array of the LS-PRI and per-pixel splineknots are stored along the third dimension. Since the knot counts may be different between pixels in the LS-PRI, many texels in the 3D texture may contain no data.

8.4.1 Tiled 2D Texture Stack

The knots may be stored in a 2D texture stack. Each 2D texture has a pixel resolution that matches the LS-PRI. The spline knots are assigned to the textures as follows. For each per-pixel spline, each knot is assigned a number starting from one and increased by one according to its front-to-back order. If the spline with most number of knots has *m* knots, then the texture stack will have *m* 2D textures. Each 2D texure is assigned a unique number between 1 to *k* according to its order in the stack. All knots labeled number one are placed into the first 2D texture in the stack at their corresponding pixel locations. All knots labeled two are placed into the second 2D texture at their corresponding pixel locations. This continues for all knots to the label *k*. Many of the texels in the 2D texture stack that were assigned with no knot data can be removed from the representation. We partition each of the 2D textures into a tile grid of *n* x *m* tiles. The tiles that contain no knot data at all are removed from the stack. The tiles that overlap the same region on the 2D texture are collected into a tile run for a total of *n* x *m* runs. The order of the tiles in the 2D texture stack is preserved in the tile run. Since each spline must contain at least two knots, each tile run will contain at least two tiles. The tile runs are stored into a 2D texture in consecutive order by image order and then tile order. A second 2D texture with the same dimensions as the pixel resolution of the LS-PRI stores indices into this texture. Each texel in the second 2D texture indexes in to the corresponding texel that holds the first knot in the run of knots in that tile run. In addition, each texel also holds the number of knots in the run for this spline. An image aligned quad is rendered as before mapped with this second texture. The fragment program retrieves the index and traverses the corresponding knot list in the tile run starting from the value referred to by the index using dependent texturing for the specified number of knots in the list. The fragment program computes the projection equation and outputs the final pixel color.

8.4.2 2D Texture Map

The tiled 2D textured stack will require storing texels that hold no data. The per-pixel splines may be stored compactly into a 2D texture where the splines are placed one after the other in pixel order. A secone 2D texture holds texture coordinates index to the location of the first knot of a corresponding per-pixel spline in the 2D texture holding the splines. In addition, it holds the number of knots in the spline. The fragment program will traverse the spline from this starting point to compute the projection equation for the pixel.

8.5 Linear Spline Simplification

The total size of the LS-PRI is the total number of knots used to represent all the per-pixel linear splines. In addition, there are two values associated with each knot, i.e. its distance from the image plane and a scalar value. The volume data sizes we are considering will require the LS-PRI to contain at least 1 million pixels at a minimum pixel resolution of 1K x 1K. If the knots for a pixel's linear spline are chosen as in volume raycasting, where samples along the pixel ray are located with a uniform spacing, then the minimum number of knots will be 1K for lower resolution data in order to satisfy the Nyquist limit condition since more than one sample should be placed inside each volume cell. Since each per-pixel spline will have an equal number of knots, the memory footprint of the LS-PRI will be at least 1GB. This is essentially a revoxelization of the volume. If we assume that the scalar function is reconstructed linearly, then knots may be located at the intersections between a pixel ray and the volume faces. Each spline segment exactly represents the scalar function inside the volume cells. If volume reconstruction is nonlinear, then more samples need to be taken within a volume cell. In either case, the number of knots per spline will be on the order of the number of volume cells along a direction or at a minimum over 1K cells. The size of the LS-PRI should be reduced in terms of the number of knots to reduce the memory footprint of the data structure. In addition, the speed of the rendering algorithms is proportional to the number of spline segments. Thus, we seek to reduce the knot count of the splines in a lossy manner but with control over the inevitable error that will be introduced in the approximation.

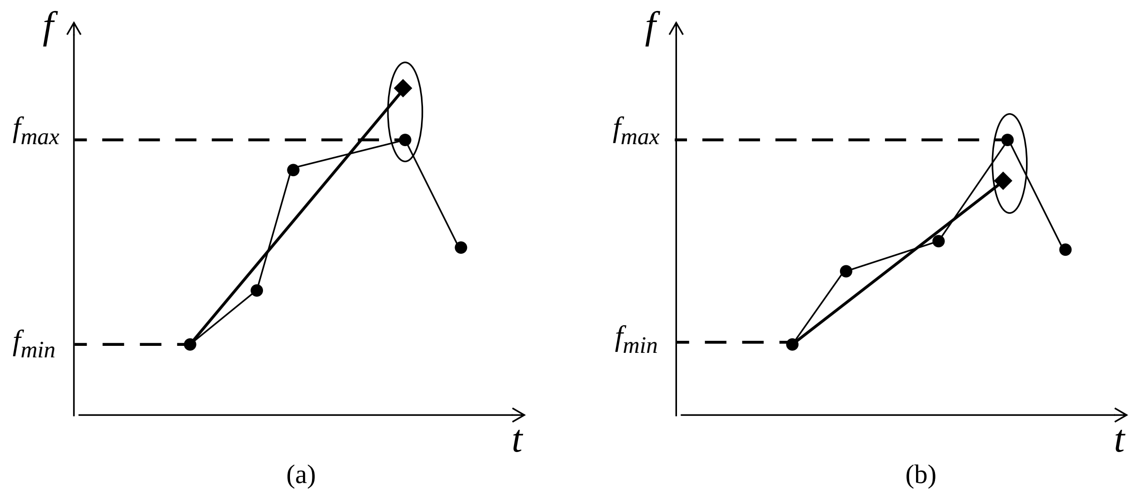


Figure 17. Two problems can occur if a spline simplification algorithm does not ensure that the endpoints of a new linear spline segment coincides with knot locations from the original spline. An "overshoot" case (a) may occur if an endpoint surpasses the scalar bounds, $[f_{min}, f_{max}]$, of the original linear segments it approximates. An "undershoot" case (b) may occur if an endpoint is contained inside the scalar bounds.

We assume that the volume has been resampled along pixel rays with a high enough sampling rate to construct an initial set of per-pixel linear splines for the LS-PRI. Our goal is to reduce the number of knots in the representation to obtain a good approximation to the original per-pixel splines. This is done independently for each per-pixel spline. A simple method to reduce the knot count is to replace consecutive runs of knots that are colinear with only two knots. Even though no approximation error is introduced in the resulting spline, the resultant knot count is a small reduction when applied to real data. In order to obtain substantial reductions, a spline simplification method must replace near linear runs of spline segments into a single linear segment and incur an approximation error, i.e. lossy compression. The goal of any such method is to try to preserve the shape of the function.

This problem has been addressed for quite some time. The methods presented over the years can be classified into knot removal, simulated annealing, and greedy algorithms, which can output different resulting linear splines.

Lokovich [54] uses a greedy approach to reduce linear splines that represent visiblity information in a shadow map. The approach fits new linear segments to consecutive runs of knots such that every knot in the run is within a specified maximum vertical distance from the new line segment. The algorithm initially assigns the first knot in the original linear spline as the left endpoint of the first new line segment. The right endpoint of this new segment is calculated by visiting the original spline knots in order until a knot is encountered that violates the error threshold criterion. The right endpoint is placed with a vertical offset from the last knot in the traversal that satisfied the criterion. This new knot is set as the left endpoint of the second new linear segment and the same traversal is used on the knots after this one to calculate this segment's right endpoint. New line segments are computed consecutively in this manner until all the knots or the the original spline have been visited. Even though this algorithm provides a good curve fit, the resulting linear spline may suffer from improper indexing into the transfer function [93]. The problem is illustrated in Figure 17 and may occur in two instances. The first instance, we call an *overshoot* (Figure 17a), is where a new linear segment extends above or below the scalar bounds of a run of consecutive linear segments it approximates. This will result in adding extra portions of the transfer function when calculating a projection equation due to incorrect transfer function indexing. As Srivastava [93] states, the error may become worse with higher frequencies in the transfer function. The second instance, we call an *undershoot* (Figure

17b), is where the new linear segment does not extend to the scalar bounds of the linear segments it approximates. In this case, portions of the transfer function that should contribute to the projection equation will not. To avoid this occurence, a simplification algorithm should ensure that the endpoints of the new linear segments coincide with original knot locations. Srivistava[93] presents an algorithm that operates the same as [54] and satisfies the same error criterion, except that it ensures this desirable property on the location of endpoints for new line segments.

A drawback of these greedy approaches is that the decisions made when locating a new linear segment is based upon a limited look-ahead. This may lead to increased flattening where the function curves more. This introduced a second source into the projection equation, besides incorrect transfer function indexing, where the slope of the new line segment deviates more from the slopes of the original line segments at a turn in the original function. We use an approach that builds the new linear spline incrementally, but considers the entire spline at each step. It is a modified version of the mesh simplification algorithm for terrain data presented by Duchaineau [23] in its 1D form. The simplified spline is constructed in a bottom up fashion. At each iteration, consecutive pairs of linear segments are possibly merged into single segments. The pairs are nonoverlapping with other pairs, i.e. if the segments are numbered in order, say 0 to n, then a segment pair is denoted by the segments numbered as i and $i + 1$, where i is even. At iteration 0, the algorithm starts with the original linear spline segments and tries to merge segment pairs in order starting from the first pair. If n is odd, then the last segment is not considered in the current iteration, though it may be considered in the next iteration. After all the segment pairs have been traversed and possibly merged into single segments, the next iteration starts from the beginning of this new spline and repeats the process. To merge a segment pair into a single segment, we apply the same error criterion used by [54][93]. A segment pair is defined by three consecutive knots, left, middle, and right. If it is merged, then the new segment will have its endpoints at the left and right knots of the segment pair. A merge is done only if all knots from the original spline that lie between the left and right knots are within a specified vertical distance threshold to the new candidate segment. This guarantees that the final simplified spline will satisfy the error criterion. The algorithm stops when no segment pairs are merged in an iteration. The algorithm is easy to implement and takes O(*nlogn*) time.

We show in Figure 18 six iterations (top to bottom) of our algorithm on a trigonometric function sampled at 128 equally spaced knots. The scalar range is normalized to [0, 1] and the error threshold used is the vertical distance 0.2. The number of segments in successive iterations are: 127, 63, 32, 16, 13, and 7. The parallelpipeds surrounding a segment in the figure represents the vertical distance of the knot from the original spline that maximally deviates from the new segment it overlaps. Any of these intermediate splines may be used for our purposes and so we typically would not choose the spline in the final iteration. This decision could be based upon a second error criterion that chooses the new linear spline in an iteration that has a tolerable number of segments and whose segments have slopes closer to the slopes of the original line segments. Figure 19 shows our algorithm applied to a spline sampled from the the Richtmeyer-Meshkov dataset at time step 134. The original spline has 2048 equally spaced knots, which is shown at the rightmost graph. The graphs shown from top to bottom are the simplified splines in the final iterations when changing the error thresholds to be 0.01, 0.05, 0.1, 0.15, and 0.2, respectively. The table at the bottom right displays the number of knots in the simplified spline and the number of iterations for the algorithm to complete. The knot count falls below one thousand for a tight threshold of 0.01 and much of the detail is preserved in the range of 200 ~ 400 knots.

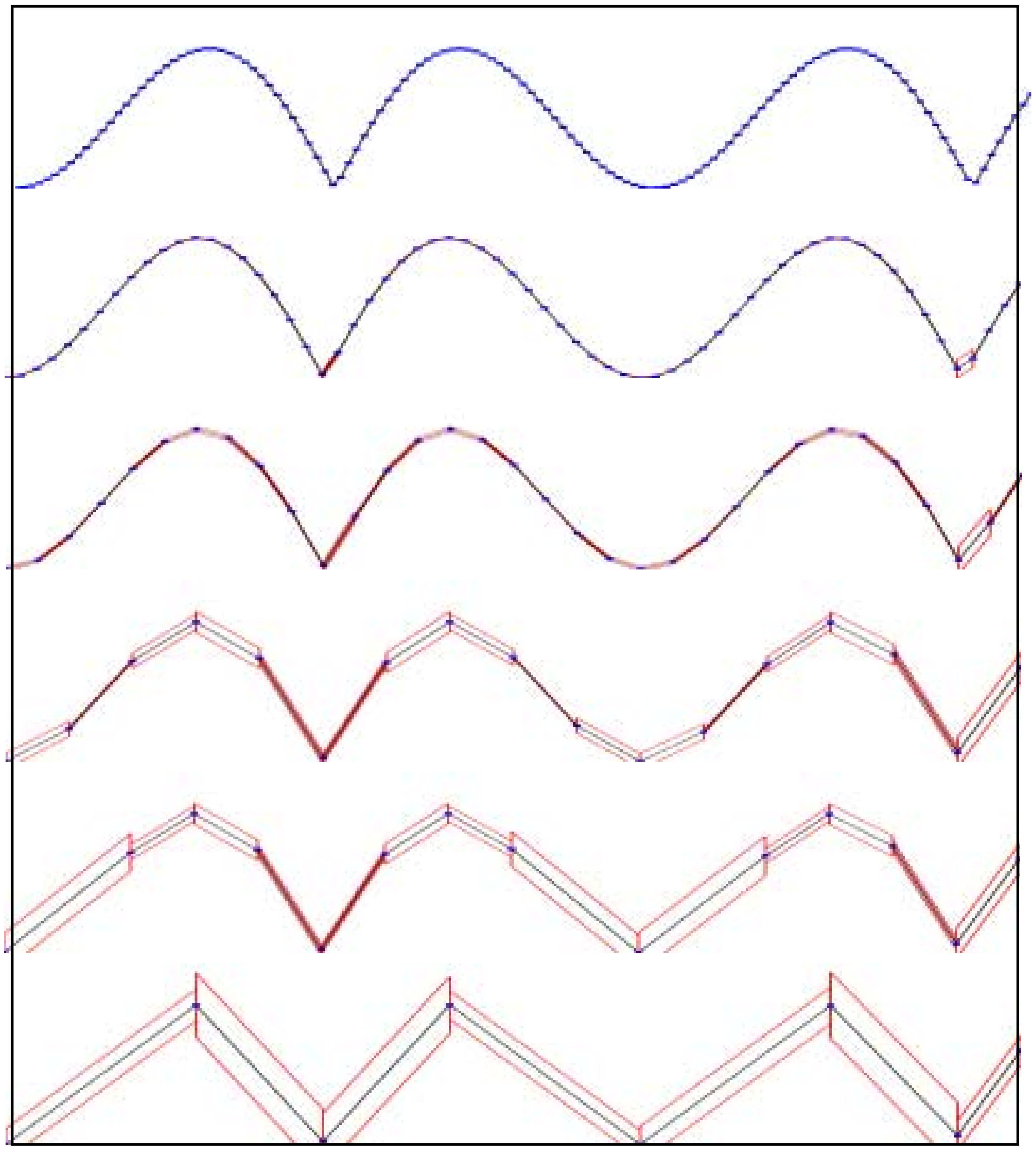
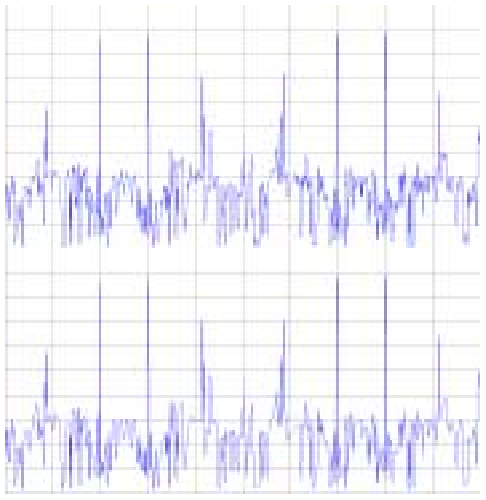
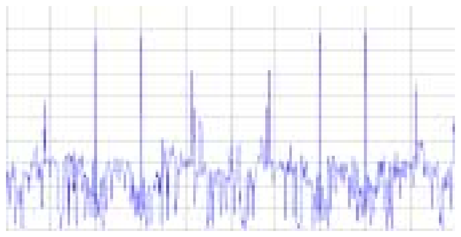
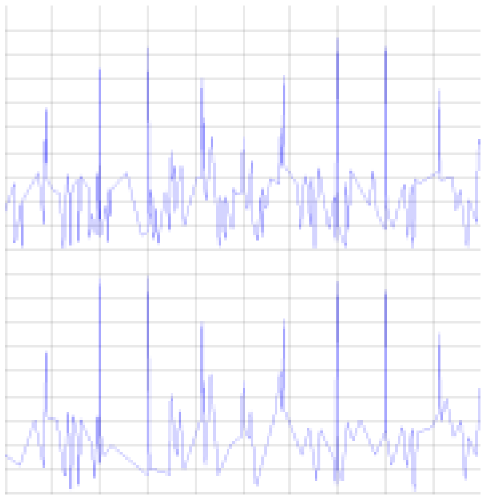


Figure 18. Five iterations of our linear spline simplification algorithm on a trigonometric function sampled at constant spacing and the error threshold set to .2.







Error Tolerance	Knots	Iterations
.01	890	6
.05	368	6
.1	205	9
.15	137	9
.2	86	9

Figure 19. Results of our linear spline simplification algorithm on a pixel ray through real data.

CHAPTER 9

Volume Rendering From Compressed Data

We present a novel volume rendering approach that uses only three LS-PRI. Three such LS-PRI provide both a sufficient sampling and compression of the volume (structured or unstructured). This construction is similar to the Layered Depth Image (LDI) [86], which is used to hold ray-object intersections in a polygonal scene of opaque surfaces. Lischinski [52] samples a polygonal scene with three LDIs positioned to point in orthogonal directions to each other along the three coordinate axes, called a Layered Depth Cube (LDC). In this way, all surfaces in the scene may be sampled regardless of their orientation. We construct a similar scheme and place three LS-PRI's in orthogonal directions to sample the volume. Our rendering algorithm reconstructs the volume from a single LS-PRI, which is determined by the angle between the user's view angle and the one of the three sampling directions. Our approach is similar to the shear-warp [45] approach and Rezk-Salama's [76] approach, which uses only 2D textures and the dependent texturing facility to reconstruct the volume on intermediate data slices.

Our graphics pipeline consists of four steps, as shown in Figure 20. The first three steps convert the volume into the compressed format, which is encoded into three 2D texture pairs. The fourth step renders the volume directly from this compressed format and requires no decompression. In the compression phase, we resample the volume along the sampling rays in the three LS-PRIs. If the volume is assumed to be reconstructed linearly within volume cells, then spline knots are located at the intersections between the sampling ray and a cell face. Otherwise, samples are chosen using a uniform sample distance. In the second step, the per-pixel splines of each of the three LS-PRI are compressed using our linear spline simplification approach presented in Chapter 8.5. The third step creates a pair of 2D textures for each LS-PRI, as presented in Chapter 8.4.2. The rendering phase (step four) is a two-pass algorithm that reconstructs scalar slices of the volume in the first pass and then classifies, projects, and blends the slice to the screen in the second pass. The intersection of the bounds of each the image planes from the LS-PRI's defines the smallest bounding box that encloses the volume. Data slices that will hold sheets of reconstructed scalar values will be defined within the bounds of this bounding box. The slices are projected in depth order, i.e. front-to-back or back-to-front.

9.1 Per-Slice Two-pass Rendering

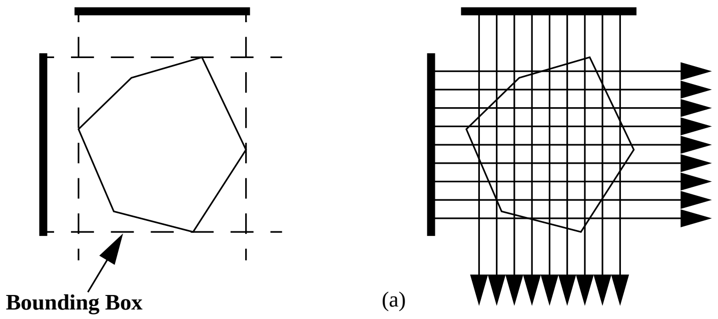
After the compression phase, the volume is rendered from one of the three compressed LS-PRIs. The choice is determined by the smallest angle between the user's view direction and each of the sampling directions of the LS-PRIs. The volume is reconstructed on data slices that are oriented to be no more than 45 degrees from the sampling direction of the chosen LS-PRI. Our algorithm supports both axis- and view-aligned slicing, as described in the following sections. Each data slice is rendered in two-passes. In the first pass, the proxy geometry representing the slice is rendered orthogonally to the LS-PRI's image plane. This projection is captured with the render-to-texture facility to a float pbuffer. The proxy geometry is multitextured with the 2D texture pair. A fragment shader program uses the index retrieved from the index texture and the proxy's distance from the image plane to search for the linear segment in the linear spline texture and perform linear interpolation. This search is performed efficiently using binary search. In the second pass, the float buffer is mapped to the proxy geometry, which is now projected and blended into the framebuffer, where classification is performed in a second fragment shader program.

9.2 Axis-aligned Slice Planes

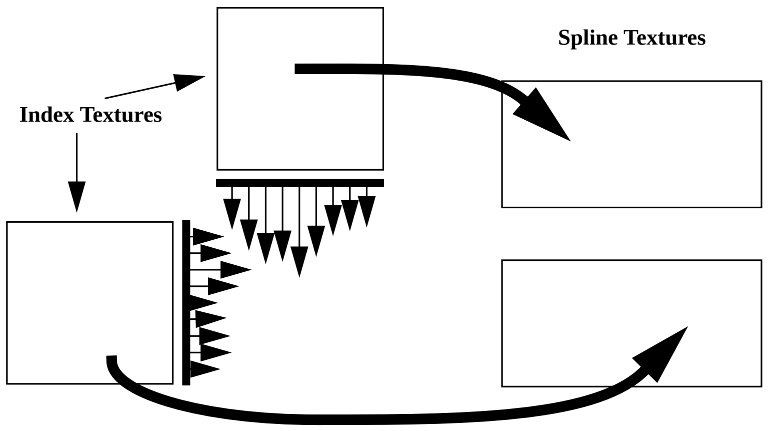
In axis-aligned slicing, data slices are uniformly spaced quads that are oriented to be perpendicular to the sampling direction of the LS-PRI. This is commonly used for 2D texture volume rendering. Rekz-Salama [76] computes trilinearly interpolated scalar values from consecutive slices of a stack of 2D textures constructed from a rectilinear grid. Similarily, we are able to do the same with our linear spline representation. Since each data slice is located at some distance from this plane, this distance may be input to the fragment program through the object color. Each quad is the same size as the LS-PRI's image plane. The index texture is mapped to fill the quad exactly so that the fragment shader program can access each per-pixel linear spline correctly. The resulting scalar sheet is mapped to the quad in the second pass and rendered with blending to the framebuffer.

9.3 View-aligned Slice Planes

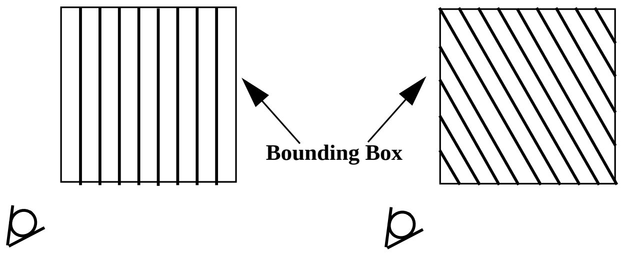
View-aligned slices may be rendered using our approach without the need for a 3D texture. This alignment avoids popping artifacts [45] at view boundaries where the axis-aligned slicing changes abruptly. View-aligned proxy geometry is defined as in 3D texture volume rendering and clipped to the bounding box. A proxy may have a minimum of three sides and maximum of 5 sides, if we ignore the degenerate cases when the data slice intersects only at a bounding box vertex or edge. Scalar values are reconstructed across the data slice as in axis-aligned slicing. The exception is that since the proxy geometry may not be orthogonal with the sampling direction of the LS-PRI, the distance to each per-pixel location on the proxy must be interpolated across the polygon. The fragment program will use this interpolated distance value to perform a search and interpolation in each linear spline. After the first pass, the pbuffer will contain the reconstructed scalar values in a portion of the buffer. In the second pass, the pbuffer is texture mapped to the proxy and the texture coordinates are easily computed from location vertices of the proxy on the boundinx box edges using a simple planar projection. This ensures that only the portion of the pbuffer that was assigned scalar values will texture the proxy.



(a)



(b)



(c)

Figure 20. Our rendering pipeline for rendering from compressed data. The compression step is shown in (a) and (b). The volume is sampled and in each LS-PRI (a). The splines are compressed and mapped to 2D texture pairs (b). In the rendering step (c) the volume is rendered with either axis-aligned slices (left) or view-aligned slices (right) using the texure pairs.

BIBLIOGRAPHY

[1] Accelerated Strategic Computing Initiative (ASCI), *http://www.llnl.gov/asci*.

[2] M. J. Ackerman, “The Visible Human Project,” *J. Biocomm.*, Vol. 18, p. 14, 1991.

[3] D. Aliaga, “Visualization of Complex Models Using Dynamic Texture-based Simplification,” *IEEE Visualization ‘96*, pp. 101-106, 1996.

[4] C. Bajaj, V. Pascucci, and D. Schikore, “The Contour Spectrum,” *IEEE Visualization ‘97*, pp. 167-173, 1997.

[5] W. Bethel, "Visualization Dot Com," *IEEE Computer Graphics and Applications*, May/June 2000.

[6] W. Bethel, B. Tierney, J. Lee, D. Gunter, and S. Lau, “Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization,” *Proc. of Supercomputing ‘00*, 2000.

[7] J. Blinn, “Light Reflection Functions for Simulation of Clouds and Dusty Surfaces,” *Proc. of SIGGRAPH ‘82*, Vol. 16, No. 3, pp. 21-29, 1982.

[8] C. P. Botha and F. H. Post, “Interactive Previewing for Transfer Function Specification in Volume Rendering,” *Proc. of the Symposium on Data Visualization*, pp. 71-, 2002.

[9] M. L. Brady, K. K. Jung, H.T. Nguyen, and T. PQ. Nguyen, “Interactive Volume Navigation,” *IEEE TVCG*, Vol. 4, No. 3, pp. 243 - 256, 1998.

[10] P. Bunyk, A. Kaufman, and C. T. Silva, “Simple, fast and robust ray casting of irregular grids,” *Scientific Visualization Conference (dagstuhl)*, pp. 30, 1997.

[11] B. Cabral, N. Cam, and J. Foran, “Accelerated Volume Rendering And Tomographic Reconstruction Using Texture Mapping Hardware,” *1994 Symp. on Vol. Vis.*, pp. 91-98, 1994.

[12] W. Cai and G. Sakas, “Maximum Intensity Projection Using Splatting in Sheared Object Space,” *Proc. EUROGRAPHICS ‘98*, pp. C113-C124, 1998.

[13] B. Carneiro, C. Silva, and A. Kaufman, “Tetra-Cubes: An Algorithm to Generate 3D Isosurfaces based upon Tetrahedra,” *Anais do IX SIBGRAPI ‘96*, pp. 205-210, 1996.

[14] B. Chen, A. Kaufman, and Q. Tang, “Image-Based Rendering of Surfaces from Volume Data,” *IEEE Workshop on Volume Graphics*, 2001.

[15] S. Chen, “QuickTime VR: An Image-based Approach to Virtual Environment Navigation,” *Proc. of The 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 29-38, 1995.

[16] E. Chen and L. Williams, “View Interpolation for Image Synthesis,” *Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 279-288, 1993.

[17] J-J Choi and Y. Shin, “Efficient Image-Based Rendering Of Volume Data,” *Proc. Pacific Graphics ‘98*, pp. 70-78, 1998.

[18] B. Csebfalvi, A. Konig, and E. Groller, “Fast Maximum Intensity Projection Using Binary Shearwarp Factorization,” *TR-186-2-98-27 at the Institute of Computer Graphics, Vienna University of Technology*, 1998.

[19] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman, “High-Quality Volume Rendering Using Texture Mapping Hardware,” *Proc. SIGGRAPH/Eurographics Graphics Hardware Workshop*, 1998.

[20] P. Debevec, G. Borshukov, and Y. Yu, “Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping,” *Proc. 9th Eurographics Rendering Workshop*, 1998.

[21] X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey, “Multi-layered imposters for accelerated rendering,” *Proc. Eurographics ‘99*, pp. 145-156, 1999.

[22] R. Dreben, L. Carpenter, and P. Hanrahan, “Volume Rendering,” *Proc. of SIGGRAPH ‘88*, Vol. 22, No. 4, pp. 65-74, 1988.

[23] M. Duchaineau, M. Wolinsky, D. Sigeti, C. Aldrich, and M. Mineev, “ROAMing Terrain: Real-time Optimally Adapting Mesh,” *IEEE Visualization ‘97*, pp. 81-88, 1997.

[24] G. Dutton, “Locational Properties of Quaternary Triangular Meshes,” *Proc. of the Fourth International Symp. Spatial Data Handling*, pp. 901-910, 1990.

[25] H. Edelsbrunner, “Dynamic Data Structures For Orthogonal Intersection Queries,” *Technical Report F59, Inst. Informationsverarb, Tech. Univ. Graz*, Graz, Austria, 1980.

[26] K. Engel, M. Kraus, and T. Ertl, “High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading,” *Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop on Grahics Hardware*, pp. 9-16, 2001.

[27] S. Fang, T. Biddlecome, and M. Tuceryan, “Image-based Transfer Function Design for Data Exploration in Volume Visualization,” *IEEE Visualization ‘98*, pp. 319-326, 1998.

[28] J. Gao, J. Huang, H-W Shen, and J. Kohl, “Visibility Culling Using Plenoptic Opacity Functions for Large Volume Visualization,” *IEEE Visualization ‘03*, pp. 341-348, 2003.

[29] J. Gao and H-W Shen, “Parallel View-dependent Isosurface Extraction Using Multi-Pass Occlusion Culling,” *Proc. IEEE Symposium On Parallel and Large-data Vis. and Graphics 2001*, pp. 67-74, 2001.

[30] C. Giertsen, “Volume Visualization of Sparse Irregular Meshes,” *IEEE Computer Graphics & Applications*, Vol.12, No. 2, pp. 40-48, 1992.

[31] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, "The Lumigraph," *Proc. of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 43-54, 1996.

[32] N. Greene, "Environment Mapping and Other Applications of World Projections," *IEEE Computer Graphics &Applications*, Vol. 6, No. 11, pp. 21-29, 1986.

[33] N. Greene and M. Kass, "Approximating Visibility with Environment Maps," *Technical Report 41, Apple Computer, Inc.*, 1993.

[34] S. Guthe, S. Roettger, A. Schieber, W. Strasser, and Thomas Ertl, "High-Quality Unstructured Volume Rendering on the PC Platform," *Eurographics/SIGGRAPH Graphics Hardware Workshop*, 2002.

[35] S. Guthe, M. Wand, J. Gonser, and W. Strasser, "Interactive rendering of large volume data sets," *IEEE Visualization'02*, pp 53-59, 2002.

[36] T. He, L. Hong, A. Kaufman, and H. Pfister, "Generation of Transfer Functions With Stochastic Search Techniques," *Proc. IEEE Vis. '96*, pp. 227-234, 1996.

[37] W. Heidrich, M. McCool, and J. Stevens, "Interactive Maximum Projection Volume Rendering," *Proc. IEEE Vis. '95*, pp. 11-18, 1995.

[38] Hugues Hoppe, "Progressive Meshes," *Proc. SIGGRAPH '96*, pp. 99-108, 1996.

[39] J. Kajiya and B. Von Herzen, "Ray Tracing Volume Densities," *ACM SIGGRAPH Comput. Gr.*, Vol. 18, No. 4, pp. 165-174, 1984.

[40] K. Kaneda, Y. Dobashi, K. Yamamoto, and H. Yamashita, "Fast Volume Rendering with Adjustable Color Maps," *Proc. 1996 Symp. on Vol. Vis.*, pp. 7-14, 1996.

[41] J. Kniss, G. Kindlmann, and C. Hansen, "Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets," *IEEE Visualization '01*, pp. 255-262, 2001.

[42] J. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, C. Hansen, "Interactive Texture-Based Volume Rendering for Large Data Sets", *IEEE Computer Graphics & Applications*, Vol. 21, No. 4, pp. 52-61, 2001.

[43] G. Knittel and W. Strasser, "Vizard - Visualization Accelerator for Real-time Display," *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pp. 139-146, 1997.

[44] A. König and E. Gröller, "Mastering Transfer Function Specification by Using VolumePro Technology," *IEEE Proc.of 17th Spring Conf. on Comp. Graphics (SCCG'01)*, Budmerice, Slovakia, pp. 279-286, April 2001.

[45] P. Lacroute and M. Levoy, "Fast Volume Rendering Using A Shear-Warp Factorization Of The Viewing Transformation," *Proc. of SIGGRAPH '94*, pp. 451-458, 1994.

[46] E. LaMar and V. Pascucci, "A Multi-Layered Image Cache for Scientific Visualization," *IEEE Symp. on Parallel and Large-Data Visualizatin and Graphics*, pp. 61-68, 2003.

[47] M. Levoy, "Display Of Surface From Volume Data," *IEEE Comp. Graph. & Appl.*, Vol. 8, No. 5, pp. 29-37, 1988.

[48] M. Levoy, "Efficient Ray Tracing Of Volume Data," *ACM Trans. Comp. Graph.*, Vol. 9, No. 3, pp. 245-261, 1990.

[49] M. Levoy, "Expanding the Horizons of Image-Based Modeling and Rendering," *Panel Presentation at Siggraph '97 on Image-Based Rendering*, 1997.

[50] M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. Siggraph '96*, pp. 31-42, 1996.

[51] A. Lippman, "Movie-Maps: An Application of the Optical Videodisc to Computer Graphics," *Proc. SIGGRAPH '80*, pp. 32-42, 1980.

[52] D. Lischinski and A. Rappaport, “Image-based Rendering for Non-Diffuse Synthetic Scenes,” *Rendering Techniques ‘98,* pp. 301-314, Vienna, Austria, 1998.

[53] Y. Livnat and C. Hansen, “View Dependent Isosurface Extraction,” *Proc. IEEE Vis. ‘98*, pp. 175 - 180, 1998.

[54] T. Lokovich and E. Veach, “Deep Shadow Maps,” *ACM SIGGRAPH ‘00*, pp. 385-392, 2000.

[55] W. E. Lorensen and H. E. Cline, “Marching Cubes: A High Resolution 3D Suface Construction Algorithm,” *Proc. of SIGGRAPH ‘87*, pp. 163-169, 1987.

[56] E. J. Luke, C. D. Hansen, “Semotus Visum: A Flexible Remote Visualization Framework”, *Proc. IEEE Visualization ‘02*, pp., 2002.

[57] P. Maciel and P. Shirley, “Visual Navigation of Large Environments Using Textured Clusters,” *Proc. Symposium on Interactive 3D Graphics SIGGRAPH ‘95,* 1995.

[58] M. Magallon, M Hopf, T. Ertl, “Parallel Volume Rendering Using PC Graphics Hardware”, *Ninth Pacific Conference on Computer Graphics and Applications ’01,* Tokyo, 2001.

[59] W. R. Mark, L. McMillan, and G. Bishop, “Post-Rendering 3D Warping,” *Proc. 1997 Symp. on Interactive 3D Graphics,* pp. 7-16, 1997.

[60] J. Marks, B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. "Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation," *Proc. of SIGGRAPH ‘97,* pp. 389-400, 1997.

[61] N. Max, “Optical Models for Direct Volume Rendering,” *IEEE TVCG*, pp. 99-108, 1995.

[62] N. Max, P. Hanrahan, and R. Crawfis, “Area and Volume Coherence For Efficient Visualization of 3D Scalar Functions,” *ACM SIGGRAPH Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, No. 5, pp. 27-33, 1990.

[63] N. Max and K. Ohsaki, “Rendering Trees From Precomputed Z-buffer View,” *Proc. Eurographics Workshop on Rendering*, June 1995.

[64] J. Meredith and K-L Ma, “Multiresolution View-Dependent Splat Based Volume Rendering of Large Irregular Data,” *Parallel and Large Data Visualization Symposium*, October 22-23, 2001.

[65] L. McMillan and G. Bishop, “Plenoptic Modeling: An Image-Based Rendering System,” *Proc. SIGGRAPH ‘95*, pp. 39 - 46, 1995.

[66] L. Mroz, A. Konig, and E. Groller, “Maximum Intensity Projection at Warp Speed,” *Technical Report TR-186-2-99-24 Institute of Computer Graphics and Algorithms Vienna University of Technology: A-1040 Karlsplatz 13/186/2*, 1999.

[67] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, “IBR-Assisted Volume Rendering,” *LBHT Vis. ‘99*, pp. 5-8, 1999.

[68] S. Muraki, M. Ogata, K-L Ma, K. Koshizuka, K. Kajihara, X. Liu, Y. Nagano, and K. Shimokawa, “Next-Generation Visual Supercomputing using PC Clusters with Volume Graphics Hardware Devices,” *Proc. ACM/IEEE Supercomputing 2001*, 2001.

[69] M. Oliveira, “Image-Based Modeling and Rendering Techniques: A Survey,” *RITA - Revista de Informa’tica Teo’rica e Aplicada*, Volume IX, Number 2, pp. 37-66, October 2002.

[70] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan, “Interactive Ray Tracing for Isosurface Rendering,” *Proc. IEE Vis. ‘98*, pp. 233-238, 1998.

[71] S. Parker, P. Shirley, Y. Livnat, C. Hansen, P.-P. Sloan, and M. Parker, “Interacting with Gigabyte Volume Datasets on the Origin 2000,” *The 41st Annual Cray User’s Group Conference*, 1999.

[72] H. Pfister, J. Hardenbergh, G. Knittel, H. Lauer, and L. Seiler, “The VolumePro Real-Time Ray-Casting System,” *Proc. of SIGGRAPH ‘99*, pp. 251-260, 1999.

[73] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. S. Avila, K. Martin, R. Machiraju, and J. Lee, “The Transfer Function Bake-Off,” *IEEE Computer Graphics and Applications*, Vol. 21, No. 3, pp. 16-22, 2001.

[74] T. Porter and T. Duff, “Compositing Digital Images,” *Proc. of SIGGRAPH ’84*, pp. 253-259, 1984.

[75] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle, “View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data,” *Proc. of 8th Eurographics Workshop on Rendering*, pp. 23-34, 1997.

[76] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, “Interactive Volume Rendering On Standard PC Graphics Hardware Using Muti-textures and Multi-stage Rasterization,” *Proc. of Eurograhics Workshop on Graphics Hardware*, pp. 109-118, 2000.

[77] S. Roettger, M. Kraus, and T. Ertl, “Hardware-Accelerated Volume And Isosurface Rendering Based On Cell-Projection,” *IEEE Visualization 2000*, pp. 109-116, 2000.

[78] S. Roettger and T. Ertl, “A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructrued Grids,” *Proc. 2002 Symposium on Volume Visualization*, 2002.

[79] P. Sabella, “A Rendering Algorithm for Visualizing 3D Scalar Fields,” *Proc. of SIGGRAPH ‘88*, Vol. 22, No. 4, pp. 51-55, 1988.

[80] G. Sakas, M. Grimm, and A. Savopoulos, “Optimized Maximum Intensity Projection,” *Proc. of 5th EUROGRAPHICS Workshop on Rendering Techniques*, pp.55-63, 1995.

[81] Y. Sato, N. Shiraga, S. Nakajima, S. Tamura, and R. Kikinis, “LMIP: Local maximum intensity projection - a new rendering method for vascular visualization,” *Journal of Computer Assisted Tomography*, 22(6), 1998.

[82] G. Schaufler, “Dynamically Generated Imposters,” *MVD ‘95 Workshop ‘95*, pp. 129-136, 1995.

[83] G. Schaufler, “Per-Object Image Warping with Layered Imposters,” *Rendering Techiniques*, pp. 145-156, 1998.

[84] G. Schaufler and W. Sturzlinger, “A Three Dimensional Image Cache for Virtual Reality,” *Proc. of Eurographics ‘96*, 1996.

[85] S. M. Seitz and C. R. Dyer, “View Morphing,” *Proc. SIGGRAPH ‘96*, pp. 21-30, 1996.

[86] J. Shade, S. Gortler, Li-Wei He, and R. Szeliski, “Layered Depth Images,” *ACM SIGGRAPH ‘98*, pp. 231-242, 1998.

[87] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder, “Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments,” Proc. of Siggraph ‘96, pp. 75-82, 1996.

[88] N. Shareef and R. Crawfis, “A View-dependent Approach to MIP for Very Large Data,” *Proc. SPIE EI ‘02*, Vol. 4665, pp. 13-21, 2002.

[89] P. Shirley and A. Tuchman, “A Polygonal Approximation to Direct Scalar Volume Rendering”, *Proc. of SIGGRAPH ‘90*, Vol. 24, No. 5, pp. 63-70, 1990.

[90] H-Y Shum and S. B. Kang, "A Review of Image-based Rendering Techniques", *IEEE/SPIE Visual Communications and Image Processing (VCIP) 2000*, Perth, pp. 2-13, 2000.

[91] C Silva and J. Mitchell, “ The Lazy Sweep Ray Casting Algorithm for Rendering Irregular Grids,” *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 2, pp. 142-157, 1997.

[92] V. Srivastava, U. Chebrolu, and K. Mueller, “Interactive Transfer Function Modification For Volume Rendering Using Pre-shaded Sample Runs,” *Pacific Graphics ‘02*, Beijing, China, Oct. 2002.

[93] V. Srivastava, U. Chebrolu, and K. Mueller, “Interactive Transfer Function Modification For Volume Rendering Using Compressed Sample Runs,” 2003.

[94] C. Stein, B. Becker, and N. Max, “Sorting and Hardware Assisted Rendering for Volume Visualization,” *Proc. of Symposium on Volume Visualization ‘94*, Washington, D.C., 1994.

[95] A. Van Gelder and K. Kim, “Direct Volume Rendering via 3D Texture Mapping Hardware,” *Proc. 1996 Volume Rendering Sympiosium*, pp. 23-30, 1996.

[96] The Visualization Toolkit (VTK), *http://www.vtk.org/*, function vtkVolumeRayCastMIPFunction.

[97] M. Weiler and R. Westermann, “Level-Of-Detail Volume Rendering via 3D Textures,” *Proc. IEEE Vol. Vis. and Graphics Symp.*, 2000.

[98] R. Westermann and T. Ertl, “Efficiently Using Graphics Hardware in Volume Rendering Applications,” *Proc. of SIGGRAPH ‘99*, pp. 169-177, 1999.

[99] L. Westover, “Interactive Volume Rendering,” *1989 Chapel Hill Volume Visualization Workshop*, pp. 9-16, 1989.

[100] J. Wilhelms and A. Van Gelder, “A Coherent Projection Approach For Direct Volume Rendering,” *Proc. of SIGGRAPH ‘91*, pp. 275-284, 1991.

[101] P. L. Williams and N. Max, “A Volume Density Optical Model,” *Proc. Workshop on Volume Visualization*, pp. 61-68, 1992.

[102] P. L. Willams, N. Max, and C. Stein, “A High Accuracy Volume Renderer for Unstructured Data,” *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No.1, 1998.

[103] R. Yagel, D. M. Reed, A. Law, P. Shih, and N. Shareef, “Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing,” *Proc. Symposium on Volume Visualization*, pp. 55-62, 1996.

[104] S. Y. Yen, S. Napel, and G. D. Rubin, “Fast Sliding Thin Slab Volume Visualization,” *Proc. IEEE Volvis. ‘96*, pp. 79-86, 1996.

[105] I. Yoon, J. Demers, T. Y. Kim, and U. Neumann, "Accelerating Volume Visualization by Exploiting Temporal Coherence," *IEEE Visualization 97 LBHT*, pp 21-24, 1997.

[106] I. Yoon and U. Neumann, "IBRAC: Image-Based Rendering Acceleration and Compression", *Journal of High Performance Computer Graphics, Multimedia, and Visualization*, 2002.

[107] C. Zhang and T. Chen, "A Survey on Image-Based Rendering - Representation, Sampling and Compression", E*URASIP Signal Processing: Image Communication*, pp. 1-28, Vol. 19, No.1, 2004.