ADAPTIVE VISUALIZATION OF DYNAMIC UNSTRUCTURED MESHES

by

Steven Paul Callahan

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

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

August 2008

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Steven Paul Callahan

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

5/5/8	Cláudio T. Silva Chair: Cláudio T. Silva
Apr 28, 2008	[illegible] Robert M. Kirby II
APR 28, 2008	[illegible] Peter Shirley
4/28/2008	[illegible] Valerio Pascucci
4/28/2008	[illegible] João L. D. Comba

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

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

05/07/08

Date

Cláudio T. Silva

Cláudio T. Silva
Chair, Supervisory Committee

Approved for the Major Department

[illegible]

Martin Berzins
Chair/Dean

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

The amount of data available from simulation and measurement is growing at an incredible rate. A major challenge for the visualization community is to develop methods that allow users to explore these data interactively. For three-dimensional scalar fields, direct volume rendering has become an important technique in research and commercial settings. Interactive volume rendering requires the efficient use of available computational resources to keep pace with the disparity, resolution, and complexity of the volumes that are commonly produced from simulations (e.g., computational fluid dynamics or structural mechanics) and measurements (e.g., environmental observation and forecasting systems). For structured grids, direct volume rendering is well-studied and sufficiently straightforward with modern graphics hardware. This is not the case with unstructured volumes, because the elements that compose the mesh do not so easily map to current hardware. These datasets may be extremely large and contain more than a single static instance. Therefore, advanced solutions are required to achieve interactive visualization of this type of data.

The goal of this dissertation is to provide several new techniques to facilitate the visualization of disparate unstructured meshes. Two new methods are proposed to accelerate volume rendering for the case of static data, one of which operates in object-space and the other in image-space. Acceleration methods may not always be enough, however, to allow interactive visualization for data that are too large to fit in the main memory of a computer. Therefore, a new progressive rendering approach is proposed that adaptively refines a visualization using remote resources. These new techniques are of great assistance for a large class of static imagery. However, dynamic volumes that change over time create unique challenges because of the amount of data that needs to be transmitted at each step. To address this issue, a new method for efficiently handling time-varying unstructured volumes is also presented in this dissertation.

Together, these methods for interactive visualization provide a powerful framework for analyzing large amounts of unstructured data. To demonstrate this, a final application for transfer function design that combines many of these approaches is presented. This application includes an evaluation performed by a group of expert users to elaborate on the importance of these proposed techniques for interactive visualization.

To Kristy

CONTENTS

ABSTRACT iv

LIST OF FIGURES ix

LIST OF TABLES xi

ACKNOWLEDGMENTS xii

CHAPTERS

1. INTRODUCTION 1

1.1 Direct Volume Rendering of Unstructured Meshes 1
1.1.1 Sampling 2
1.1.2 Classification 3
1.1.3 Compositing 4
1.2 Dissertation Motivation 5
1.3 Thesis Statement 5
1.4 Dissertation Objectives 5

2. BACKGROUND 7

2.1 Hardware Assisted Volume Rendering of Unstructured Grids 7
2.2 Acceleration Techniques for Volume Rendering 9
2.3 The Bilateral Filter 10
2.4 Remote Visualization 11
2.5 Time-Varying Visualization 11
2.6 Transfer Function Design 13

3. OBJECT-SPACE ACCELERATION FOR VOLUME RENDERING 16

3.1 Algorithm Overview 17
3.1.1 Preprocessing the Tetrahedral Mesh 18
3.1.2 Rendering the Points 19
3.1.2.1 Circular Footprints 19
3.1.2.2 Ellipsoidal Footprints 20
3.1.2.3 Exact Projection Footprints 21
3.1.3 Compositing the Points 22
3.2 Level Of Detail 22
3.3 Results 24
3.3.1 Image Quality 26
3.4 Discussion 26
3.5 Summary 28

4. IMAGE-SPACE ACCELERATION FOR VOLUME RENDERING 29
4.1 The Algorithm 31
4.1.1 The Joint Bilateral Upsampling Filter 31
4.1.2 Computing the Reference Image 32
4.1.3 Implementation 33
4.2 Results 34
4.2.1 Timing Results 34
4.2.2 Quality Results 36
4.3 Discussion 38
4.4 Summary 40

5. PROGRESSIVE VOLUME RENDERING 41
5.1 System Overview 42
5.2 The Server 43
5.2.1 Geometry Processing 43
5.2.2 Network Protocol 44
5.3 The Client 45
5.3.1 Interactive Mode 47
5.3.2 Progressive Mode 48
5.3.3 Completed Mode 49
5.3.4 Considerations 50
5.4 Results 51
5.5 Discussion 54
5.6 Summary 56

6. VOLUME RENDERING TIME-VARYING SCALAR FIELDS 58
6.1 Adaptive Time-Varying Volume Rendering 60
6.1.1 Compression 61
6.1.2 Data Transfer 62
6.1.3 Volume Rendering 63
6.1.3.1 Direct Volume Rendering 64
6.1.3.2 Isosurfaces 64
6.1.4 Time-Varying Level-of-Detail 65
6.1.4.1 Local Sampling 66
6.1.4.2 Global Sampling 67
6.2 Results 68
6.2.1 Datasets 68
6.2.2 Compression 69
6.2.3 Rendering 71
6.2.4 Level-of-Detail 71
6.3 Discussion 72
6.4 Summary 74

7. APPLICATION: A FRAMEWORK FOR TRANSFER FUNCTION DESIGN . . 75
7.1 System Overview 78
7.2 Scalar Range Mapping 80
7.3 Transfer Function Ensembles 83
7.3.1 Blending Strategies 85

7.4 Time Varying Datasets 87
7.5 Evaluation 89
7.6 Discussion 91
7.7 Summary 91

8. CONCLUSIONS AND FUTURE WORK 93

REFERENCES 95

LIST OF FIGURES

1.1 Direct volume rendering of an unstructured volume simulating the pressure in the air around a fighter jet 3

1.2 A cylinder of semitransparent particles 4

3.1 Point-based volume rendering algorithm overview. 17

3.2 An illustration of the point-based strategy in 2D 18

3.3 The transformation taking a unit tetrahedron to a general tetrahedron 19

3.4 Rendering of the Heart dataset with HAVS and with this algorithm 20

3.5 Hardware-accelerated point-based volume rendering is faster than the current state of the art approaches 21

3.6 Dynamic level-of-detail of the SPX2 dataset with point resizing 24

3.7 For adaptive meshes, many elements are subpixel sized 25

3.8 The errors associated with point-based rendering 27

4.1 A comparison of joint bilateral upsampling with linear upsampling for an opaque mesh 30

4.2 A comparison of volume rendering accelerated with bilateral and other upsampling methods 32

4.3 Timing statistics for a software raycaster and a hardware raycaster 35

4.4 Quality comparisons of joint bilateral versus linear upsampling 36

4.5 Quality comparisons for various resolutions of upsampling 37

4.6 Removing high frequency noise from object-space acceleration 39

5.1 A sequence of progressive volume rendering steps 42

5.2 The client-server architecture 43

5.3 A zoomed-in view of the STP dataset captured during interaction 48

5.4 The progressive volume renderer on the client 49

5.5 A snapshot of the interaction with the *Completed Mode* 50

5.6 Quality comparisons of progressive steps 53

5.7 A zoomed-in portion of the F16 dataset shown progressively 54

5.8 A zoomed-in portion of the Fighter dataset shown progressively 55

6.1 Different time instances of the Turbulet Jet dataset 59

6.2 An overview of the proposed system for dynamic volume rendering 61

6.3 An illustration of decompression of vector quantized scalars 63

6.4 The time-varying level-of-detail strategy 67

6.5 Comparison of direct volume rendering using uncompressed and compressed scalars 70

6.6 Different time instances 72

7.1 An example demonstrating some of the capabilities of the proposed system for transfer function design 76

7.2 The visualization process provided by the proposed system 78

7.3 The user interface for interactive transfer function specification 79

7.4 A description of how high-dynamic range data affect a lookup table 81

7.5 Volume exploration of the San Fernando earthquake simulation through range mapping 82

7.6 An ensemble is used to combine multiple transfer functions with different blending strategies 86

7.7 Multiple 1D scalar transfer functions are used for different time steps of a time-varying simulation 87

7.8 The time-varying interface 88

LIST OF TABLES

3.1 Performance summary of point-based volume rendering 25

3.2 Results of point-based volume rendering 26

4.1 Experimental datasets used for measuring rendering performance 34

5.1 Experimental datasets for progressive rendering 51

5.2 Performance analysis of the preprocessing and one step of the progressive volume rendering 52

5.3 Latency analysis of different server settings 53

6.1 Results of compression sizes, ratios, and error for time-varying volumes. 69

6.2 Performance measures for static and time-varying scalar fields 71

ACKNOWLEDGMENTS

I am very grateful for those that made this dissertation possible. I would like to thank my advisor, Cláudio Silva, for his guidance, effort, and support, without which this dissertation would not have been written. I would like to thank my committee members, who helped organize and provided valuable feedback for this work: Cláudio Silva (chair), Mike Kirby, Peter Shirley, Valerio Pascucci, and João Comba.

Many collaborators had a substantial impact on the work that is summarized in this dissertation. I would like to thank the following co-authors for their contributions: Erik Anderson, Louis Bavoil, Fabio Bernardon, Linh Ha, Carlos Scheidegger, and John Schreiner. Others who provided assistance and advice include members of the VisTrails team (under direction of Cláudio Silva and Juliana Freire) and members of the Visualization and Geometric Computing group (under direction of Cláudio Silva). Of these groups, I would especially like to thank Juliana Freire, David Koop, Emanuele Santos, and Huy Vo. The entire staff of the Scientific Computing and Imaging Institute and the School of Computing at the University of Utah have been very helpful. I would like to thank Karen Feinauer and Deborah Zemek for their assistance in completing the necessary documentation to finish this dissertation.

I would like to thank J. Schneider for his vector quantization code and acknowledge Hopenfeld and MacLeod (University of Utah) for the heart dataset, O'Hallaron and Shewchuk (CMU) for the earthquake dataset, Notrosso (electricite de France) for the SPX dataset, Neely and Batina for the fighter dataset, Tremel (EADS-Military) for the F16 dataset, Ma (UC Davis) for the Turbulent Jet and Five Jets datasets. I thank NVIDIA and ATI for donating hardware for this research. My work has been partially funded by the Department of Energy under the ASCI VIEWS program and the MICS office, the National Science Foundation (grants CCF-0401498, EIA-0323604, OISE-0405402, IIS-0513692, CCF-0528201), Sandia National Laboratories, Lawrence Livermore National Laboratory, the Army Research Office, an IBM Faculty Award, a University of Utah Seed Grant, and a University of Utah Graduate Research Fellowship.

Finally, I would like to thank my family. I am extremely grateful to my wife, Kristy, for her unending patience and support. Without her I probably would not have embarked on this five-year detour, and I certainly would not have survived it. I am also grateful to my parents, Janice and

Lowell, for instilling the importance of education at an early age and to Kristy's parents, Helen and Verlon, for valuable encouragement.

CHAPTER 1

INTRODUCTION

In scientific computing, large scale simulations are frequently performed on large clusters of parallel machines. The ability to simulate phenomena that are not easily tested in the field is of the utmost importance for fields such as computational fluid dynamics and structural mechanics. A major concern for the simulation scientists is that the amount of data being generated far outpaces the ability to analyze it. Visualization is an important piece in the analysis process, because it presents the data in a manner that makes it easier to understand [97]. Future advances in science depend on the ability to comprehend these vast amounts of data being produced and acquired.

For 3D scalar fields, direct volume rendering has become an important method for gaining insight into large volumes of data. Interactive volume rendering for large unstructured volumes has been the subject of much research in the visualization community [124]. However, current techniques are still not capable of keeping pace with the size, disparity, and complexity of the volumes that are being produced. Thus, the problem is likely to be a challenge for the visualization community for years to come [85]. This dissertation presents several methods to improve the manner in which unstructured volumes are visualized through direct volume rendering.

It is common to represent a scalar function $f : D \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$ as sampled data by defining it over a domain D, which is represented by a tetrahedral mesh. For visualization purposes, we define the function f as linear inside each tetrahedron of the mesh. In this case, the function is completely defined by assigning values at each vertex $v_i(x,y,z)$, and is piecewise-linear over the whole domain. The domain D becomes a 3D simplicial complex defined by a collection of simplices c_i. It is important to distinguish the domain D from the scalar field f. The purpose of visualization techniques, such as direct volume rendering, is to study intrinsic properties of the scalar field f.

1.1 Direct Volume Rendering of Unstructured Meshes

For the visualization of three-dimensional scalar fields, direct volume rendering has emerged as a leading, and often preferred, method. In rendering volumetric data directly, a participating medium is composed of semitransparent material that can emit, transmit, and absorb light, thereby allowing one to “see through” (or see inside) the data. By changing the optical properties of the material, different lighting effects can be achieved [92]. Figure 1.1 shows an example of a volume rendering of the air flow around a fighter jet.

Direct volume rendering consists of three major components: sampling, classifying, and compositing [20]. To compute an image, the effects of the optical properties must be continuously integrated throughout the volume. However, since the volume is represented by discrete cells, this needs to be done in a piecewise manner. *Sampling* deals with selecting the piecewise steps that are taken through the volume, *classification* is the process of computing a color and opacity for each step, and *compositing* is the how these classified steps are blended together to form an image.

1.1.1 Sampling

The most difficult aspect for volume rendering unstructured grids is finding the sample locations throughout the volume that can be combined to create a final image. Sampling strategies generally fall into three categories: image-space, object-space, and hybrid approaches.

Image-space techniques compute the image by sampling the volume from the pixel plane of the image. Raycasting [50, 15] unstructured grids involves sending rays for each pixel in the image that originate from the viewpoint and pass through the volume. The cells are sampled at the locations where the ray intersects the boundary faces. Because the cells are traversed in the order they are encountered, ordering the samples happens automatically. Performing this raycasting using graphics hardware has been a popular subject of research [9, 138, 140, 96].

Object-space techniques view the problem from the other direction. Each cell in the volume is projected onto the image plane, resulting in an image. For correct compositing, the cells require a strict front-to-back or back-to-front ordering before the projection [144, 28, 125, 66, 30]. Because a projected cell can be decomposed into triangles [123], much of the algorithm can be performed on graphics hardware [67, 139, 149].

A final classification of sampling strategies are those that are hybrid—they operate in both object-space and image-space. Examples of hybrid algorithms are those that incrementally slice the volume with image-aligned planes [111, 51], and those that project the cells in object-space, but sort the fragments in image-space [24, 45, 22].

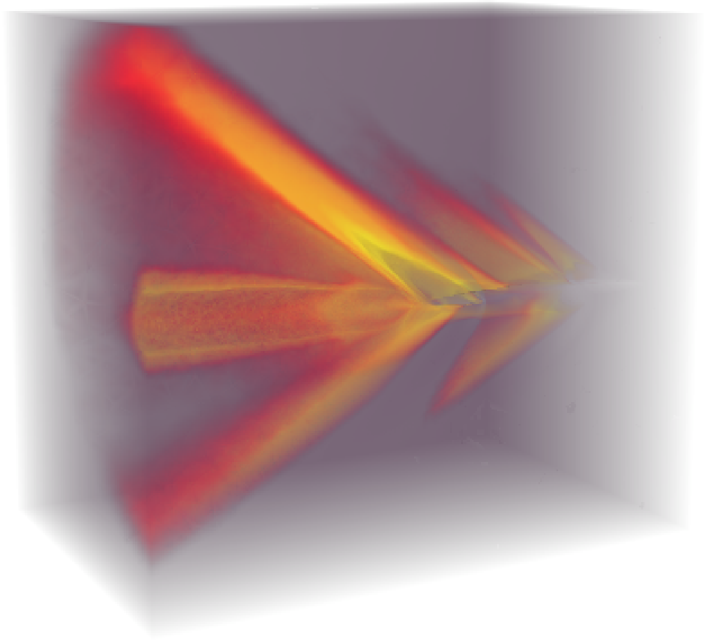


Figure 1.1. Direct volume rendering of an unstructured volume simulating the pressure in the air around a fighter jet. The volume is shown as a transparent cloud that allows internal features to be distinguished.

1.1.2 Classification

Direct volume rendering requires the use of optical models to simulate the look of a real medium that both occludes light (absorption) and adds to it (emmission) [12, 92, 60, 117]. The *low albedo* model is frequently used because it generates a high level of realism, but remains tractable because it does not involve multiple scattering effects.

Consider a cylindrical region of the volume with base B of area E and thickness Δs that contains ρ particles per unit that have a projected area A on B (see Figure 1.2). The volume of the cylinder can be expressed as $E\Delta s$, from which the area occluded on the base can be computed: $\rho AE\Delta s$. This leads to a ratio of occluded area $\rho AE\Delta s/E$, or just $\rho A\Delta s$, from which the intensity of light I can be expressed, using the following differential equation:

$$\frac{dI}{ds} = C(s)\rho(s)A - \rho(s)AI(s) \tag{1.1}$$

where the emissive term C denotes a glow per unit projected area. This equation has the solution

$$I(D) = I_0 e^{-\int_0^D \rho(t)A dt} + \int_0^D C(s)\rho(s)Ae^{-\int_s^D \rho(t)A dt} ds \tag{1.2}$$

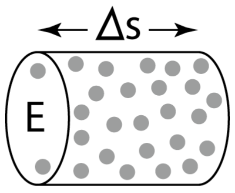


Figure 1.2. A cylinder of semitransparent particles.

where $s = 0$ at the edge of the volume and $s = D$ at the eye. An approximation to this equation can be derived using a Reimann Sum. The results divide the integral into n equal segments of size Δx:

$$I(D) \approx I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j \tag{1.3}$$

where

$$t_i = e^{-\rho(i\Delta x)A\Delta x}, \tag{1.4}$$

$$g_i = C(i\Delta x)\rho(i\Delta x)A. \tag{1.5}$$

Classification has received a lot of attention in the research commmunity, in terms selecting transfer functions that map the scalar value to colors [4, 61, 62, 104] as well as precomputing the optical properties for efficient lookup [41, 114, 94].

1.1.3 Compositing

Once a sample has been classified, the contribution is combined into the final image using alpha compositing [107]. For back-to-front compositing, the compositing equations are computed as a function of color and opacity:

$$\mathbf{c}_i = \mathbf{c}_i\alpha_i + \mathbf{c}_{i+1}(1-\alpha_i) \tag{1.6}$$

or similarly for front-to-back compositing:

$$\mathbf{c}_i = \mathbf{c}_{i-1} + \mathbf{c}_i\alpha_i(1-\alpha_{i-1}) \tag{1.7}$$

$$\alpha_i = \alpha_{i-1} + \alpha_i(1-\alpha_{i-1}) \tag{1.8}$$

for the steps before $(i-1)$ or after $(i+1)$ the current step (i), RGB colors ($\mathbf{c}$), and transparencies (α). The resulting color is considered *premultiplied* by alpha. Compositing in this form requires a strict ordering, such as the one imposed by marching a ray through the volume or imposed by sorting cells, because the equations are not commutative.

1.2 Dissertation Motivation

The time and space complexity of existing techniques for direct volume rendering are heavily dependent on the size (or number of simplices) and shape of the volume. Even for modestly sized volumes, existing techniques fail to remain interactive. Furthermore, visualizing volumes that are too large to fit in memory is very difficult, because most existing algorithms are not set up to operate outside memory constraints. The problem of interactive visualization is further complicated when the scalar field changes dynamically. Outside of the research presented in this dissertation, there are no methods available for handling this complex type of data.

The most common manner in which this problem is handled is to simplify or reduce the volume representation to one that is more manageable [49, 136]. However, fundamentally this is not an acceptable solution because reducing the resolution of the volume for analysis defeats the entire purpose of running simulations at a high resolution. Important features may be removed in a reduced representation. A more acceptable solution is to provide acceleration techniques for visualizing the data that create efficient approximations of the data for interaction. Returning to a full quality representation should always be possible so that high resolution features are not lost. The user should be allowed to control the speed versus quality trade-off in the analysis process.

1.3 Thesis Statement

Interactive volume rendering of dynamic unstructured grids requires a combination of novel software algorithms and frameworks that efficiently amortize recent hardware configurations.

1.4 Dissertation Objectives

This dissertation presents several methods for accelerating the visualization of large static and dynamic unstructured meshes through direct volume rendering. The goal is to provide a framework that enables interactive exploration of these volumes at a scale that until now has not been possible. The outline of this dissertation can be separated into five distinct contributions:

- A method is presented for volume rendering that simplifies the problem of projecting cells by approximating them with points [3]. This object-space method reduces the amount of geometry that is processed by graphics hardware and results in substantial savings for large static volumes.
- A complementary acceleration technique is introduced that operates in image-space. The algorithm is based on upsampling images computed at low resolutions and can be added to virtually any existing volume rendering algorithm to provide fast intermediate visualizations

during user interaction [23]

- A technique is described for performing progressive volume rendering of static volumes too large for conventional methods. A data server incrementally streams portions of the volume to a remote thin client that combines each piece into a final image [18, 19]. Intermediate representations are updated continuously, leaving the application interactive for the user.
- A framework is outlined for efficiently rendering volumes with dynamic, or time-varying scalar fields. A careful balance of computational resources is used to distribute the load and allow for interactivity [6, 7].
- Combining these new interactive techniques, an application is described for designing transfer functions within large, static and dynamic volumes. The result is a tool that is useful for analyzing and exploring disparate unstructured volumes [8]. An evaluation of the application is provided by expert users to validate the effectiveness of the methodology.

The rest of this dissertation is outlined as follows. Chapter 3 describes the object-space acceleration algorithm and Chapter 4 describes the image-space acceleration algorithm. Progressive volume rendering is outlined in Chapter 5 and the framework for dynamic scalar fields is provided in Chapter 6. Finally, the application and evaluation are given in Chapter 7 and conclusions and future work are presented in Chapter 8.

CHAPTER 2

BACKGROUND

This chapter provides a background of work related to that which is proposed in this dissertation. Volume rendering for unstructured grids using graphics hardware is described in Section 2.1 and additional acceleration techniques are described in Section 2.2. A brief summary of the bilateral filter for image processing is described in Section 2.3. Related work on handling remote and time-varying visualization is described in Sections 2.4 and 2.5, respectively. Finally, a synopsis of existing techniques for specifying transfer functions is given in Section 2.6.

2.1 Hardware Assisted Volume Rendering of Unstructured Grids

Volume rendering on a commodity PC has been the subject of much research recently, due to the steady increase in processing power on graphics processing units (GPUs) and the advent of programmable shaders. Here the state-of-the-art for hardware-assisted volume rendering of unstructured grids is summarized; for more detail the reader is refered to recent surveys by Silva et al. [124] and Krüger and Westermann [68].

For unstructured grids, volume rendering algorithms generally fall into three categories: ray-casting, splatting, and hybrid approaches. Recently, Weiler et al. [138] proposed an algorithm to perform ray-casting completely on the GPU by storing the mesh and traversal structure in GPU memory. This algorithm was made more efficient and extended to handle nonconvex meshes in subsequent work by Weiler et al. [140] and Bernardon et al. [9]. These algorithms benefit from low latency because they avoid CPU to GPU data transfers. However, the limited memory of GPUs prevents these algorithms from rendering even moderately sized datasets. To address this problem, Muigg et al. [96] proposed a method for bricking the volume in memory so that portions of the mesh can be passed to the GPU separately. The idea is to treat the CPU memory as a cache system for the GPU, resulting in a more scalable approach for hardware raycasting.

Pioneering work on tetrahedral splatting by Shirley and Tuchman [123] introduced the Projected Tetrahedra (PT) algorithm. For each viewpoint, PT decomposes a tetrahedron into one to four triangles that can be rendered efficiently in hardware. Subsequent work by Roettger et al. [115] improves upon the quality of the projection by incorporating arbitrary transfer functions. Similarly, Kraus et al. [67] presented another technique that uses programmable shaders to improve compositing and handle perspective projections. Unfortunately, the compositing of the triangles using PT requires an explicit visibility ordering that is implicit to raycasting. Many algorithms have been proposed to perform the visibility ordering in object-space [144, 28, 125, 30]) The goal of much of the recent work on PT has been to push more of algorithm onto the GPU. Wylie et al. [149] described how the decomposition of tetrahedra into triangles can be performed using graphics hardware. Marroquim et al. [91] extend this further to perform an approximate visibility sorting on the GPU as well.

Another efficient way for splatting is to represent the mesh as points instead of cells. Point-based representations of underlying meshes have been well-studied [116, 105, 13]. For structured grids, Zwicker et al. [155] introduced the use of convolution kernels to help mitigate errors associated with representing a full tetrahedron with a single point. This work was improved by Xue and Crawfis [150] as well as Chen et al. [25] to enhance the performance using graphics hardware. The work of Mao et al. [89] and Laur et al. [70] for point-based representations of triangle meshes formed a basis of Museth and Lombeyda's TetSplat [98] for unstructured volumes. TetSplat creates hierarchies of points at different resolutions to perform rendering of datasets at a large-scale using a limited, albeit useful, opaque representation of the volume that can be probed by cutting pieces away.

Hybrid approaches have also been introduced that combine some elements from raycasting and others from splatting. Incrementally slicing the tetrahedron in a series of screen-aligned planes was originally proposed by Reed et al. [111] and more recently adapted using geometry shaders by Georgii et al. [51]. A different approach introduced by Shareef et al. [121] represents the unstructured mesh as a Pixel Ray Image (PRI), a data structure that stores a series of rays cast from canonical planes that intersect cells in the mesh. This representation is then stored in textures and rendered using a slice based-approach. Recent work by Callahan et al. [22, 17] introduced the Hardware-Assisted Visibility Sorting (HAVS) algorithm which sorts in object-space and image-space. The algorithm operates on the triangles that compose the tetrahedral mesh. For each new view, the triangles are approximately sorted in object-space based on their centers. Then, in image-space, a data structured called the *k*-buffer is used to maintain a fixed list of fragments for each pixel that can be used to finalize the sorted order at a fragment level. The HAVS algorithm is fast, efficient, and flexible enough to handle dynamically changing data, and thus, it is used as a basis for the techniques proposed in Chapters 3, 5, 6, and 7 of this dissertation.

Graphics hardware has also been used in recent research to visualize isosurfaces of volumetric data. The classic Marching Cubes algorithm [79] and subsequent Marching Tetrahedra algorithm [34] provide a simple way to extract geometry from structured and unstructured meshes in object space. However, image-space techniques are generally more suitable for graphics hardware. One such algorithm, proposed by Sutherland et al. [127], uses alpha-test functionality and a stencil buffer to perform plane-tetrahedron intersections. Isosurfaces are computed by interpolating alpha between front and back faces and using XOR operation with the stencil buffer. Later, Röttger et al. [115] revised this algorithm and extended the PT algorithm to perform isosurfacing. Instead of breaking up the tetrahedra into triangles, the algorithm creates smaller tetrahedra which can be projected as triangles and tested for isosurface intersection using a 2D texture. More recent work by Pascucci [102] uses programmable hardware to compute isosurfaces. The algorithm considers each tetrahedron as a quadrilateral, which is sent to the GPU where the vertices are repositioned in a vertex shader to represent the isosurface. Chapter 6 extends the HAVS algorithm [22] described above to perform isosurface extraction of time-varying scalar fields.

2.2 Acceleration Techniques for Volume Rendering

Acceleration techniques for direct volume rendering have been the subject of much research in the visualization community. For a more complete summary of volume rendering algorithms for structured and unstructured grids, the reader is referred to recent surveys [109, 55, 124].

For structured grids, some of the original acceleration techniques are performed in image-space, and are still in use today to make ray casting more efficient. Levoy introduced the idea of casting one ray for multiple pixels [74], casting more rays in areas that vary across neighboring rays [73], or by not casting any rays in regions that do not contribute to the final image [72]. Extending these ideas, Danskin and Hanrahan [32] introduced adaptive ray sampling to sparsely sample along viewing rays in homogeneous regions. A similar object-space approach was introduced by Parker et al. [100], where the volume is partitioned into bricks that can be skipped during ray traversal. One class of acceleration techniques are multiresolution or level-of-detail (LOD) methods, which trade off quality of results for speed in rendering. With the advent of hardware-accelerated texture-based volume rendering [16], LaMar et al. [69] introduced a multiresolution approach for slicing regions of the volume at different resolutions that are stored in an octree. This allowed regions of less interest, such as those farthest from the view point, to be drawn at a coarser resolution. Weiler et al. [141] improved upon this idea by making it more efficient and guaranteeing consistent interpolation between different resolution levels.

For unstructured grids, the image-space approaches for ray casting introduced by Levoy for structured grids are still applicable. However, object-space approaches are not so easily adapted. To mitigate this problem, Leven et al. [71] sampled the unstructured grid regularly into an octree hierarchy that can be rendered using LOD techniques for structured grids. Volume simplification techniques, such as edge collapsing via the quadric error metric [49], provide the means for reducing the geometry representation to improve rendering performance. Adapting the simplification paradigm for LOD, Cignoni et al. [27] proposed a technique for creating a progressive hierarchy of tetrahedra that are stored in a multitriangulation data structure that is updated dynamically for interactivity. More recently, Callahan et al. [21, 17] introduced a simpler approach that samples the geometry during rendering based on a pre-computed importance. This approach avoids hierarchies and maintains interactivity by dynamically adjusting the amount of geometry rendered at each frame. This method is adapted in Chapter 6 of this dissertation to handle dyanamic geometry as well.

Several systems have previously taken advantage of multiple processors and multithreading to increase performance. The iWalk system [31] by Correa et al. uses multiple threads to prefetch geometry and manage a working set of renderable primitives for large triangles scenes. More recent work by Vo et al. [137] extends this idea to perform out-of-core management and rendering of large volumetric meshes. In Chapters 5 and 6 of this dissertation, algorithms are introduced that also take advantage of multiple processors to balance computation.

2.3 The Bilateral Filter

Since first introduced for image denoising [129], the bilateral filter has been used for many image processing applications, such as tone mapping for high dynamic range imaging [37]. The filter has also been used for problems other than image processing, such as mesh denoising [46]. For upsampling, the use of the bilateral filter is relatively new and has seen less use. Durand et al. [38] applied it to compute advanced shading effects with fewer samples. Sawhney et al. [118] adapted the filter for stereoscopic images at different resolutions. More recently, Kopf et al. [65] showed how joint bilateral upsampling could be used for computing downsampled solutions over an image and combining them with the original image. They demonstrated tone-mapping, colorization, stereo depth, and graph cuts as applications for the approach. In Chapter 4 of this dissertation, a method is presented for accelerating volume rendering that is similar to this latter technique. However, instead of enhancing an image with a downsampled solution, the method renders the image at a small resolution for increased performance, upsamples the image, then enhances it with a computed solution to achieve a high quality approximation.

2.4 Remote Visualization

The problem of remotely visualizing large datasets has been the subject of research for many years. The most widely recognized solutions perform the visualization task on large clusters using software algorithms [101] or with hardware-assisted algorithms [95] through the use of specialized graphics hardware [57]. Typically, an image is created using the cluster, then compressed for transmission to the client, where it is decompressed and displayed to the user [43]. Systems such as Vizserver [135] are available from vendors for performing client-server visualization in this manner. The Visapult system introduced by Bethel et al. [10] was developed to push more of the burden onto the client. This is done by rendering blocks of the data from the server in a distributed system and compositing the results on the client.

A less restrictive class of algorithms performs the visualization on more limited resources by assuming a simple server (e.g., a web server). To this end, Lippert et al. [76] introduced a system in which the server stores compressed wavelet splats that are transmitted to the client for rendering. As the splats are received by the client, the image is progressively refined. Another approach by Engel et al. [42] described a progressive isosurface visualization algorithm for use on the web. This is done by allowing the server to compute a hierarchy of isosurfaces that are transmitted to the client progressively. For efficiency, only the difference between two successive levels of the hierarchy is sent across the network. More recently, the client-server architecture provided by Kaehler et al. [59] performed visualization of Adaptive Mesh Refinement (AMR) data that are stored remotely on a server and adaptively rendered locally on a client by interpolating the hierarchical structure of the grids. A method is presented in Chapter 5 of this dissertation that is similar to this latter class of algorithms, but for the more difficult case of unstructured grids. A limited server prepares the geometry and streams it to the client in a series of progressive steps that avoid redundant transmission and unnecessary storage. This allows the client to receive the nonoverlapping geometry and refine the image with assistance of the GPU.

2.5 Time-Varying Visualization

The visualization of time-varying data is of obvious importance, and has been the source of substantial research. Of particular interest is the research literature related to compression and rendering techniques for this kind of data. For a more comprehensive review of the literature, the interested reader is referred to the recent surveys by Ma [84] and Ma and Lum [86].

Very little has been done for compressing time-varying data on unstructured grids. Therefore all the papers cited below focus on regular grids. Some researchers have explored the use of spatial data structures for optimizing the rendering of time-varying datasets [40, 87, 122]. The Time-Space Partitioning (TSP) Tree introduced by Shen et al. [122] used in those papers is based on an octree which is extended to encode one extra dimension by storing a binary tree at each node that represents the evolution of the subtree through time. The TSP tree can also store partial subimages to accelerate rendering by ray-casting.

The compression of time-varying isosurfaces and associated volumetric data with a wavelet transform was first proposed by Westermann [142]. With the advance of texture-based volume rendering and programmable GPUs, several techniques explored shifting data storage and decompression into graphics hardware. Coupling wavelet compression of structured grids with decompression using texturing hardware was discussed in work by Guthe et al. [52, 53]. They describe how to encode large, static datasets or time-varying datasets to minimize their size, thus reducing data transfer and allowing real-time volume visualization. Subsequent work by Strengert et al. [126] extended the previous approaches by employing a distributed rendering strategy on a GPU cluster. Lum et al. [80, 81] compressed time-varying volumes using the Discrete Cosine Transform (DCT). Because the compressed datasets fit in main memory, they are able to achieve much higher rendering rates than for the uncompressed data, which needs to be incrementally loaded from disk. Because of their sheer size, I/O issues become very important when dealing with time-varying data [152].

More related to this dissertation is the technique proposed by Schneider et al. [119]. Their approach relies on vector quantization to select the best representatives among the difference vectors obtained after applying a hierarchical decomposition of structured grids. Representatives are stored in textures and decompressed using fragment programs on the GPU. Since the multi-resolution representations are applied to a single structured grid, different quantization and compression tables are required for each time instance. Issues regarding the quality of rendering from compressed data were discussed by Fout et al. [47] using the approach described by Schneider et al. as a test case.

2.6 Transfer Function Design

Volume visualization through direct volume rendering has received much attention in the research community, yet the specification of transfer functions is still a challenging task. There have been many attempts to automate the process of specification. Levoy [75] described how boundaries can be visualized using the computed gradient of the scalar field to generate a transfer function. Along similar lines, Kindlmann and Durkin [61] proposed the use of histograms of the first and second derivatives to automatically generate a transfer function which emphasizes boundaries around homogeneous regions in the volume. An alternate approach was proposed by Fujishiro et al. [48] which uses a hyper Reeb graph to distinguish features of the dataset topologically.

Automatic techniques are good at extracting important boundary features from a volume. However they may not always give the user the desired visualization. Therefore, many systems have been developed that push more of the specification burden to the user. Marks et al. [90] introduced Design Galleries which allow the user to explore the transfer function parameter space by iteratively picking images from collections of visualizations. More recently, a parallel coordinate interface for parameter exploration was described by Tory et al. [131]. Tzeng et al. [132] introduced a system that learns regions of the volume by allowing the user to paint on slices of the volume. Along similar lines, Roettger et al. [113] include spatial information in standard 2D histograms to allow selection by region. Another high-level specification system was proposed by Rezk-Salama [112] that provides semantics for different visualization tasks and hides much of the underlying specification with the use of simple user interfaces for parameter exploration.

The most common approach to user-assisted transfer function specification is to incorporate histograms of the scalar field that classify the volume into different materials [36]. Bajaj et al. [4] proposed a system that analyzes the volume to extract isocontour information that is plotted for the user as a collection of 1D histograms for static datasets and 2D histograms for time-varying datasets. Kniss et al. [62] introduced widgets to facilitate transfer function specification on the multidimensional histograms introduced in Kindlmann and Durkin's work. Lum and Ma [82] modified the 2D joint histogram by showing the scalar pair along the gradient direction to introduce the Lighting Transfer Function that selectively enhances the boundary surface of interest. Another recent approach by Sereda et al. [120] uses Gaussian kernels to determine material transitions in CT scans and displays their L and H parameters as a 2D histogram that facilitates boundary extraction. The Scout system [93] takes a different approach to specification by directly applying mathematical expressions or queries to the data in the form of a programming language.

Focus and context approaches have received a lot of attention recently due to the recognition that features deserve different levels of focus. Hadwiger et al. [54] introduced a volume rendering system for segmented data that selects a different transfer function based on segmented voxel IDs. Similarly, Viola et al. [134] use importance compositing to assign higher opacities to more important features in segmented data. Svakhine et al. [128] extend this work to perform different rendering techniques for the unique materials in the data. For multiple volumes, Bruckner and Gröller [14] introduce a system that controls the compositing of inter-penetrating objects by using an opacity weighted average of two dimensional intersection transfer functions. Ma [83] and later König and Gröller [64] recognized that feature extraction would be simplified by defining transfer functions separately then consolidating them into one image with additive blending. Wu et al. [148] extended this idea with the use of a genetic algorithm for nonlinear combinations of transfer functions. Chapter 7 describes an approach similar to these additive approaches, except that it allows the user to blend transfer functions defined on different 1D and 2D histograms.

More specific techniques for transfer function specification have also been developed to handle high dynamic range data. By using a Gaussian transfer function, Kniss et al. [63] avoid the inaccuracies that are present with a low resolution lookup table. Another approach was introduced by Potts et al. [108], which uses a logarithmic scaling on their transfer function. Kraus et al. [67] use a similar logarithmic approach for lookup tables that was later used by Qiao et al. [110] to render large simulation data. These latter approaches based on logarithmic scaling assume data centered near zero. Recognizing that this is not always the case, a recent system by Yuan et al. [153] provides a high precision lookup table and the ability to nonlinearly zoom into regions of the transfer function for detailed specification. Since this is not always sufficient, Chourasia and Shulze [26] perform an automatic opacity-weighted histogram equalization to distribute the colors of a lookup table nonlinearly. The specification described in Chapter 7 allows more control by remapping scalars through a user-controlled range mapping.

Techniques have also been developed to handle transfer function specification for time-varying datasets. These techniques must consider the data for the entire time series when specifying transfer functions. The work of Jankun-Kelly and Ma [58] analyzes the generation of a single transfer function that works globally for a time-varying dataset. Doleisch et al. [35] presented a framework using time histograms to analyze unsteady flow data from computational fluid dynamics (CFD) simulations. Younesy et al. [151] proposed the Differential Time Histogram Table, using temporal coherence to minimize the amount of data required from disk to accelerate the rendering process. Usually a transfer function is designed to capture features that have a regular or periodic behavior. If a dataset presents different behavior, a complex transfer function is required to capture all features at once [84]. One recent paper has attempted to address the problem of aperiodic time sequences. Akiba et al. [1] extended the time varying transfer function framework to handle statistically dynamic time-varying volume data by performing temporal reduction and visualization feedback to find suitable time classified intervals. Wu et al. [147] extend the idea of transfer function fusion to create animations of static data by keyframing focus and context visualizations. Chapter 7 introduces a technique for specifying multiple transfer functions over time to handle time-varying data. These transfer functions can be key-framed, allowing custom transitions defined by the user.

CHAPTER 3

OBJECT-SPACE ACCELERATION FOR VOLUME RENDERING

This chapter describes interactive visualization of large, unstructured tetrahedral meshes using a point-based approach. The proposed technique renders each tetrahedron in the mesh as a single point. Approximating the tetrahedra in the mesh in this way is fast, and when combined with point reshaping methods, this technique produces high-quality renderings of large datasets at interactive rates.

There are many advantages to point-based rendering techniques. Many datasets contain tetrahedra that, after projection, represent subpixel sized areas. Due to the inclusion of connectivity information associated with rendering triangular or quadrilateral primitives, these tetrahedra are not optimally rendered. By representing these tetrahedra with single points, subpixel sized primitives can be rendered accurately using less data. Thus, rendering is often performed faster when using point-based techniques. In addition, difficulties such as storage and indexing additional data structures associated with dynamic level-of-detail for large datasets are substantially reduced. The introduced technique is flexible as it is not limited to vertex-centered data—it can be easily applied to large cell-centered volumes as well. In spite of its advantages, point-based rendering presents several sources of error that are addressed in this chapter.

The results of this work add several unique insights to the point-based rendering community. This chapter presents the following contributions:

- A novel point-based approach for rendering unstructured tetrahedral meshes with either vertex- or cell-centered data.
- Through the use of a compact point representation, this method eliminates the need to process and store connectivity information, while easing the implementation of level-of-detail strategies.
- Point reshaping using a GPU raycasting technique to minimize tetrahedron approximation error is introduced.

- Through the direct compositing of the point fragments in hardware shaders, the need for convolution operations used in existing splatting techniques [25] is avoided.

This chapter is organized as follows. Section 3.1 outlines the approach to point-based volume rendering of tetrahedral meshes, including the required preprocessing. Section 3.2 discusses the level-of-detail methodology, including performance and quality metrics. The results of the method are shown in Section 3.3 while a discussion of the work and summary are described in Sections 3.4 and 3.5.

3.1 Algorithm Overview

This volume rendering approach relies on the representation of complex tetrahedral meshes by simple approximations of the elements comprising the mesh. As illustrated in Figure 3.1, this algorithm can be broken down into three distinct components: preprocessing of the mesh, per-frame processing on the CPU, and per-frame processing on the GPU. After forming the approximation elements by finding a representative transform and scalar value for each point in a preprocessing step, the mesh is rendered at each frame as a dynamically adjusted set of screen-aligned points, as shown in Figure 3.2. The CPU first sorts the points front-to-back based on the current viewpoint and then determines the level-of-detail to use in the current frame. The GPU's vertex program then uses information in the transform associated with the point being rendered to re-size it, ensuring that the tetrahedron is adequately represented. Then, the fragment program culls fragments based on the shape of the approximating element using a technique reminiscent of Blinn et al. [78]. One fragment is kept in a texture and used to composite the ray-gap between the two fragments using a preintegrated table [41]. The error associated with

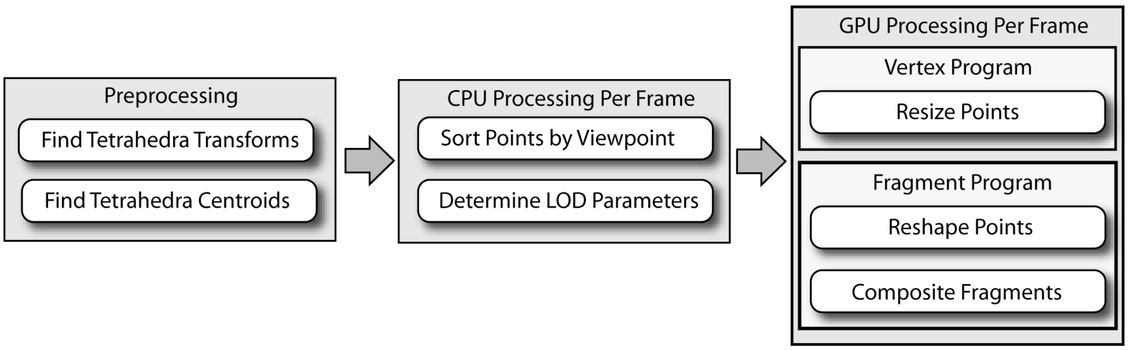


Figure 3.1. Point-based volume rendering algorithm overview.

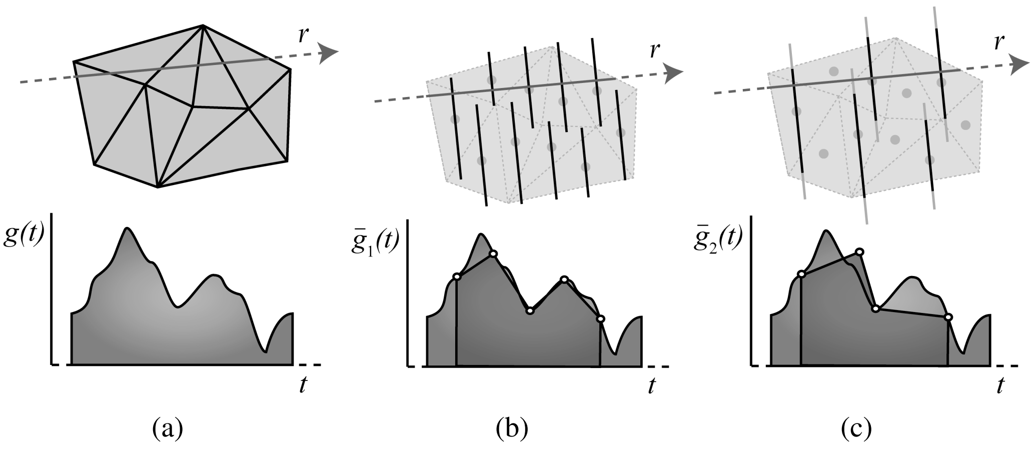


Figure 3.2. An illustration of the point-based strategy in 2D. (a) The original mesh with viewing ray r through the mesh and associated scalar function $g(t)$. (b) The point-based representation of the elements approximates the original function, $\bar{g}_1(t) \approx g(t)$, along the viewing ray. (c) The dynamic level-of-detail algorithm samples (here at 50%) and resizes (grey) the original (black) representation of the elements, resulting in a faster but coarser approximation, $\bar{g}_2(t)$, along the viewing ray.

over-representation of a tetrahedron is thus mitigated by compositing fragment distances instead of applying preassigned values for alpha.

3.1.1 Preprocessing the Tetrahedral Mesh

To create a point-based representation of the tetrahedra, a small preprocessing step is required. As will be shown, each tetrahedron can be approximated with exactly one affine transformation from a unit tetrahedron defined on the standard 3D basis to any given tetrahedron. The reference tetrahedron's barycenter is also transformed by the same affine transformation to a vertex, v, describing the location of the point object representing the associated tetrahedron.

Since an entire tetrahedron is represented by a single point, this point must be assigned a value based on the scalar field being described by the underlying geometry. For vertex-centered data, the mean of the scalar values at each vertex is assigned to v; otherwise, the cell value is assigned. While this method does not account for the distance of the vertices from the point representing them, in practice it adequately approximates a general, well-formed tetrahedron.

Because graphics point primitives (e.g., GL_POINT) are rasterized as squares they must be shaped to better approximate the tetrahedra they represent in order to improve image quality. The shaping is accomplished by *general representative transformations* for each tetrahedron. This additional information allows the GPU to reshape the point primitive in the fragment shader.

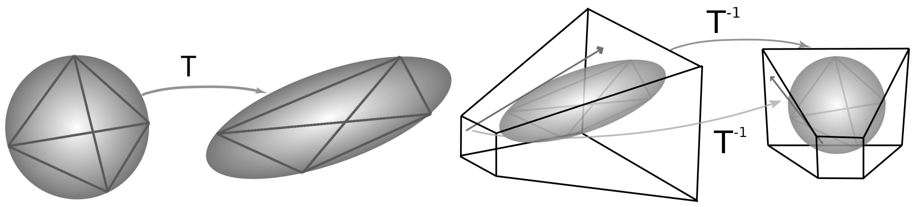
To generate this transformation, a regular tetrahedron is defined centered at the origin such that it is inscribed in the unit sphere (this regular tetrahedron is referred to as $\hat{\sigma}$). A transformation T is then found such that $T(\hat{\sigma}) = \sigma$. This transformation takes the unit tetrahedron to a given tetrahedron as shown in Figure 3.3.

3.1.2 Rendering the Points

As the method presented above produces an approximation of the true tetrahedral mesh, the approximation error must be reduced to create an adequate rendering. Here, various methods for footprint generation are discussed, and their results are shown Figure 3.4. Each footprint shows a trade-off between final image quality and rendering speed. In all of the following footprint generation techniques, point sizes are generated by taking the size of the tetrahedron and the distance from the viewpoint into account.

3.1.2.1 Circular Footprints

Generating footprints without regard to the shape of the tetrahedron can produce a high degree of error resulting from overdraw. Although the method by which point size is generated is unchanged from that of the unshaped points, to reshape the point into a circle the fragment program must be informed of the radius of the circle to be formed and its location in screen-space. These data must be sent down the pipeline, as fragments being processed by the fragment shader no longer contain information about the generating primitive. By passing the center of the point and radius to the fragment program via texture coordinates, the program is made aware of all data required to properly rasterize the tetrahedron approximation. Culling fragments that fall outside the radius of the circle is straightforward.



(a) (b)

Figure 3.3. (a) The transformation taking a unit tetrahedron to a general tetrahedron also takes a unit sphere to the min-volume ellipsoid. (b) Ray-casting on the GPU is efficient when taking advantage of the representative transformation.

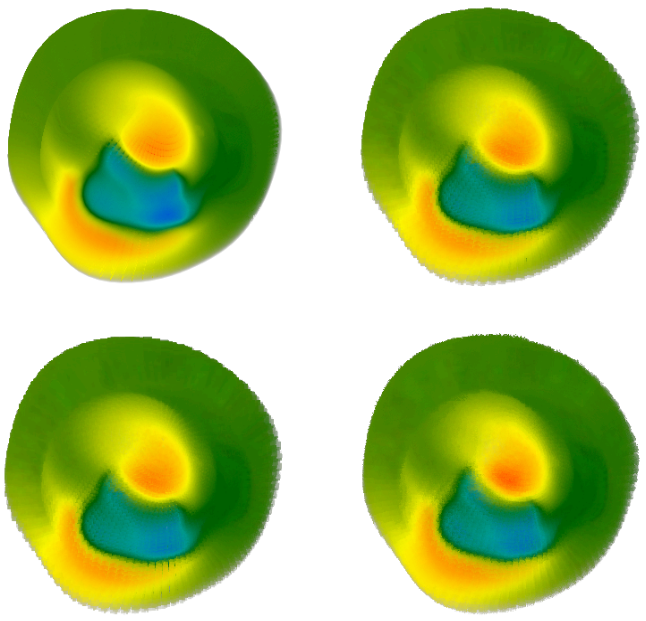


Figure 3.4. Rendering of the Heart dataset with HAVS [22] (top left) and with this algorithm using circular footprints (top right), ellipsoidal footprints (bottom left); and exact projections (bottom right). Note: This dataset has malformed tetrahedra that cause HAVS to produce erroneous renderings; the point-based method does not fail in these situations.

Figure 3.5 shows the image quality improvement by representing tetrahedra as circles compared to simple squares, but even reshaping the points to form circles renders many fragments unnecessarily.

3.1.2.2 Ellipsoidal Footprints

To reduce overdraw and increase image quality, ellipsoidal footprints are useful. Their smooth appearance makes them good candidates for point-based level-of-detail (see Section 3.2). From geometric probability, it is known that for a convex volume A contained in another convex volume B, the conditional probability that a random ray that hits B will also hit A is the ratio of surface areas, s_A and s_B: $p(A|B) = s_A/s_B$ [106]. Since s_A is the surface area of the tetrahedron, and hence fixed, the surface area s_B should be minimized to maximize the probability. This defines the best approximating ellipsoid over all viewpoints.

Finding the ellipsoid with minimum surface area that encloses the tetrahedron involves a constrained minimization over integrals with no closed form. Instead, the enclosing ellipsoid

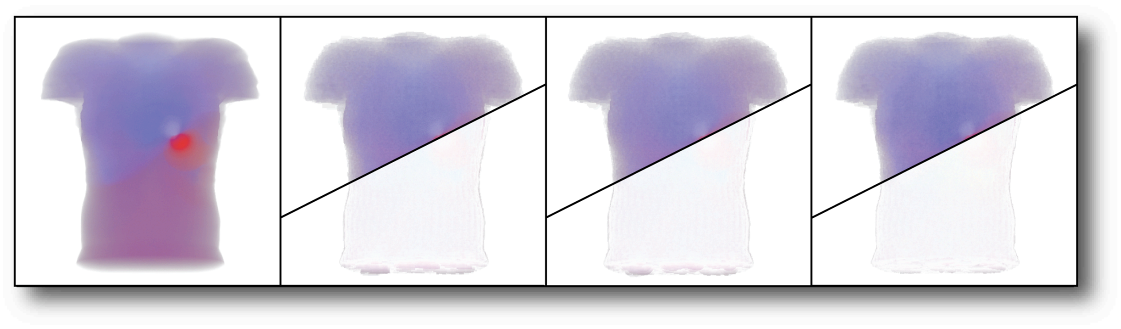


Figure 3.5. Hardware-accelerated point-based volume rendering is faster than the current state of the art approaches while still generating high quality images. Left to right: a baseline image is rendered at full quality and using points with unshaped footprints, circular footprints and ellipsoidal footprints, respectively. The lower half of the point-based images shows the difference to the baseline exact image. Notice the increasing level of fidelity.

with smallest volume can be found (called the *min-volume ellipsoid*). This is a simpler problem, and in practice, usually generates an ellipsoid with small surface area.

To find the min-volume ellipsoid, it is first noted that the unit sphere is the min-volume ellipsoid for the unit tetrahedron $\hat{\sigma}$. The transformation associated with the tetrahedron in question moves the min-volume ellipsoid for the unit tetrahedron to an ellipsoid for the represented tetrahedron. The volume of any shape is scaled by the determinant of T. Since this is a constant for all possible shapes, it must be the case that T preserves extrema, and so the ellipsoid is min-volume (see Figure 3.3).

The transform, along with the position of the point in $\mathbb{R}^3$ and its associated scalar value must be transferred to the GPU to properly cull fragments from the rasterized point. A simple raycaster provides a perspective-correct method for determining the projection of an ellipsoid. Since the reference space used during ray-casting is isomorphic to worldspace via the transformation T, transforming the rays allows the fragments to be culled on the GPU, based on a simple ray-sphere intersection. Figure 3.3 describes the results of this culling procedure from a rasterized square into the projection of a min-volume ellipsoid.

3.1.2.3 Exact Projection Footprints

As with ellipsoidal projection, the transform taking a unit sphere to an enclosing ellipsoid is equivalent to that describing the underlying tetrahedra. Similarly, an appropriate ray-casting test can be used in the fragment program to cull fragments leaving only the exact tetrahedral projection to be rendered.

Several opportunities for optimization are inherent in the underlying algorithm used to describe the geometry being rendered. First, the method by which the transformation is formed allows the tetrahedron being processed to be expressed as a right-angled tetrahedron. When expressed as a right-angled tetrahedron, the ray-caster needs only to test rays against three congruent right triangles in order to properly shape the tetrahedral projection.

3.1.3 Compositing the Points

As this method for footprint generation does not require any kernel-based convolution, it can be implemented independently of the compositing technique. This provides a true point-based volume rendering approach that can be easily implemented for many different compositing algorithms. In this case, a compositing scheme based on HAVS [22] is used, in which the incoming fragment is directly composited with the previous fragment. This is possible as the previous fragment is always stored in a texture so that the ray-gap can be determined. The scalar values of the front and back fragments as well as the distance between them are used to look up into a preintegrated table [41], then composited directly into an off-screen framebuffer. Since the initial drawing primitives can be thought of as screen-aligned polygons, there is no need to perform per-fragment depth sorting as with HAVS. A single fragment and its associated scalar and depth from the compositing step previously performed is the only information required at each step. However, as with any volume rendering algorithm depth ordering is important. Fortunately, as point primitives are rasterized as screen-aligned squares, a sort by centroid method exactly sorts all approximating elements alleviating the need for per-fragment sorting operations.

3.2 Level Of Detail

One of the advantages of using a point-based approach for rendering is that the lack of connectivity between primitives facilitates dynamically changing data. In particular, level-of-detail (LOD) approaches become much easier. Recently, Callahan et al. [21] introduced a simple, dynamically-adjusting LOD algorithm based on a sample-based simplification of the renderable primitives. Instead of simplifying the geometry, the algorithm performs importance sampling of the primitives, which enables dynamic LOD by adjusting the number of primitives rendered at each frame. To reduce holes in the resulting image, the boundary geometry of the mesh is always rendered.

This strategy is adapted in this point-based volume renderer by using a purely Monte Carlo approach to importance sampling. As before, the number of points rendered at each frame is adjusted based on the framerate of the previous frame. A key difference is that by using points as described, holes may appear in the volume with a lower LOD because the points may no longer overlap. To minimize these discontinuities in the resulting image, the ellipsoidal footprint is used. Because this problem will occur even on the boundary, there is no distinction between boundary and internal points. Instead, a technique is used for resizing the points based on the LOD to reduce the number of gaps created. Cook et al. [29] demonstrated this technique using a scale that is linear to the reduction for opaque geometry in a distant scene.

Since this volume renderer relies heavily on fragment processing, the LOD algorithm used corresponds to the number of fragments, instead of primitives as in the previous work[21]. Thus, a 50% LOD rasterizes 50% of the fragments. The problem is to find the correct number of points to render that generates the correct number of fragments with the scaling factor. If the number of fragments rasterized is represented as a function f of the LOD: $f(\lambda) = N(\lambda)S(\lambda)x^2$, where λ is the LOD reduction, N is the a function of the number of points, S is a function of the point scaling, and x is the point size. With no scaling, the number of fragments generated is directly proportional to the number of points rendered, i.e., $N(\lambda) = \frac{n}{\lambda}x^2$, where n is the number of points. Introducing a linear scaling factor, the number of points can be computed to obtain the same fragment count: $N(\lambda)\lambda x^2 = \frac{n}{\lambda}x^2$ or $N(\lambda) = \frac{n}{\lambda^2}$. Therefore, at each LOD step, the number of points rendered is adjusted by the square of the LOD, where the LOD $\in [0,1]$.

The resulting LOD scheme efficiently and dynamically adapts the number of fragments rendered to achieve a target interactivity. In practice, image quality is not dominated by the ellipsoidal representation of tetrahedra because the distance is composited directly instead of through the use of convolution kernels. Figure 3.6 shows an example of the described dynamic LOD with point resizing. It is important to note that although fewer points are selected for rendering, the sizes of the points being rasterized increases, thus increasing the number of points approximating the scalar field as illustrated by Figure 3.2. Although the point primitives are reshaped during rendering, the fragments being culled must still be processed. Each fragment, regardless of its final contribution to the framebuffer, is processed by the fragment program. Because of the fragment processing, during application of the LOD scheme, the relative size of the points selected for rendering directly impacts the performance of the selected LOD. This LOD strategy exploits the fact that as the total fragment count decreases, the framerate increases.

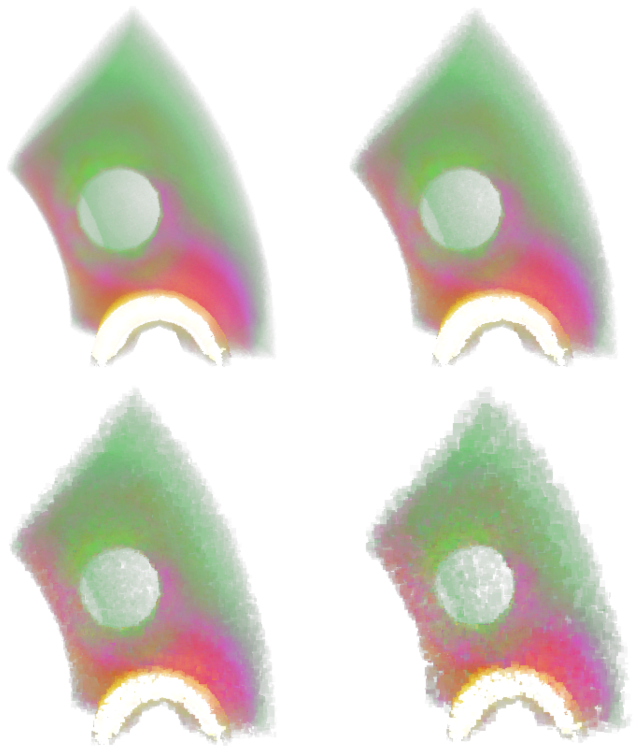


Figure 3.6. Dynamic level-of-detail of the SPX2 dataset (∼800K tetrahedra) with point resizing. The LOD at each step represents a percentage of the fragment count used: 100% at 5.6 fps (top left), 50% at 20 fps (top right), 25% at 50 fps (bottom left), and 15% at 100 fps (bottom right).

3.3 Results

Here the results are presented for the proposed algorithm in terms of visual quality and interactivity as they relate to HAVS [56]. To adequately analyze the performance of the volume rendering approach HAVS is used with comparable optimizations. Each rendering method employs Vertex Buffer Objects and parallel sorting and rendering. To generate images with the highest possible quality in HAVS, the largest size for the k-buffer is used to remove as many artifacts as possible. All results and images were generated on a desktop machine with Intel Core 2 Duo 2.2 GHz processors, 2.0 GB RAM, and an NVIDIA 7950 GX2 graphics card.

Table 3.1 describes the performance of the point-based method based on the shape of the footprints being generated compared with the HAVS algorithm. As the table describes, this method's performance is better than the HAVS algorithm for all datasets while yielding acceptable image quality. However, the presented algorithm approaches the performance of HAVS for large meshes. This is due to the quantity of data being sent to the GPU for further processing.

Table 3.1. Performance summary of point-based volume rendering for full-quality renderings in frames-per-second.

Dataset	Num Tets	Unshaped	Circles	Ellipses	HAVS
SPX2	827,904	13.8	14.7	17.2	5.26
Torso	1,082,723	6.7	7.1	6.8	3.70
Fighter	1,403,504	5.3	5.1	4.9	2.78
F-16	6,345,709	0.3	0.28	0.29	0.2

Although each of the increasingly accurate footprint generation methods reduces the amount of fragment overdraw that occurs at each step, more data must be sent to the GPU, reducing the overall rendering speed of particularly large datasets. As tetrahedra approach subpixel accuracy, it becomes unnecessary to reshape the points representing them as unshaped points at that scale represent tetrahedra just as well as shaped points, generating additional speedups. The degree to which these small tetrahedra affect the final visualization depends on the structure found in the original tetrahedral mesh. In Figure 3.7 the fighter dataset is rendered so that all tetrahedra with subpixel sized footprints are colored green. The reshaping of these tetrahedra will do very little to mitigate any rendering error in the final visualization, so they remain represented with squares.

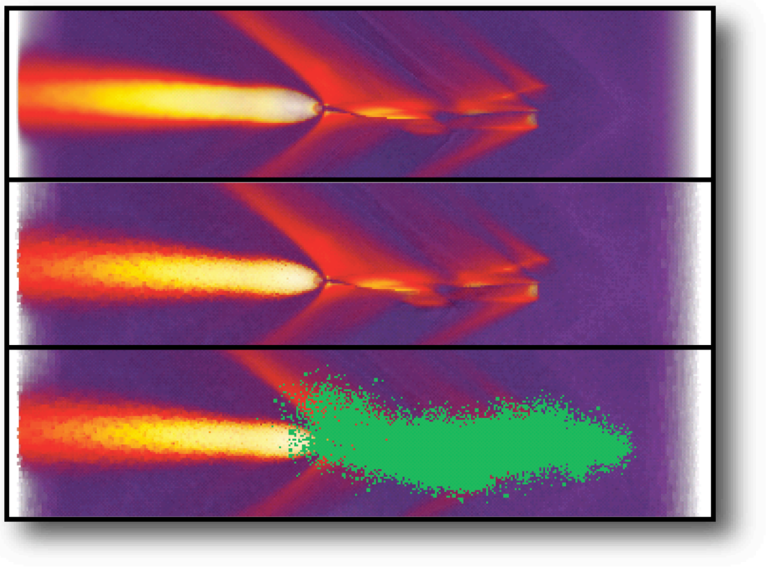


Figure 3.7. For adaptive meshes, many elements are subpixel sized. The top image shows HAVS [22], the middle image shows ellipsoidal projection of the tetrahedra, and the bottom image uses green pixels to show subpixel element locations.

3.3.1 Image Quality

To quantitatively evaluate the quality of a resulting image a quality metric must be described. The root-mean squared error metric is used here to analyze each rendered image. The implementation of this metric represents the mean of all distances in colorspace between corresponding pixels. This provides a global metric by which images generated by this point-based method can be compared to an image generated by exact volume rendering. Table 3.2 presents the results of the application of the above described quality metric. As the error metric implies, lower numbers correspond to higher quality images.

3.4 Discussion

The method presented here is generally faster than current direct volume rendering approaches; however, it is clear that some meshes are better represented by certain footprint shapes than others. Due to the number of fragments generated by the GL_POINT primitive, directly rendering the tetrahedral mesh may be a better approach than approximating the projections of the tetrahedra. This is true for datasets in which the tetrahedra sizes vary widely as the size of points representing large tetrahedra increase more rapidly than the sizes for small tetrahedra thereby increasing the overall number of fragments unnecessarily generated. Although many of these fragments may be culled in the fragment program, the testing and branching in the fragment program is an expensive operation. Additionally, only the exact projection approach does not render a large number of fragments that lay outside of the true projection of the tetrahedron that must be composited to

Table 3.2. Results of point-based volume rendering. (a) Image quality in Root Mean Squared (RMS) error for the Fighter dataset at different levels of detail. (b) Respective rendering performance in frames-per-second.

(a) Image quality

Level of Detail	Unshaped	Circles	Ellipses	Exact
100	6.51	6.12	3.89	3.74
75	6.97	6.46	4.72	3.23
50	8.54	7.77	6.80	3.49
25	12.22	10.62	13.60	14.15

(b) Performance

Level of Detail	Unshaped	Circles	Ellipses	Exact
100	1.7	1.7	2.0	2.1
75	3.1	3.1	3.2	3.2
50	6.4	6.6	6.7	6.7
25	25.3	25.2	25.5	25.2

form the final rendering. These extra fragments are culled in the point reshaping process.

As Figures 3.4 and 3.8 suggest, the point based rendering method presented here may yield a poorer quality rendering of a dataset when compared with a direct rendering approach. This is particularly noticeable at the dataset boundaries, as highlighted by Figure 3.8. This is caused by three error sources when all the points are rendered. First, depending on the point-based representation of the cell (e.g., ellipsoids), there may be overdraw noticeable at the boundaries. Second, the point primitives are screen-aligned and placed at the centroids of the cell, instead of at the cell boundaries, and thus there is a small error associated with the positioning and orientation (see Figure 3.2). Finally, for vertex-centered data, the scalar value is constant throughout the point instead of interpolated from the vertices as it should be. These errors are more noticeable where the depth complexity of the volume is low.

Although this approach to point-based rendering suffers from problems related to fragment generation and approximation of the tetrahedral mesh, it poses several unique advantages over previous methods. Since the footprint generation and reshaping are completely independent of the compositing and rendering stage, it is trivial to implement various compositing mecha-

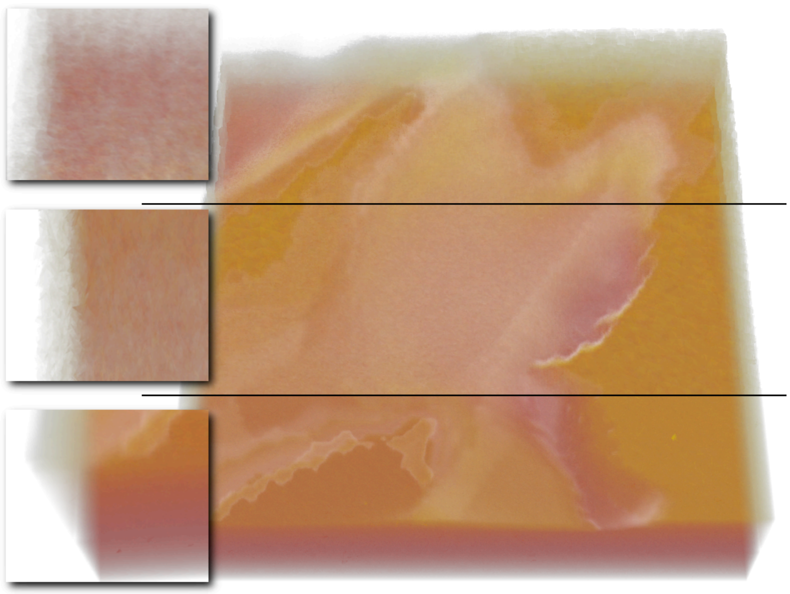


Figure 3.8. The errors associated with point-based rendering. Although interior areas of the volume are rendered with a high degree of accuracy, volume borders are more obviously affected by the point-based technique being used. An earthquake simulation (∼3.5M tetrahedra) displays these errors clearly. Ellipsoidal footprints (top) display more error than exact footprints (middle) when compared with the exact rendering (bottom).

nisms. Also, as has been shown, a global level-of-detail strategy based on importance sampling, generated as a preprocess, combined with point resizing, enables this method to excel during interaction. Furthermore, if lighting and shading are desirable for the final rendering, passing normal information gathered from gradient calculation in a preprocessing step can easily be used to implement a variety of shading models.

3.5 Summary

This chapter described a new method for point-based volume rendering of unstructured tetrahedral meshes that has proven to be a fast an effective method for visualizing large datasets. As with any visualization method, there is a trade-off between speed and accuracy. As the complexity of the approximations increases, so does the accuracy of the final rendering. However, this increased accuracy necessarily comes with a decrease in performance of the technique. The proposed method achieves interactive framerates through a combination of point-based footprint generation and a natural level-of-detail strategy that compromises accuracy for speed during periods of interaction while rendering at full-quality after interaction stops.

CHAPTER 4

IMAGE-SPACE ACCELERATION FOR VOLUME RENDERING

Acceleration techniques that approximate full quality images are common to provide interactivity with volumes too large or complex to handle otherwise. The general idea is to switch to a reduced representation of the rendering during interaction, but still allow a full quality representation to be rendered if desired. Approximation strategies for both structured and unstructured volumes fall into two categories: those that operate in object-space and those that operate in image-space. Whereas object-space methods involve simplifying or downsampling the volume to reduce the amount of data rendered (as described in Chapter 3), image-space methods usually involve reducing the number of pixels that are rendered. The result is a fast approximation to the full-quality image that contains either low frequency error, such as blurring, or high frequency error, such as "jaggies" caused by aliasing, from object-space and image-space methods, respectively.

This chapter introduces an image-space approach that downsamples for efficient rendering then upsamples for display using a joint bilateral filter to remove aliasing artifacts while still preserving sharp features. The upsampling algorithm is a postprocess that can be used independently or in combination with existing object-space and image-space acceleration approaches with very little computation or implementation overhead.

The bilateral filter [129] was first introduced as a method for denoising images and works by combining a linear kernel, such as a Gaussian, with a nonlinear, feature preserving term that weights the pixels based on intensities. The introduction of a separate reference image for performing the feature preservation is useful in some cases and is termed joint (or cross) bilateral filtering [103, 39]. This has recently been shown to be useful for enhancing images with solutions computed over downsampled images [65]. This work builds on this latter approach to improve volume rendering performance by rendering normally into a downsampled image and combining that with a low-cost reference image computed at full size. Instead of pixel intensities, the reference image contains depth information that can be used to encode the shape of the volume in a full size image. The result is an image that preserves the color of the downsampled image with the sharp features of the reference image. For opaque renderings, this has the appearance of smoothing the geometry in object-space, though it happens entirely in image-space. Figure 4.1 shows an example of the effect of this algorithm applied to an opaque rendering of a triangle mesh.

The main contributions of this chapter are as follows:

- A simple and fast image-space acceleration algorithm is introduced that is based on joint bilateral upsampling and results in substantial improvements for virtually any fragment or pixel bound volume renderer for unstructured grids;
- A method for quickly capturing reference images of the volume is proposed that is used to improve the quality of the technique over traditional approaches;
- A description is given of how the algorithm can be performed as a postprocess on the latest graphics hardware with very little overhead
- Quality and timing results are provided for the image-space acceleration algorithm on a variety of datasets using several volume rendering algorithms.

The rest of the chapter is organized as follows. In Section 4.1 the acceleration based on joint bilateral upsampling is described, along with how it can be applied to volume rendering and other

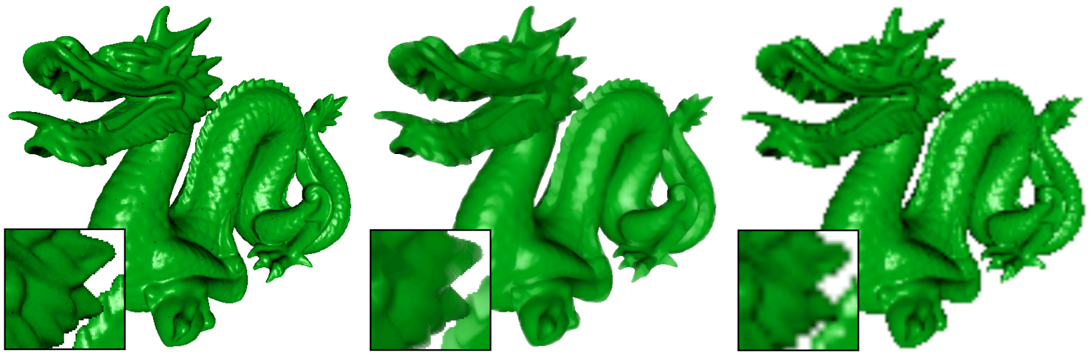

(a) Original (b) Bilateral (c) Linear

Figure 4.1. A comparison of joint bilateral upsampling with linear upsampling for an opaque mesh. The dragon dataset rendered (a) normally at full opacity at 512^2, (b) upsampled from a 128^2 rendering using joint bilateral filter that combines a low resolution color buffer with a high resolution depth buffer, and (c) upsampled linearly from a 128^2 rendering. Joint bilateral upsampling is an acceleration method that can be applied to volumes to remove unwanted aliasing while still preserving sharp features.

implementation details. In Section 4.2, results of the algorithm are provided and in Section 4.3 the trade-offs of its use are discussed. Finally, the work is summarized in Section 4.4.

4.1 The Algorithm

The proposed acceleration algorithm is briefly summarized by the following steps:

1. Render the volume into a small offscreen image *I* using an existing volume rendering algorithm.
2. Render the boundary geometry of the volume at full size and capture the depths of the fragments in a reference image *R*.
3. Upsample the offscreen image *I* to full size using texturing hardware and combine it with the reference image *R* using the joint bilateral filter.

Figure 4.2 shows the visual effect of this algorithm on a volume. The details of each step in the method are described in the remainder of this section.

4.1.1 The Joint Bilateral Upsampling Filter

The original bilateral filter uses both a domain (spatial) and a range filter kernel on the input image to produce a denoised output image. For some position p, the filtered result is:

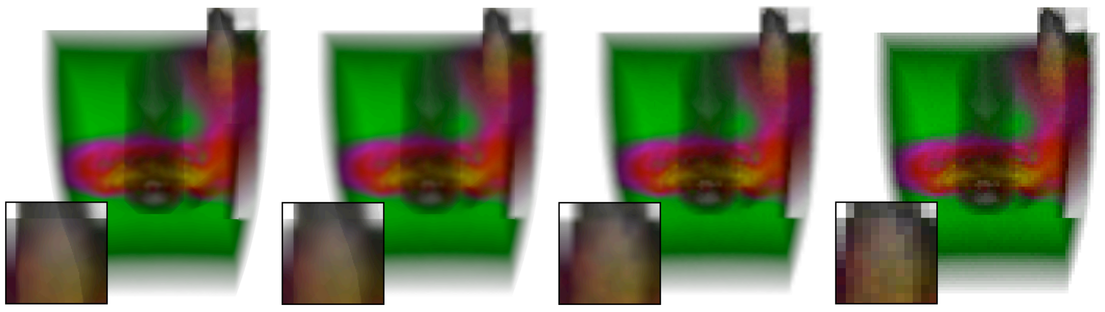
$$J_p = \frac{1}{k_p} \sum_{q \in \Omega} I_q f(\| p - q \|) g(\| I_p - I_q \|), \tag{4.1}$$

where f is the spatial filter, such as a low pass filter that operates on pixel colors centered over p, and g is the range filter kernel, such as a low pass filter that operates on pixel intensities centered over p. Ω is the spatial support of the kernels f and g, and k_p is the normalization computed as the sum of the f and g filter weights. Intuitively, $f \cdot g$ is just a new filter kernel that changes per pixel to respect intensity boundaries.

The joint bilateral upsampling filter uses separate images at different resolutions for the domain and range to compute an upsampled solution S from a given high resolution image I and a low resolution solution R that is used as a reference image:

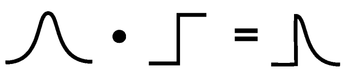
$$S_p = \frac{1}{k_p} \sum_{q_\downarrow \in \Omega} R_{q_\downarrow} f(\| p_\downarrow - q_\downarrow \|) g(\| I_p - I_q \|), \tag{4.2}$$

where p and q denote coordinates in I, and $p_\downarrow$ and $q_\downarrow$ denote the corresponding coordinates in the low resolution reference image R. This formulation is used to compute costly solutions for



(a) Original (b) Bilateral (c) Linear (d) Nearest

Figure 4.2. A comparison of volume rendering accelerated with bilateral and other upsampling methods. The SPX dataset rendered using a software raycaster (a) normally at a 1024^2 resolution at one frame per second, and upsampled from a 128^2 image at 10 frames per second using (b) the proposed feature preserving joint bilateral upsampling, (c) linear interpolation, and (d) nearest neighbor interpolation (similar to a method that casts one ray per 8^2 pixel grid). Only the original and the bilateral method preserve the diagonal edge that appears in the center of the inset images.



high resolution images at lower resolutions. The algorithm proposed here is different, as an inexpensive solution R at high resolution is used to upsample a low resolution image I. In the same notation, this filter could be expressed as:

$$S_p = \frac{1}{k_p} \sum_{q_\downarrow \in \Omega} I_{q_\downarrow} f(\| p_\downarrow - q_\downarrow \|) g(\| R_p - R_q \|), \tag{4.3}$$

where p and q denote coordinates in the high resolution reference image R, and $p_\downarrow$ and $q_\downarrow$ denote the corresponding coordinates in the low resolution image I.

4.1.2 Computing the Reference Image

To preserve features in the upsampled version, a full resolution reference image is needed for the range component of the bilateral filter. This reference image needs to be fast to compute and general enough to apply to a variety of volume renderers. For unstructured grids, it is common to represent the domain (or boundaries) of the volume to facilitate the understanding of the features that are contained therein. Fortunately, the boundaries are easy to capture—they are often already used by volume rendering algorithms as starting points for ray traversal [15, 138] or as the base case for object-space LOD [21]. Other sharp boundaries within the volume could also be used as well, such as those provided by isosurfaces, if they are readily available.

For bilateral upsampling to faithfully preserve the features of the volume's domain, more than just the front-most boundary needs to be captured. Multiple depth layers are already used by many volume renderers to handle nonconvex meshes either in software [15] by creating a sorted depth list for each pixel, or in hardware [9, 140] using depth peeling [44]. Depth peeling is a multipass algorithm that captures one layer of depth on each pass, starting with the nearest fragment per pixel, then the second nearest, third nearest, and so on. This can be performed efficiently in hardware by rendering the first pass normally, resulting in a depth buffer of the nearest surface. In subsequent passes, the depth buffer computed in the previous pass is used to *peel* away depths less than or equal to those already captured in previous passes. These depth peeling passes can even be reduced to a single pass using stencil routing [5].

To compute the reference image in an existing volume rendering algorithm, the framework already in place is leveraged for capturing depths whenever possible. If depths are not already captured, a depth peeling pass is simply added to the renderer. The number of depth passes that are used is dependent on the volume being rendered and the opacity at which it is being rendered. Generally, two or three layers are sufficient for most datasets to capture the visible boundary features.

4.1.3 Implementation

The joint bilateral upsampling is implemented with minimal changes to an existing algorithm. The low resolution image I and the reference image R are rendered offscreen. Then in a final pass, a full resolution, screen-aligned quadrilateral is drawn that binds both images as textures and uses a fragment shader to perform the joint bilateral filter. If the texturing hardware is set to linearly interpolate I, the small resolution image will be upsampled to full resolution linearly in the shader, improving the quality of the upsampling by adding an inexpensive low pass filter. In the shader, the I and R textures are accessed using the same coordinates to retrieve color and depth information used in the joint bilateral filter.

For each pixel p in the final image, the joint bilateral filter is a low pass filter that blurs a fixed neighborhood around p to remove noise. Choosing the spatial support Ω for the filter should be based on the amount of upsampling that is being performed on I, i.e., more blurring is required for higher upsampling factors. The spatial support for the joint bilateral filter is matched with the spatial support for the linear interpolation performed by texturing hardware: if upsampling to 1024^2, a 512^2 image will use $\Omega = 4$, a 256^2 image will use a $\Omega = 8$, etc.

For domain and range filters, f and g, Gaussian low pass filters are used:

$$f(x,y) = g(x,y) = e^{-D(x,y)^2/2\sigma^2}. \tag{4.4}$$

For the domain filter f operating on the low resolution image I, $D(x,y)$ is the distance between (x,y) and the origin of the filter p, and σ is the spread of the Gaussian, or $\Omega/2$. For the range filter g operating on the reference image R, $D(x,y)$ is the difference between the depth value at (x,y) and the depth value at p. For multiple depth layers, this simply becomes the distance between the vectors defined at (x,y) and p. The range σ is the value that expresses the resolution of the depth features that should be preserved. Thus, the range σ is dependent on the resolution of the depth buffer and should be as low as possible to capture depth changes, without causing artifacts due to depth precision. A $\sigma = 0.01$ for the range is generally adequate in practice.

4.2 Results

To demonstrate the flexibility of the joint bilateral upsampling algorithm for unstructured grids, it was added to existing source code for three popular algorithms and the numbers are reported for the speed and quality of the technique.

4.2.1 Timing Results

The fragment shader for bilateral sampling itself is relatively inexpensive, for a 1024^2 image, kernel sizes of 4, 8, and 16 achieve framerates of 100 fps, 50 fps, and 15 fps, respectively, on a Quadro FX 5600 graphics card. For timing experiments, the datasets shown in Table 4.1 are used to gather statistics for upsampling from 128^2, 256^2, and 512^2 to 1024^2 and compare them with the original rendering times for a 1024^2 image. At each resolution, the dataset is rendered from 14 viewpoints, defined by the corners and faces of a cube around the dataset, and the times are averaged. To make comparisons between upsampling factors easier, a uniform kernel size of 12 (at 25 fps) is used for all resolutions. All of the results were rendered on a machine with 2 Dual Opteron 2.25 GHz processors, 4 GB RAM, and an NVIDIA Quadro FX 5600 graphics card with 1.5 GB RAM.

Table 4.1. Experimental datasets used for measuring rendering performance, with vertex and tetrahedron coount.

Dataset	Vertices	Tetrahedra
SPX	3 K	13 K
Blunt Fin	41 K	187 K
F117	49 K	240 K
SPX2	166 K	828 K

The first algorithm that was modified is a software raycaster from Bunyk et al. [15] that has freely available source code and runs completely on the CPU. The algorithm first rasterizes boundary triangles to capture starting and ending points for the rays at each pixel. It then marches rays through the volume cell to cell by exploiting connectivity of cell faces. To make the modification, the ray casting is performed as normal, except into a small image. Then, the existing depth capturing code is used to find the boundary depths in a large image. These two images are then bound as textures and rendered to the screen using a fragment shader written in OpenGL. Figure 4.3 shows a series of plots for several datasets comparing times (logarithmically scaled) for the varying resolutions. In the experiments, the acceleration for these datasets ranges from about 16 times to 28 times for 128^2 resolution images upsampled to 1024^2.

The second algorithm that was modified is a hardware-assisted raycasting algorithm from Bernardon et al. [9] that is freely available and is written in DirectX9. As with the software raycaster, the algorithm was adapted to render into a small offscreen buffer and use the existing

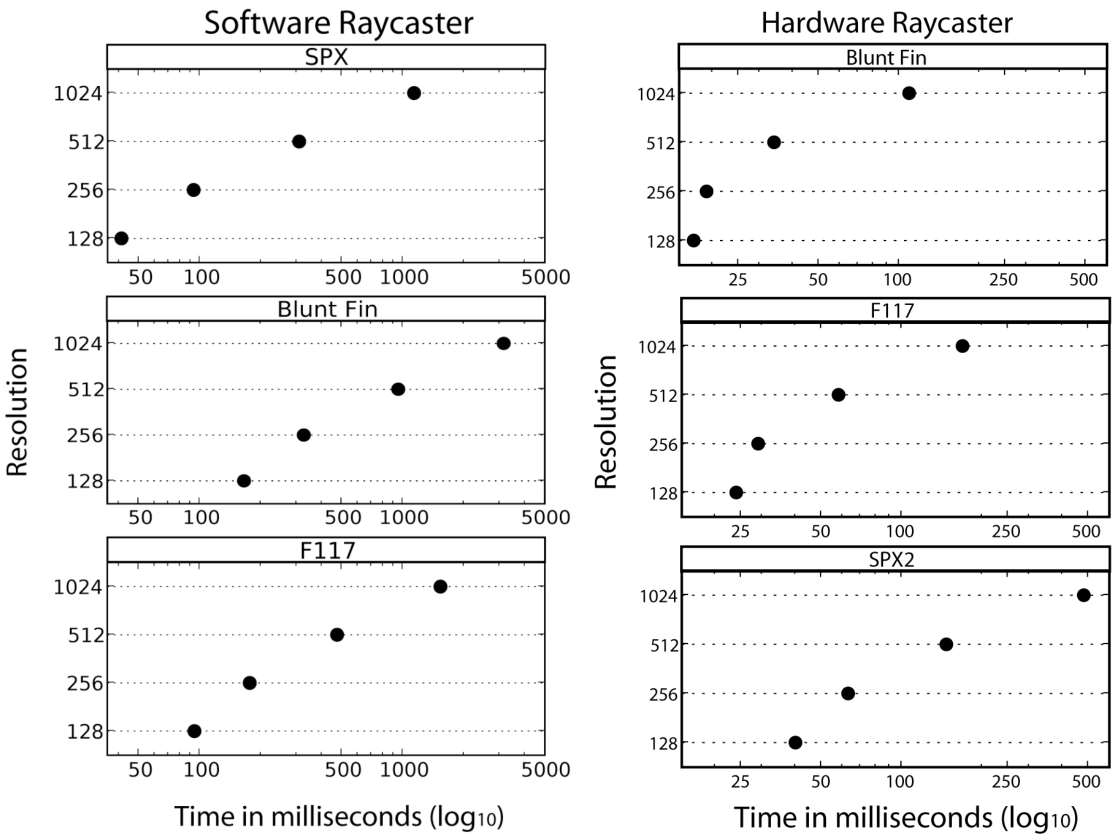


Figure 4.3. Timing statistics for a software raycaster and a hardware raycaster for various resolutions upsampling to 1024^2 using joint bilateral upsampling. The 1024 resolution represents the time for a full quality image without upsampling.

depth peeling routines to capture the depth in a full size offscreen buffer. An HLSL program was then used to perform the joint bilateral upsampling and display the final image. Figure 4.3 shows a series of plots comparing times (also logarithmically scaled) for varying resolutions. With the hardware raycaster, acceleration gains range from about 7 times for the smallest dataset to about 12 times for the largest for 128^2 resolution images upsampled to 1024^2.

4.2.2 Quality Results

By using a full size reference image, the joint bilateral upsampling is able to achieve better imagery than upsampling alone. This is shown both quantitatively and visually. Figure 4.4 shows rate distortion curves for the quality of upsampling using a joint bilateral filter and linear interpolation (as provide by texturing hardware). The measurements were computed using root mean squared error (RMSE) comparisons between full quality images at 1024^2 and images upsampled at various resolutions. In all cases, the bilateral upsampling exhibits less error than with linear interpolation alone. Figure 4.5 shows rendered solutions at various resolutions for a

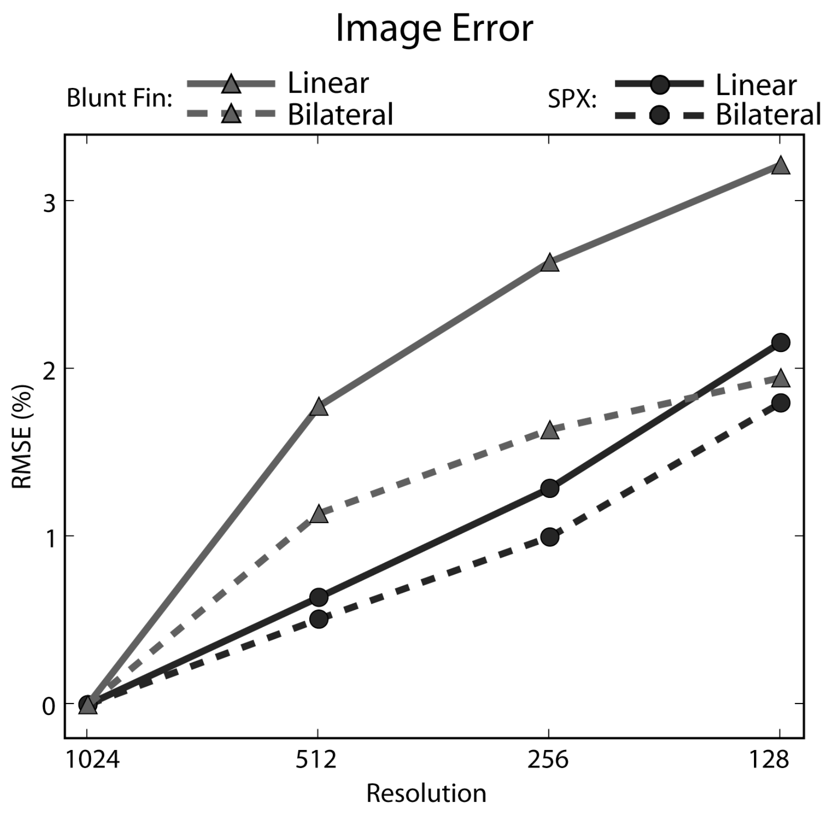


Figure 4.4. Quality comparisons of joint bilateral versus linear upsampling. Rate distortion curves for various resolutions compare RMS error of full quality images.

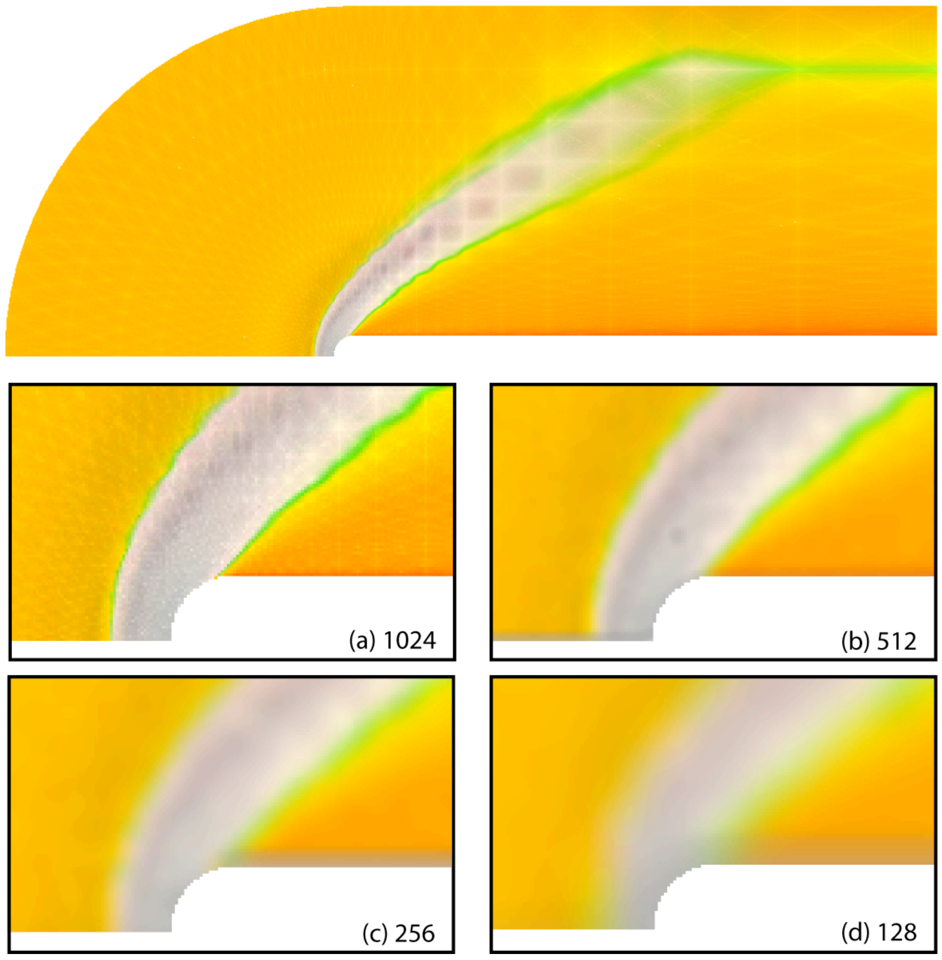


Figure 4.5. Quality comparisons for various resolutions of upsampling. The Blunt Fin dataset rendered into a 1024^2 image at (a) full quality and upsampled from (b) 512^2, (c) 256^2, and (d) 128^2 using the proposed technique.

visual comparison of the quality change.

One interesting application of the filter is in improving the appearance of existing acceleration techniques by denoising results while still preserving edges. The upsampling filter was added to the HAVS volume rendering algorithm [22] to improve the appearance of a dynamic LOD algorithm that operates by sampling the geometry of the volume [21]. HAVS sorts the triangles that compose the mesh first in object-space using a simple sorting routine, then in image-space by storing a fixed number of fragments. The LOD algorithm samples the triangles before the sorting, based on precomputed importances, to make the rendering more efficient. Because the LOD already uses boundary geometry as the base sampling case, an additional pass was easily added to render these boundaries into a reference image before combining it with the original rendering pass using joint bilateral upsampling.

Due to the nature of the algorithm, HAVS is more vertex bound than pixel bound. Thus, the acceleration when using the upsampling approach is negligible on the most recent graphics cards. However, by sparsely sampling the geometry in the mesh but leaving boundary geometry, the number of primitives rendered is reduced and the speed of the algorithm is improved. This sample-based simplification has the side effect of producing high frequency error in the reduced representation, unlike domain-based simplification techniques (i.e., simplification via edge collapses [49]). Using the joint bilateral filter on the resulting imagery, the noise was reduced and the overall appearance was improved for the LOD strategy with little effect on the performance. Figure 4.6 shows an example of this LOD before and after a joint bilateral filter is applied. Because the acceleration due to the LOD algorithm is dominant, the rendered image I does not need to be computed at a reduced representation and upsampling is not necessary.

4.3 Discussion

Because the proposed method is simple, it can easily be utilized as a technique to accelerate interaction, while still allowing full quality images to be rendered when the user stops interacting with the viewing parameters. Tools such as ParaView [99] use a similar strategy during rendering, by either changing the number of slices for texture based methods, or number of rays for raycasters. This algorithm could be used as a replacement or enhancement for these existing techniques because it produces better approximations of the full quality image with less visual artifacts. Because it is easy to change the speed/quality trade-off by adjusting the upsampling factor, the algorithm could also be used for dynamic level-of-detail.

Many existing acceleration techniques that trade-off speed for image quality create high frequency error in the resulting image in the form of stair-casing or aliasing artifacts. In contrast, the proposed method produces low frequency error, which results in more visually pleasing images that retain edges that are supposed to be in the image, while removing those that are not. Although joint bilateral upsampling is used, other upsampling strategies have been introduced [145] and could be used instead. However, without the additional shape information that is provided by the reference image, other upsampling strategies are not likely to perform as well as they can result in halos and other undesired artifacts, as shown by Kopf et al. [65].

The proposed solution for performing joint bilateral upsampling for volume rendering was implemented in OpenGL and DirectX using fragment programs. It was also implemented with NVIDIA's CUDA library, which is efficient for offscreen processing, but not as fast as fragment

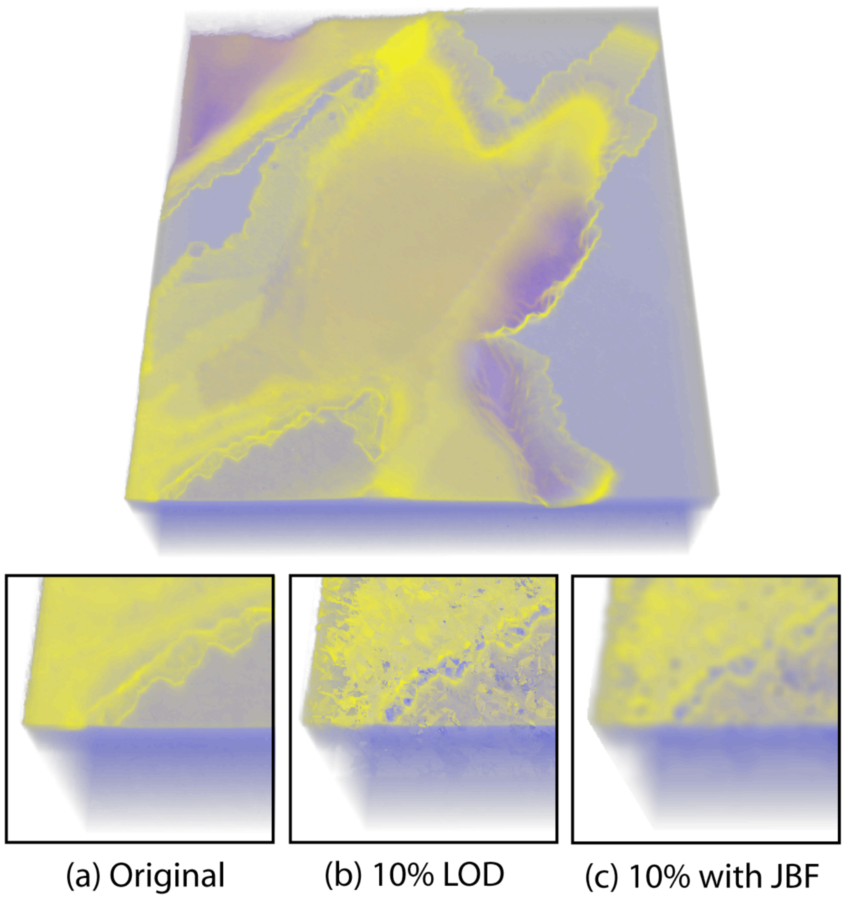


Figure 4.6. Removing high frequency noise from object-space acceleration. The San Fernando Earthquake dataset (1.4 million tetrahedra) is rendered using HAVS [22] with sample-based simplification [21]. (a) The full quality image is rendered at 1.7 fps compared with (b) sampling 10% of the geometry (20 fps) and (c) sampling 10% of the geometry then using the joint bilateral filter without upsampling to remove the high frequency error while still preserving boundary features (15 fps).

programs for interactive graphics. Even for large images and large filter domains, the computational cost of the algorithm is not high relative to the volume rendering cost. As with most acceleration techniques, the trade off for image quality is performance (i.e., more downsamping results in higher speed).

The experiments included several datasets at various sizes to demonstrate the acceleration that the proposed technique can provide. The size of datasets used in these experiments was limited by the volume rendering methods employed, not by any limitations of the acceleration technique. Volume rendering algorithms that handle larger datasets, such as raycasters that use bricking strategies for memory management [96] or point-based techniques that are fragment-bound [3], would also benefit from this acceleration technique.

4.4 Summary

In this chapter, an acceleration technique for unstructured grid volume rendering that operates in image-space was introduced. By rendering small images and upsampling them with a smart filter, performance improvements of up to 30 times have been measured. The upsampling strategy based on joint bilateral upsampling results in a high quality approximation that avoids the high frequency noise common in existing acceleration techniques based on rendering reduced representations of the data. The major advantages of this algorithm are that it is simple to implement, it is flexible enough to be included as a postprocess to virtually any direct volume rendering algorithm, and it can easily be used in combination with existing acceleration techniques.

CHAPTER 5

PROGRESSIVE VOLUME RENDERING

Specialized graphics clusters have been developed to visualize large datasets in parallel and on large displays. However, the availability of these clusters is often limited. More recently, many techniques have been developed to visualize volumetric data on commodity PCs using graphics hardware [138, 22]. This provides a solution that allows researchers to perform their visualizations locally when other resources are unavailable. However, due to the limitations of storage and memory with most desktop machines or laptops, this solution does not scale well for extremely large datasets.

As an example, consider a scientist working remotely who would like to visualize a large dataset on his laptop computer. A reduced representation of the data (e.g., simplification [49]) may not be appropriate if a high quality visualization is required for analysis. Complicating matters even further, the laptop may not have capacity on the hard disk or in memory to keep the dataset. The problem is compounded if you consider that the scientist may only want to browse through a series of datasets quickly, requiring the download of each dataset before visualization.

In this chapter, a client-server architecture is presented for hardware-assisted, progressive volume rendering. The main idea is to create an effect similar to progressive image transmission over the internet. A server acts as a data repository and a client (i.e., a laptop with programmable graphics hardware) acts as a renderer that accumulates the incoming geometry and displays it in a progressively improving manner (see Figure 5.1). This progressive strategy is unique because it only requires the storage of a few images on the client for the incremental refinement. For interactivity, a small portion of the mesh is stored on the client using a bounded amount of memory. Furthermore, the progressive representation provides a natural means for level-of-detail exploration of very large datasets without an explicit simplification step that may be difficult and costly. Because the geometry is rendered in steps, the user can stop a progression and change the view without penalty, thus facilitating exploration. The proposed algorithm is robust, memory efficient, and provides the ability to create and manage approximate and full quality volume renderings of unstructured grids too large to render interactively at full resolution.

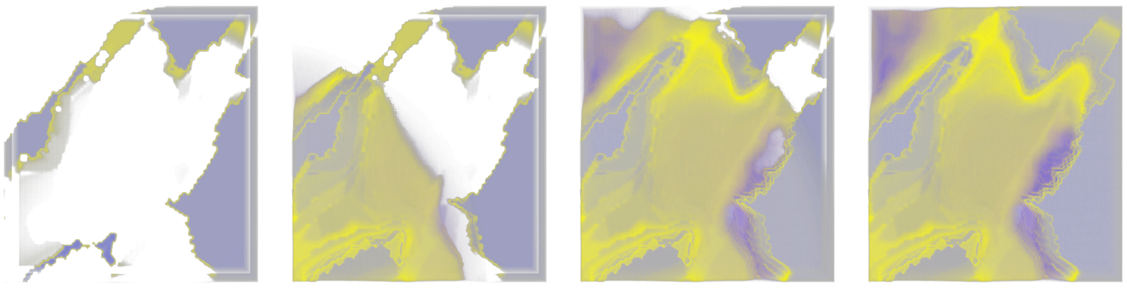


Figure 5.1. A sequence of progressive volume rendering steps for the SF1 dataset with about 14 million tetrahedra. Starting from an interactive mode that uses only the boundary (left), the algorithm progressively refines the image using incoming geometry as well as the results of the previous refinement until the full-quality rendering is achieved (right).

The contributions of this chapter include:

- A client-server architecture and interface for rendering large datasets and managing the resulting visualizations is introduced;
- A server is described that acts as a data repository by streaming a tetrahedral mesh in partial visibility order to one or more clients;
- A client is described that uses hardware-assisted, progressive volume rendering to provide an interactive approximation, progressive refinement, and full-quality rendering of large datasets;
- Experimental results are given for the algorithm and a discussion is included on the benefits and limitations of the approach.

The rest of the chapter is outlined as follows. Section 5.1 describes an overview of the client-server architecture. More detail is provided about the server in Section 5.2 and about the client in Section 5.3. In Section 5.4, experimental results are outlined, in Section 5.5 a discussion of the trade-offs of the algorithm is included, and in Section 5.6, a brief discussion of the algorithm and a summary are provided.

5.1 System Overview

The client-server system architecture is depicted in Figure 5.2. The server acts as a data repository and geometry processor. The data are stored on the server hierarchically for efficient traversal and iterative object-space sorting. The client keeps the boundary triangles for an *Interactive Mode* and requests geometry from the server in a *Progressive Mode*. In the *Progressive Mode*, the client uses hardware-assisted LOD volume rendering to refine the image using the

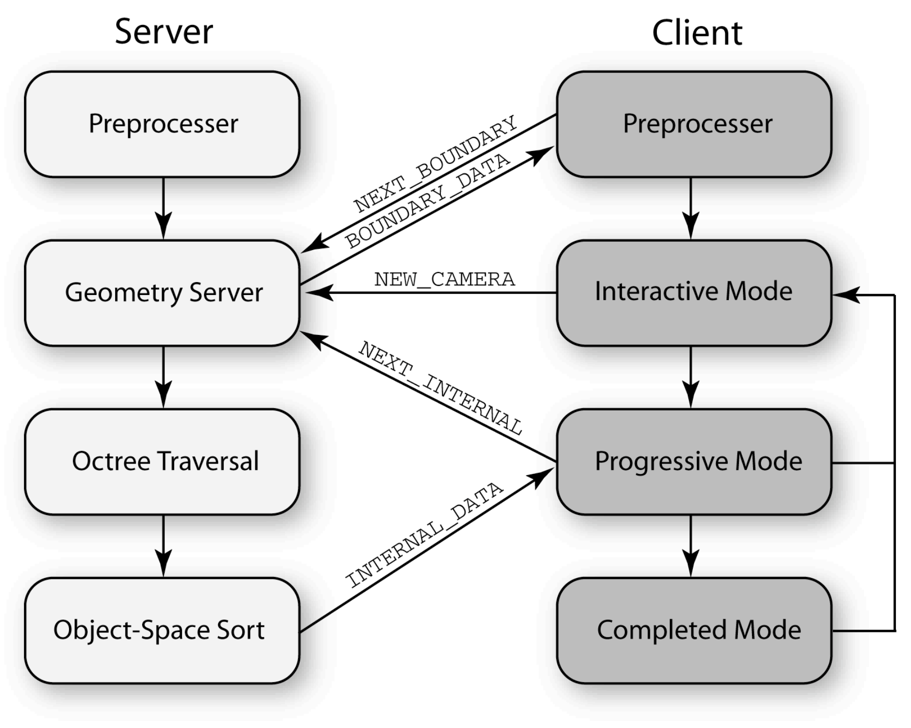


Figure 5.2. The client-server architecture. Communication between the client and the server is shown with annotated arrows.

results of the previous progressive step. Upon completion of the progressive volume rendering, the client saves a copy of the image for later browsing in a *Completed Mode*. Viewing changes in any of the client steps causes the progressive renderer to stop and return to the *Interactive Mode*.

5.2 The Server

5.2.1 Geometry Processing

The server acts as a data repository that sorts and streams triangles. However, sorting large datasets in one pass may be cumbersome. For the client to remain interactive, it should begin receiving nearly-sorted faces immediately from the server. Furthermore, the client should be able to interrupt the streaming of the faces at any time to keep the exploration interactive.

To keep the processing of the datasets to a minimum, a one-time preprocessing of the datasets is performed to extract the unique triangle faces and vertices and store them in a binary format that are read when starting the server. Then, the server is used to process the faces of the mesh into an octree structure that can be traversed by depth ranges, from front to back. For every packet of faces requested from the client, the server first culls faces outside the current depth range, sorts the remaining faces, and sends them to the client. Subsequent packets use incremented depth ranges.

This has the effect of distributing the sorting burden between each step of the progression. It also prevents unused geometry from being sorted.

For a given depth range, the associated geometry is culled using a depth-range octree. The octree is a geometric partition of the faces, according to their centroids. The depth-range octree is similar to the octree from Wilhelms et al. [143] for isosurface extraction. However, instead of using scalar values, the octree uses dynamically changing depth ranges to cull the geometry outside the current depth range. Each octree node contains an array of face indices. The depth range of a node is the minimum and maximum distance from the eye to the bounding box corners of the node. To find the faces matching a given depth range, the octree nodes are culled hierarchically by traversing nodes that may contain triangles in the given depth range. Next, the collection of triangles in the matching octree nodes are culled according to their distance from the viewpoint. The remaining triangles are then inserted into an array for sorting. A radix sort is utilized on the face centroids, as described by Callahan et al. [22]. The triangles are then sent as a vertex array for direct rendering on the client.

To increment the depth range on each pass, the range of the minimum and maximum distance from the eye to the bounding box of the mesh is uniformly divided. This has the unfortunate side effect that the number of triangles per slice can vary in size. This issue is addressed by collecting packets on the server and only transmitting them to the client when a user-specified target triangle count is reached. This results in good performance since the sorting and network transfers are more efficient for larger packets.

When interactively exploring regions of the mesh in detail, often many of the faces are outside of the view-frustum. These faces should not be transmitted to the client. Therefore, in addition to the depth range test, view-frustum culling is performed for each node of the octree. The left, right, bottom and top planes of the frustum are computed from the modelview-projection matrix sent by the client. This results in significant performance improvements for zoomed-in views of the dataset.

Network transfers become a bottleneck for client-server systems with high bandwidth. To reduce the bandwidth, the transmitted vertex arrays are compressed using the open-source zlib library [154] based on Huffman codes, which is fast and robust. The maximum level of compression for the client-server data transfers is used.

5.2.2 Network Protocol

The following description assumes a single client per server, and can be extended for multiple clients by storing the context of the stream on each client.

The server understands three types of commands: NEW_CAMERA, NEXT_BOUNDARY, and NEXT_INTERNAL. Each new client initializes the stream by sending a NEW_CAMERA command, containing a frame ID and the camera information. The frame ID is an integer initialized to 0 and incremented by the client for every NEW_CAMERA command. Upon connection, the client requests boundary faces from the server using the NEXT_BOUNDARY command. The server incrementally sends the boundary faces in a BOUNDARY_DATA packet consisting of an unsorted triangle soup (i.e., a sequence of vertex coordinates) for the client to use during interactive rendering. By sending the boundary in chunks, the client can request only the portion that it can retain in memory. If the client is limited and can only use a portion of the boundary during interaction, the NEXT_BOUNDARY command can be used at the beginning of the progressive rendering to fill in the missing boundaries before the internal faces are transmitted. To avoid unnecessary sorting when the camera is moving, the server does not sort the faces until it receives a NEXT_INTERNAL command from the client. The server culls the geometry by depth and frustum, sorts by centroid, and sends back a chunk of the interior faces in an INTERNAL_DATA packet.

A TCP/IP socket is used to transmit the data. Unlike UDP, TCP guarantees that the data packets sent from the server arrive in the same order they were sent and without error. Since the client can change the camera at any time, the state of the client and the server need to be synchronized. This is necessary to avoid issues when the camera moves and the client is receiving data asynchronously, or the server is processing data. All the packets from the client or from the server contain a frame ID. Before processing a packet, the client and the server check that the frame ID from the packet is the same as the current frame ID. Packets with obsolete frame IDs are ignored by the server and the client. In addition, when the client receives an obsolete packet, it resets the stream.

5.3 The Client

The main function of the client is as a progressive volume renderer. Because the client may be limited in disk space as well as memory (e.g., a laptop), the goal is to minimize the data stored on the machine at each progressive step. Therefore, the client acts as a *stream renderer*— it receives geometry transmitted from the server and renders it directly with the GPU. This requires a volume renderer capable of handling dynamic data in an efficient manner. In addition, the algorithm requires an approximation technique for partial geometry as well as a means of keeping previously computed information for subsequent refinement steps.

To leverage GPU efficiency, the Hardware-Assisted Visibility Sorting (HAVS) algorithm of Callahan et al. [22, 56] is extended to perform progressive volume rendering. The HAVS algorithm operates in both object-space and image-space to sort and composite the triangles that compose a tetrahedral mesh. In object-space, the triangles are sorted by their centroids using a linear-time radix sort for floating-point numbers. This provides a partial order for the triangles. Upon rasterization, the fragments are sorted again in image space using a fixed size A-buffer [24] implemented with programmable shaders called the *k*-buffer. For each pixel of the resulting image, *k* entries (scalar value *v* and depth *d*) are stored in textures on the GPU. An incoming fragment is compared to entries in the *k*-buffer to find the two closest to the current viewpoint (for front-to-back compositing). These entries are then used to look up the color and opacity for the volume gap in a preintegrated table by using the front scalar, back scalar, and distance between the entries. This color and opacity are then composited to the framebuffer, the front entry is discarded, and the remaining entries are written back into the *k*-buffer. The progressive renderer uses the HAVS algorithm as a basis for sorting and compositing. The server handles the object-space sorting, while the client handles the fixed-size image-space sorting and compositing.

The *k*-buffer is implemented in hardware using multiple render targets (MRTs), which allows the reading and writing of multiple off-screen textures in each pass. Currently, hardware limits the number of MRTs to 8, which limits the size of *k* to 14 (one texture for an off-screen framebuffer and seven textures for *k*-buffer entries). In OpenGL, the simplest access to multiple render targets is in the form of Framebuffer Objects (FBOs). An FBO is a collection of logical buffers such as color, depth, or stencil. Multiple color buffers (up to eight) can be *attached* to an FBO for off-screen rendering. FBOs make it possible (and efficient) to swap attached buffers between rendering passes. Currently, this is faster than switching between multiple FBOs. The ability to render into a subset of buffers in multiple passes is at the heart of the progressive algorithm.

The progressive volume rendering works by using five different render targets for each pass, represented as four channel, 32-bit floating-point textures. The textures are used as follows:

T_{pro}**:** An off-screen framebuffer for the progressive image (R_p, G_p, B_p, A_p).

T_{k12}**:** *k*-buffer entries 1 and 2 (v_1,d_1,v_2,d_2).

T_{k34}**:** *k*-buffer entries 3 and 4 (v_3,d_3,v_4,d_4).

T_{k56}**:** *k*-buffer entries 5 and 6 (v_5,d_5,v_6,d_6).

T_{approx}: A temporary framebuffer for the approximation of the portion of the mesh not yet received (R_a, G_a, B_a, A_a).

A combination of these textures is used for each step of the progressive volume rendering. The contents of all the textures except T_{approx} are reused in subsequent progressive passes.

The progressive volume rendering is separated into three modes of operation. *Interactive Mode* is used during camera events such as rotation, pan, or zoom. *Progressive Mode* is used when interaction stops to stream triangles from the server to the client in chunks. The *Progressive Mode* can be interrupted at any time if the user begins interaction again or the stream finishes. When a complete image is generated with the *Progressive Mode*, *Completed Mode* automatically stores the image for future browsing.

5.3.1 Interactive Mode

A reduced representation of the original mesh is often necessary when considering large datasets. The interactive mode has two requirements. First, it is fast enough to render at interactive rates (e.g., 10 fps). Second, the results of the interactive mode can be used as a first step in the progressive volume rendering. Because of the second requirement, level-of-detail techniques that require simplification hierarchies [27] are not feasible. Instead, a similar approach to Callahan et al. [21] is used, where a subset of the original geometry is used to create a reduced representation. In particular, Callahan et al. noticed that an efficient approximation of the dataset can be created by computing the volume rendering integral between only the boundary fragments of the mesh. A more robust version of this approximation is provided that can be used in future progressive steps.

Upon connection with the server, the client receives some initial data to begin the progressive rendering process. First, mesh parameters such as maximum edge length and scalar range are transferred for creating a preintegrated lookup table. Immediately following these parameters, the boundary triangles of the mesh are transferred to the client. These boundary triangles are placed in arrays on GPU memory and remain there for the duration of the client-server connection. If the entire boundary cannot be maintained in memory, a subset of the boundary can be used instead.

During user interaction (i.e., rotation, panning, or zooming), the following steps take place to create an approximate image.

1. Buffer T_{k12} is attached to the FBO as well as a depth buffer. The depth buffer is cleared and set to `GL_LESS` for a first pass on the boundary geometry. The depth buffer is then cleared again and set to `GL_GREATER` for a second pass at the boundary geometry. This

has the same effect as depth-peeling [44] the front and back fragments and placing them in the *k*-buffer.

2. Buffers T_{approx} and T_{k12} are attached to the FBO and a screen-aligned plane is rendered. Color and opacity for the ray-gaps between the front and back fragments from the boundary are looked up from T_{k12} and composited into T_{approx}.
3. Buffer T_{approx} is displayed to the screen.

These steps are repeated for every view change. Figure 5.3 shows the results of rendering the boundaries interactively. When the user stops interacting, the entries in T_{k12} are used for the progressive steps. The importance of using this depth-peeling approach instead of the *k*-buffer directly is discussed in more detail in Section 5.3.4.

5.3.2 Progressive Mode

After the boundary has been drawn in *Interactive Mode*, the internal triangles of the mesh are streamed in an approximate front-to-back order based on their centroids. For each portion of the geometry that arrives from the server, the progressive volume renderer takes the following steps:

1. Buffer T_{approx} is cleared.
2. Buffers T_{pro}, T_{k12}, T_{k34}, and T_{k56} are attached to the FBO and the incoming internal geometry

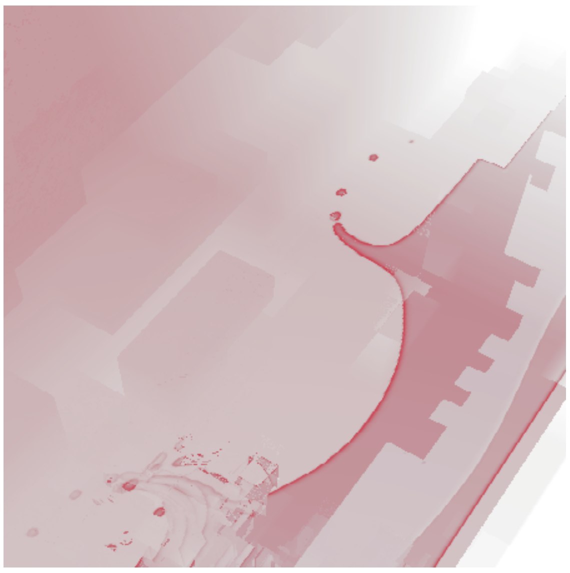


Figure 5.3. A zoomed-in view of the STP dataset (about 25M tetrahedra) captured during interaction. Significant performance improvements are made by frustum-culling the geometry on the server. Here, 50% of the geometry is culled during the progression.

is rendered. The k-buffer is used to sort the fragments and composite the results into T_{pro}. This step is similar to HAVS, but with only a portion of the geometry.

3. Buffer T_{approx} is attached to the FBO, T_{k12}, T_{k34} and T_{k56} are bound as a read-only textures, and a screen-aligned plane is drawn. The fragment shader finds the k-buffer entry f closest to the current view and the entry b farthest from the current view, looks up the color and opacity for the ray gap between f and b, and composites the result into T_{approx}.
4. Buffers T_{pro} and T_{approx} are attached to the FBO and a screen-aligned plane is drawn. The fragment shader composites T_{pro} into T_{approx}.
5. Buffer T_{approx} is displayed to the screen.

The k-buffer entries and progressive buffer T_{pro} are then ready to be used in the next rendering pass. Figure 5.4 illustrates the textures used during a progressive step. This process is repeated until all the geometry has been streamed from the server. When rendering the last portion of geometry from the server, an additional step is taken to flush the k-buffer entries into the T_{pro} by rendering $k-1$ screen-aligned planes after the second step, after which, the T_{pro} buffer contains the full quality volume rendering.

5.3.3 Completed Mode

Once the stream of geometry has terminated and the progressive volume rendering is completed, the T_{pro} buffer is captured and saved for subsequent browsing as shown if Figure 5.5. The

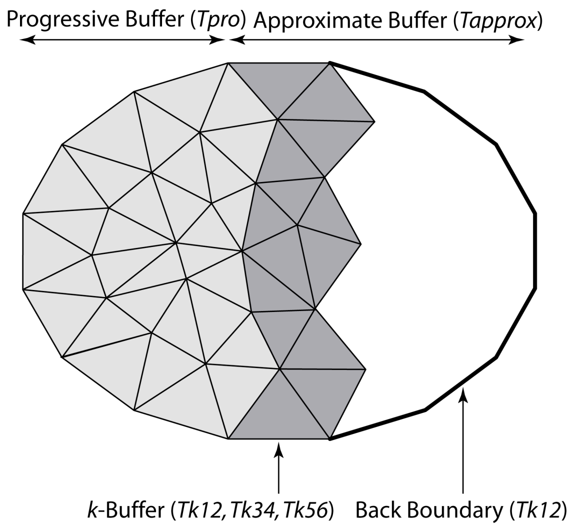

Progressive Buffer (Tpro)
Approximate Buffer (Tapprox)
k-Buffer (Tk12, Tk34, Tk56)
Back Boundary (Tk12)

Figure 5.4. The progressive volume renderer on the client. A progressive buffer is maintained between steps and an approximate buffer is created to fill the unknown region of the mesh.

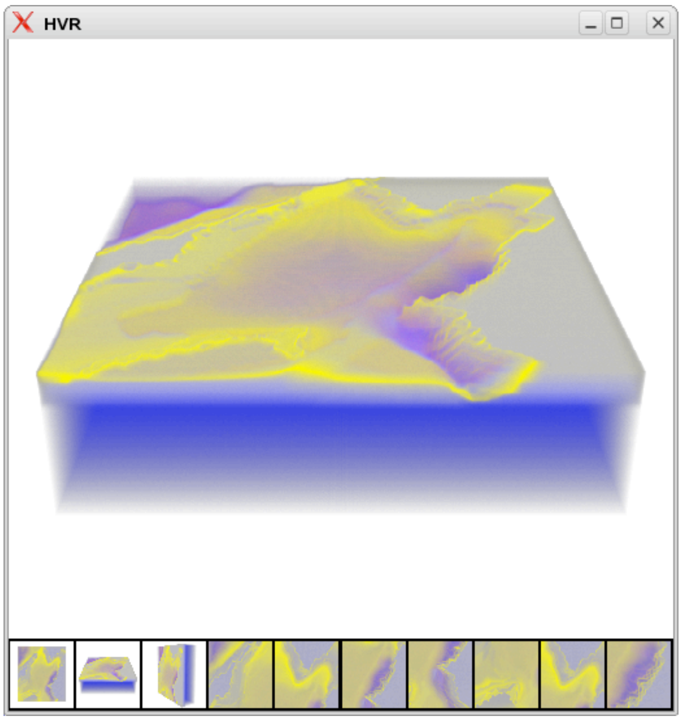


Figure 5.5. A snapshot of the interaction with the *Completed Mode* for the SF1 dataset. Upon completion of a full quality rendering, the image is automatically stored for future browsing by selecting the icons at the bottom of the window. The user may also save intermediate steps with a keystroke.

progressive volume renderer allows the user to browse any previously completed visualizations by selecting the corresponding thumbnail. Capturing these data for the user has important benefits. First, it prevents the user from losing important visualizations through interactions that could reset the results of the previous stream. Second, it allows a user to specify a set of camera positions to the progressive volume renderer so the user can easily capture an animation of the exploration process. This tool is useful for exploring previously generated results while the current view is being progressively rendered off-screen.

5.3.4 Considerations

The *k*-buffer algorithm efficiently handles streaming geometry because it simultaneously reads and writes from textures at each pass. Due to the highly parallel nature of GPU architectures, this may result in a race condition for overlapping triangles. Because the HAVS algorithm sends geometry sorted by centroid, it effectively layers the geometry in the depth direction and thus avoids these errors. However, since the depth compexity of the boundary is generally small and it is important to keep the interaction as fast as possible, the boundary faces are not sorted in object-space before rendering. This may result in some noticeable artifacts in the first pass that would remain in subsequent progressive steps. Therefore, the depth-peeling of the front and back fragments is performed before inserting them into the *k*-buffer. This resolves the race condition, improves the quality of the rendering, and maintains interactive rates (see Section 5.4).

The depth-peeling as described has the unfortunate side-effect that it removes any nonconvexities in the boundary. This can propagate through the progressive steps and cause the empty space to be composited into the final image. Since storing the back fragments in the *k*-buffer effectively reduces *k* by one during the progressive steps, storing all boundary information for nonconvex objects can severely impact sorting capabilities. A solution to this problem is to transmit boundary and internal faces during progressive steps. This would introduce redundant fragments only in the front and back and allow other boundary fragments to be used in the progression steps to avoid compositing empty space, as in HAVS. To completely remove the storage overhead of the back boundary in the *k*-buffer, an extra texture can be used to store the back fragments during *Interactive Mode* and can be bound as a read-only texture during *Progressive Mode*.

5.4 Results

The experimental results were measured on a thin client (IBM T41 laptop) running Windows with a 1.7 GHz Pentium M processor, 1.5 GB RAM, and an ATI Mobility Fire GL T2 graphics card with 128 MB RAM. The server machine was running Linux with two Dual core Opteron 2.25 GHz processors, 8 GB RAM, and an NVidia GeForce 7800 GTX graphics card. Performance timings are measured with a 512×512 viewport on a 100 Megabit/sec ethernet network with regular network load. The experiments include timing results for the progressive rendering with local and remote configurations, as well as error measurements for the progressive images. Table 5.1 shows the tetrahedra count of the test datasets, the one-time penalty to reformat them into the binary format used by the server, and the resulting size of the binary files.

The timing measurements are shown in Table 5.2 for four large meshes. The Fighter and F16 datasets are simulations of jets, the SF1 dataset is an earthquake simulation, and the STP

Table 5.1. Experimental datasets for progressive rendering with initial tetrahedra count, time to preprocess tetrahedral mesh to binary triangle format, and resulting size of the dataset.

Dataset	Tetrahedra	Preprocess	Size
Fighter	1.4 M	15 s	117 MB
F16	6.3 M	81 s	531 MB
SF1	13.9 M	110 s	1165 MB
STP	25.0 M	458 s	2087 MB

Table 5.2. Performance analysis of the preprocessing and one step of the progressive volume rendering. Measurements are given in seconds and were obtained for a client running on the server (Local) and on a laptop over the network (Ethernet).

Dataset	Server Preprocess		Server		Transfer	Client		Total
	Load	Octree	Trav.	Sort		Inter.	Prog.	
Local								
Fighter	0.25	0.92	0.79	0.78	1.32	0.01	0.21	1.53
F16	1.10	6.12	1.17	4.18	4.17	0.01	0.39	4.56
SF1	2.53	7.87	9.04	7.90	16.16	0.01	0.43	16.59
SF1 (25%)	2.53	7.87	2.32	1.49	3.10	0.01	0.71	3.81
STP	36.55	18.46	8.62	18.89	21.80	0.38	9.82	31.62
STP (25%)	36.55	18.46	2.13	2.96	3.65	0.38	0.36	4.01
Ethernet								
Fighter	0.25	0.92	0.79	0.78	13.12	0.10	10.77	23.89
F16	1.10	6.12	3.95	4.68	102.51	0.10	128.06	230.57
SF1	2.53	7.87	10.05	9.17	237.62	0.24	501.15	738.77
SF1 (25%)	2.53	7.87	2.41	1.70	87.73	0.24	32.37	120.10
STP	36.55	18.46	51.74	17.26	727.86	0.45	551.98	1279.84
STP (25%)	36.55	18.46	11.19	2.73	208.05	0.45	121.65	329.70

dataset is a simulation of a sphere going through a plate. The measurements can be broken into four important sections: server preprocessing, server, data transfer from the client to the server, and client. The preprocessing step occurs on the server and includes loading the file from disk, and building an octree, and transferring the mesh information to the client. By extending the binary file format to include the octree structure, the server preprocessing time could be decreased even further. The timing results for the server, client, data transfer, and total time represent one progressive rendering of the dataset from views that include the whole mesh. In addition, for the larger datasets, view showing only 25% of the mesh is used, which takes advantage of frustum culling. Because the client uses a thread for rendering and another for fetching data from the server, much of the data transfer and rendering work is done in parallel. Therefore, the data transfer is measured as the total client time for a progressive step minus the rendering time. In the experiments, the interactive manipulation of the progressive renderer was able to achieve interactive rates for all but the largest dataset on the thin client.

Since data transfer over the network is one of the main bottlenecks of the progressive renderer, experiments were generated to tune the stream parameters. One important consideration is the latency of the network. Several settings on the server effect the round-trip latency of the system — the time for the client to send a packet and receive a response. An obvious consideration is the compression of the geometry during transmission. For the network rendering, full compression of the stream was used, resulting in about a 60% compression rate, which dramatically improved performance. Other important considerations include the octree resolution and stream size (number of triangles sent on each progressive step). Finer octree resolution and higher stream size improves performance, but decreases the number of progressive steps and increases memory usage on the client. Table 5.3 shows the effects of these parameters on the latency for the Fighter dataset. For the experiments with the thin client (see Table 5.2), a stream size of 100K triangles and an octree resolution of 1K triangles per octree node was the limit.

The final experiment was to analyze the quality of the progressive steps. To measure this metric, root mean squared (RMS) error was used to compare incremental steps with the final rendering for all of the experimental datasets. Figure 5.6 shows a plot of these errors as the

Table 5.3. Latency analysis of different server settings on the Fighter dataset with an octree depth of seven, a 802.11 wireless network at 54 Mbps, and an ethernet network at 100 Mbps.

Stream Size	Local	Ethernet	Wireless
1K Triangles	0.082 s	0.050 s	0.051 s
10K Triangles	0.085 s	0.090 s	0.130 s
100K Triangles	0.574 s	1.042 s	1.893 s

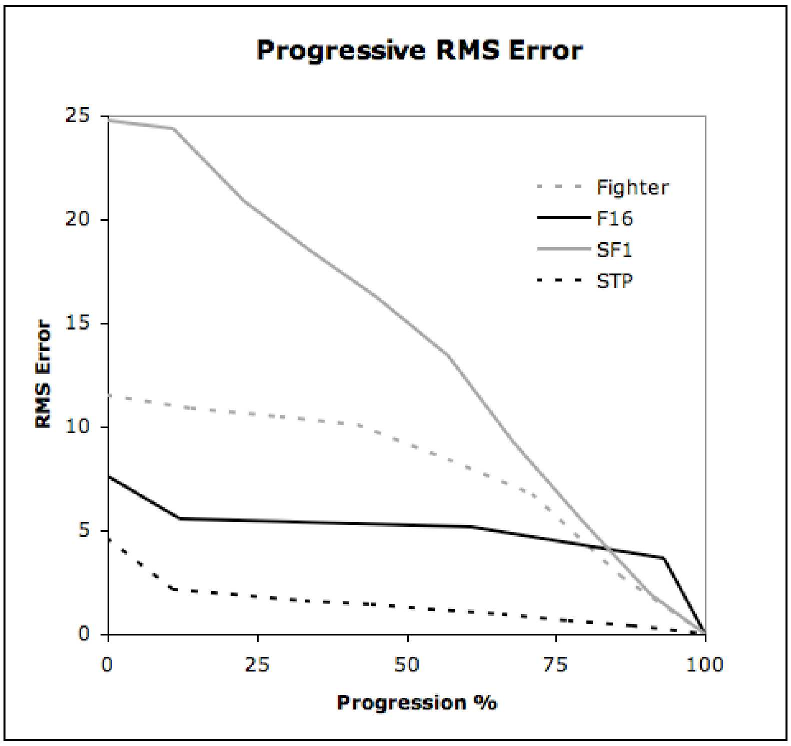


Figure 5.6. Quality comparisons of progressive steps. Root Mean Squared Error (RMS) is shown for the progressive images going from only the boundaries (0%) to the full quality (100%).

progression refines and Figures 5.7 and 5.8 show visual results of the progression. Since the quality of the approximation is directly related to the transfer function, the transfer functions shown in the figures that highlight the more relevant portions of the data was used. These results show that the image quality steadily converges to the full quality image, which is important for allowing the user to explore the dataset efficiently.

5.5 Discussion

The progressive volume rendering system is unique because it efficiently handles data of arbitrary size on a thin client. This is done using a novel technique that keeps only image data and boundary geometry in GPU memory on the client for each step. The amount of memory used on the client can be bounded by adjusting the stream size and the size of the boundary. In the experiments, the boundary was not large enough to adversely affect performance or expend memory constraints. In fact, even with a 25 million tetrahedron dataset, the boundary can be volume rendered with the algorithm on a thin client in *Interactive Mode* at about two frames-per-second. This interactivity depends heavily on the boundary complexity of the dataset. If a dataset has a

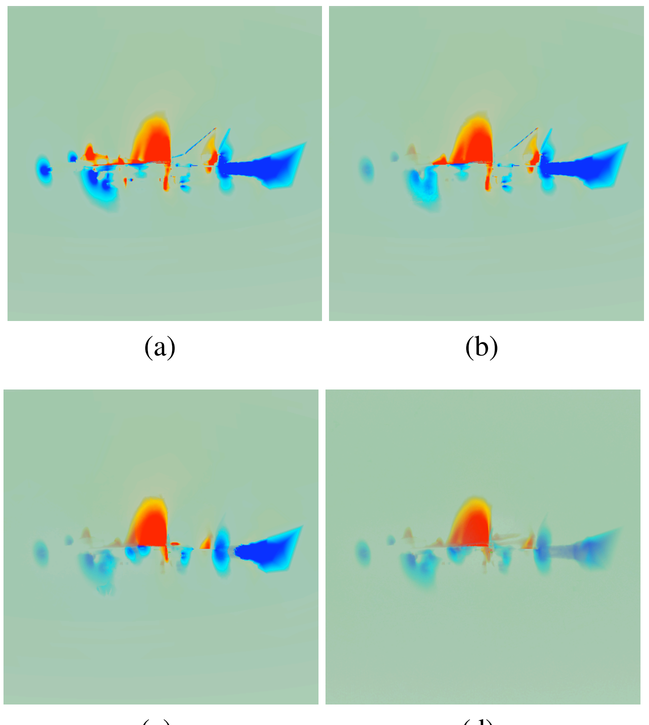


Figure 5.7. A zoomed-in portion of the F16 dataset (about six million tetrahedra) shown progressively at (a) 0%, (b) 12%, (c) 61%, and (d) 100%.

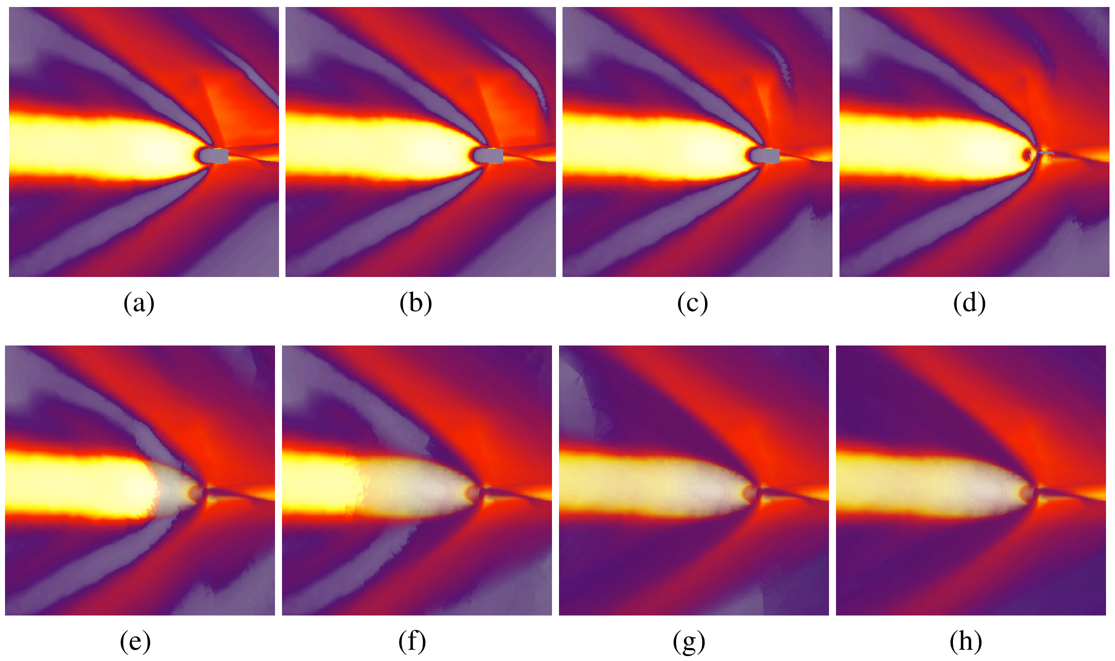


Figure 5.8. A zoomed-in portion of the Fighter dataset (about 1.4 million tetrahedra) shown progressively at (a) 0%, (b) 14%, (c) 29%, (d) 43%, (e) 57%, (f) 71%, (g) 86%, and (h) 100%. View-frustum culling on the server removed 75% of the original geometry for this view.

boundary too large to fit in GPU memory or render at acceptable rates, the algorithm would work efficiently by using only a random subset of the boundaries for an approximate rendering during interaction. This would have the effect of lightening the general appearance of the approximation. The remaining boundary triangles could then be streamed before the rest of the internal faces.

An important consideration for a progressive renderer is the depth complexity and structure of the dataset. The client rendering is fill-bound and thus depends more on the view selected or screen size than on the depth complexity. However, the depth complexity of the dataset may adversely affect performance of the geometry processing on the server because more culling and sorting passes are required. The approach for culling by depth on the server assumes an even distribution of triangles. For most of the experimental datasets, this results in few triangles selected in some ranges and many in others. This is balanced by accumulating triangles on the server until a target packet size is reached. A better approach may be to avoid using fixed depth ranges by traversing the octree incrementally from the front to the back, rather than doing a hierarchical culling. In the experiments, the aspect ratio or depth complexity of the dataset seems to impact overall performance only slightly if the server parameters are properly selected (number of depth ranges and minimum packet size).

A client-server progressive volume renderer is advantageous because it allows the data to be stored in a central repository, while the rendering can be performed with the help of graphics hardware on a client. This efficiently splits the load of the client and server for datasets that may be too large to render locally. Since processing power continues to increase rapidly for both CPUs and GPUs, it is becoming increasingly important to develop algorithms that efficiently harness the computational power without being limited by memory constraints. The proposed algorithm uses this approach by acting as a stream renderer and allowing the interactive exploration of a dataset with only a portion of the geometry.

The main disadvantage of using this paradigm is that data transfer over the network incurs a substantial penalty. This penalty is partially reduced with the use of lossless compression of the stream. If some loss in quality is acceptable, quantization techniques could also be applied to reduce the bandwidth of the geometry. Because data transfer is a limiting factor in quickly visualizing a full quality rendering, the quality of the approximation is important. Unlike naïve approaches that render only an opaque boundary mesh or outline, the proposed initial approximation gives excellent results with little overhead. With only a few iterations, the progressive volume renderer converges to the final image which facilitates dynamic exploration. This aspect is found to be useful because often in the exploration process, the user will not wait for the entire progressive volume rendering before moving on to another viewpoint. Because only the geometry within the current view frustum is transferred, efficiently exploring details of the dataset becomes easy. This feature also makes rapid transfer function exploration possible. With each update of the transfer function, the stream can be reset and the progression started. For datasets with more important features in the center, the boundary may not give a good approximation. A more advanced technique that uses multiple k-buffers could be employed to render several advancing progressions at once which results in an increased rendering overhead. A simpler approach would be the addition of user-controlled cutting planes that could cull geometry on the server, thus reducing the amount of data sent to the client and allowing the rapid visualization of internal structures. Along with the image capturing system, which keeps previously computed results, these features would provide a powerful data exploration tool for large datasets.

5.6 Summary

The algorithm presented in this chapter provides a progressive volume rendering system for interactively exploring large unstructured datasets. A novel approach is used for balancing the load of the client and server while minimizing the memory constraints of the client. In fact, the algorithm can bound the memory usage of the client machine to allow a wide variety of client devices. A novel progressive algorithm was introduced that efficiently uses the GPU to incrementally refine the visualization by retaining only image data. An interactive mode can be efficiently computed with the addition of boundary triangles that can be kept in GPU memory. To further improve interaction, the system keeps previously computed visualizations that can be interactively browsed while progressively rendering the visualization. Finally, detailed experiments were provided and the trade-offs of a client-server approach for volume rendering unstructured grids was discussed.

CHAPTER 6

VOLUME RENDERING TIME-VARYING SCALAR FIELDS

Large amounts of time-varying volumetric data are being generated by simulations. Techniques to handle this dynamic data for unstructured grids are virtually nonexistent. In a recent survey article on the topic, Ma [84] says:

> Research so far in time-varying volume data visualization has primarily addressed the problems of encoding and rendering a single scalar variable on a regular grid... Time-varying unstructured grid data sets have been either rendered in a brute force fashion or just resampled and downsampled onto a regular grid for further visualization calculations...

One of the key problems in handling time-varying data is the raw size of the data that must be processed. For rendering, these datasets need to be stored (and/or staged) in either main memory or GPU memory. Data transfer rates create a bottleneck for the effective visualization of these datasets. A number of successful techniques for time-varying regular grids have used compression to mitigate this problem, and allow for better use of resources. Most solutions described in the literature consider only structured grids, where exploiting coherence (either spatial or temporal) is easier due to the regular structure of the datasets. For unstructured grids, however, the compression is more challenging and several issues need to be addressed.

There are four fundamental pieces to adaptively volume render dynamic data. First, compression of the dynamic data for efficient storage is necessary to avoid exhausting available resources. Second, handling the data transfer of the compressed data is important to maintain interactivity. Third, efficient volume visualization solutions are necessary to provide high-quality images that lead to scientific insight. Furthermore, the framework has to be flexible enough to support multiple visualization techniques as well as data that change at each frame. Fourth, maintaining a desired level of interactivity or allowing the user to change the speed of the animation is important for the user experience. Therefore, level-of-detail approaches that generally work on static datasets must be adapted to efficiently handle the dynamic case.

Since multiple CPUs and programmable GPUs are becoming common for desktop machines, this chapter concentrates on efficiently using all the available resources. The system performs decompression, object-space sorting, and level-of-detail operations with multiple threads on the CPUs for the next time-step while simultaneously rendering the current time-step using the GPU. This parallel computation results in only a small overhead for rendering time-varying data over previous static approaches.

To demonstrate the flexibility of this framework, two of the most common visualization techniques for unstructured grids are integrated. Both direct volume rendering as well as isosurface computation are incorporated into this adaptive, time-varying framework.

Though the goal is to eventually handle data that change in geometry and even topology over time, this chapter concentrates on the more specific case of time-varying scalar fields on static geometry and topology. Figure 6.1 illustrates this system in action on an unstructured grid representation of the Turbulent Jet dataset. The main contributions of this chapter are:

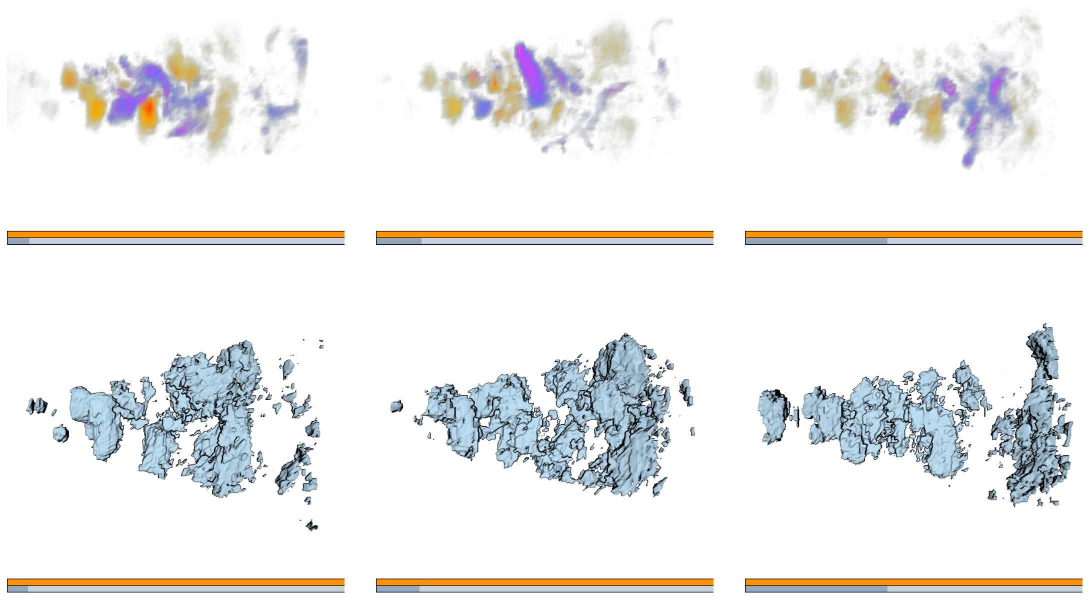


Figure 6.1. Different time instances of the Turbulent Jet dataset consisting of one million tetrahedra and rendered at approximately six frames per second using direct volume rendering (top) and isosurfacing (bottom). The user interface consists of an adjustable orange slider representing the level-of-detail and an adjustable gray slider representing the current time instance.

- The data transfer bottleneck is mitigated with compression of time-varying scalar fields for unstructured grids;
- A hardware-assisted volume rendering system is enhanced to efficiently prefetch dynamic data by balancing the CPU and GPU loads;
- Direct volume rendering and isosurfacing are incorporated into the adaptive framework;
- New importance sampling approaches for dynamic level-of-detail are introduced that operate on time-varying scalar fields.

The rest of this chapter is organized as follows. Section 6.1 outlines the proposed system for adaptively volume rendering unstructured grids with time-varying scalar fields. The results of the algorithm are shown in Section 6.2, a brief discussion follows in Section 6.3, and a summary is described in Section 6.4.

6.1 Adaptive Time-Varying Volume Rendering

The proposed system for adaptive time-varying volume rendering of unstructured grids consists of four major components: compression, data transfer, hardware-assisted volume visualization, and dynamic level-of-detail for interactivity. Figure 6.2 shows the interplay of these components.

Desktop machines with multiple processors and multiple cores are becoming increasingly typical. Thus, interactive systems should explore these features to leverage computational power. For graphics-heavy applications, the CPU cores can be used as an additional cache for the GPU, which may have limited memory capabilities. With this in mind, the operations on the CPU and GPU are parallelized with multithreading to achieve interactive visualization.

There are currently three threads in the system to parallelize the CPU portion of the code. These threads run while the geometry from the previous time-step is being rendered on the GPU. Therefore, the threads can be considered a prefetching of data in preparation for rasterization. The first thread handles decompression of the compressed scalar field (see Section 6.1.1). The second thread handles object-space sorting of the geometry (see Section 6.1.3). Finally, the third thread is responsible for rendering. All calls to the graphics API are done entirely in the rendering thread since only one thread at a time may access the API. This thread waits for the first two threads to finish, then copies the decompressed scalar values and sorted indices before rendering. This frees the other threads to compute the scalars and indices for the next time-step. These three threads work entirely in parallel and result in a time-varying visualization that requires very little overhead over static approaches.

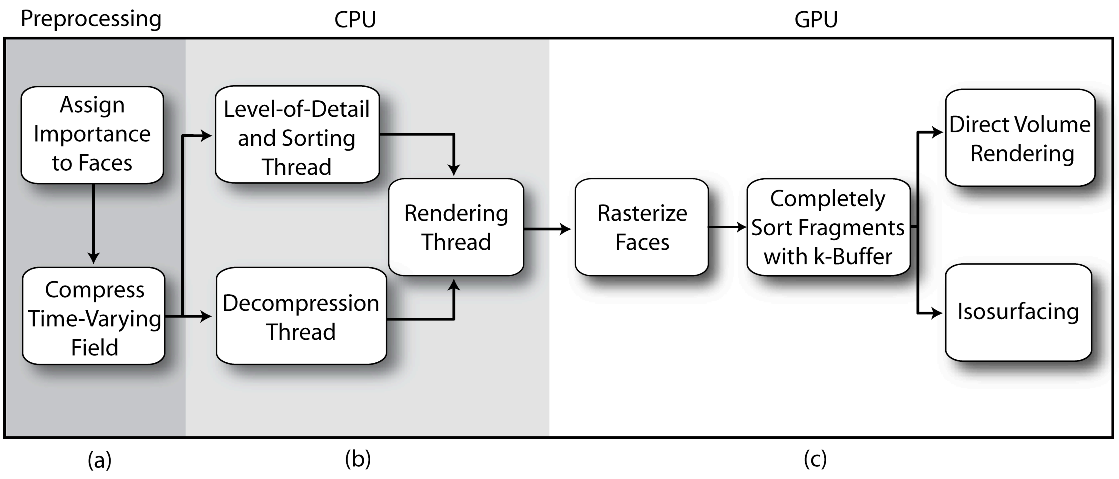

Preprocessing
CPU
GPU
Assign Importance to Faces
Compress Time-Varying Field
Level-of-Detail and Sorting Thread
Decompression Thread
Rendering Thread
Rasterize Faces
Completely Sort Fragments with k-Buffer
Direct Volume Rendering
Isosurfacing
(a)
(b)
(c)

Figure 6.2. An overview of the proposed system for dynamic volume rendering. (a) Data compression and importance sampling for level-of-detail are performed in preprocessing steps on the CPU. (b) Then during each pass, level-of-detail selection, optional object-space sorting, and data decompression for the next step occur in parallel on multiple CPUs. (c) Simultaneously, the image-space sorting and volume visualization are processed on the GPU.

6.1.1 Compression

Compression is important to reduce the memory footprint of time-varying data, and the consideration of spatial and temporal coherence of the data is necessary when choosing a strategy. For example, it is common to exploit spatial coherence in time-varying scalar fields defined on structured grids, such as in the approach described by Schneider et al. [119], where a multiresolution representation of the spatial domain using vector quantization is used. This solution works well when combined with texture-based volume rendering, which requires the decompression to be performed at any given point inside the volume by incorporating the differences at each resolution level.

In unstructured grids, the irregularity of topological and geometric information makes it hard to apply a multiresolution representation over the spatial domain. In this system compression is applied on the temporal domain by considering scalar values individually for each mesh vertex. By grouping a fixed number of scalar values defined over a sequence of time instances a suitable representation is obtained for applying a multiresolution framework.

The system collects blocks of 64 consecutive scalars associated with each mesh vertex, applies a multiresolution representation that computes the mean of each block along with two difference vectors of size 64 and 8, and uses vector quantization to obtain two sets of representatives (codebooks) for each class of difference vectors. For convenience a fixed number of time instances is used, but this is compensated by increasing the number of entries in the codebooks if temporal coherence is reduced and leads to compression errors.

This solution works well for projective volume rendering that sends mesh faces in visibility ordering to be rendered. At each rendering step, the scalar value for each mesh vertex in a given time instance is decompressed by adding the mean of a given block interval to two difference values, which are recovered from the codebooks using two codebook indices i_8 and i_{64} (see Figure 6.3).

6.1.2 Data Transfer

There are several alternatives to consider when decompressing and transferring time-varying data to the volume renderer. This is a critical point in the system and has a great impact on its overall performance. In this section, the alternatives are discussed and the current solution is explained. It is important to point out that with future CPU and GPU configurations this solution might need to be revisited.

Since the decompression is done on a per-vertex basis, the first approach was to use the vertex shader on the GPU. This requires the storage of codebooks as vertex textures, and the transfer for each vertex of three values as texture coordinates (mean and codebook indices i_8 and i_{64}). In practice, this solution does not work well because the current generation of graphics hardware does not handle vertex textures efficiently and incurs several penalties due to cache misses, and the arithmetic calculations in the decompression are too simple to hide this latency.

The second approach was to use the GPU fragment shader. Since computation is done at a fragment level, the decompression and the interpolation of the scalar value for the fragment is necessary. This requires three decompression steps instead of a single step as with the vertex shader approach (which benefits from the interpolation hardware). Also, this computation requires accessing the mean and codebook indices. Sending this information as a single vertex attribute is not possible due to interpolation, and multiple-vertex attributes increase the amount of data transfer per vertex. As the volume renderer runs in the fragment shader, this solution also increases the shader complexity and thus reduces performance of the system.

The final (and current) solution is to perform the decompression on the CPU. Since codebooks usually fit in CPU memory—a simple paging mechanism can be used for really large data—the main cost of this approach is to perform the decompression step and send scalar values through the pipeline. This data transfer is also necessary with the other two approaches. The number

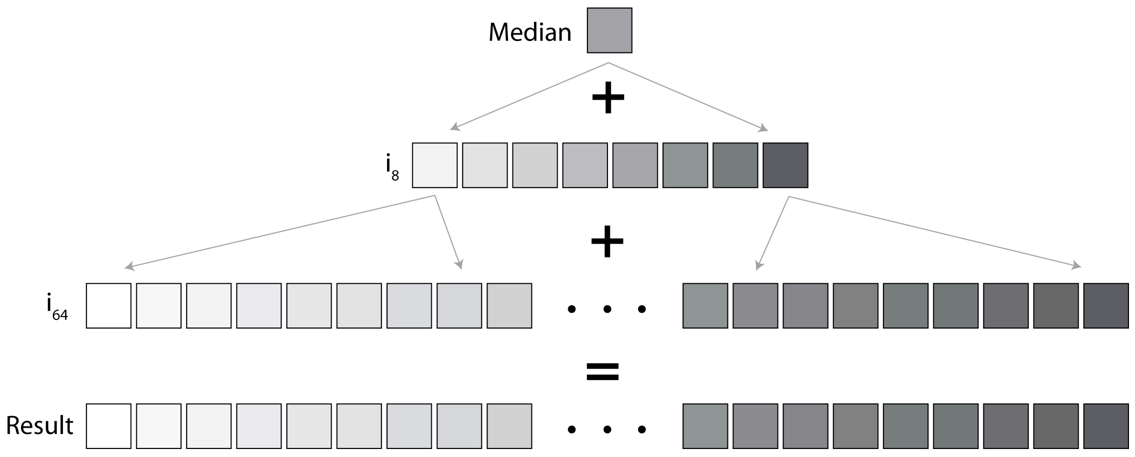


Figure 6.3. An illustration of decompression of vector quantized scalars. Decompression of a given time instance for each mesh vertex requires adding the scalar mean of a block to the quantized differences recovered from the respective entries (i_8 and i_{64}) in the codebooks.

of decompression steps is reduced to the number of vertices, unlike the vertex shader approach which requires three times the number of faces.

6.1.3 Volume Rendering

The system is based on the Hardware-Assisted Visibility Sorting (HAVS) algorithm of Callahan et al. [22, 56]. Figure 6.2 shows how the volume rendering system handles time-varying data. The framework supports both direct volume rendering as well as isosurfacing.

The HAVS algorithm is a general visibility ordering algorithm for renderable primitives that works in both object-space and image-space. In object space the unique triangles that compose the tetrahedral mesh are sorted approximately by their centroids. This step occurs entirely on the CPU. In image-space, the triangles are sorted and composited in correct visibility order using a fixed size A-buffer called the *k*-buffer. The *k*-buffer is implemented entirely on the GPU using fragment shaders. Because the HAVS algorithm operates on triangles with no need for neighbor information, it provides a flexible framework for handling dynamic data. In this case, the triangles can be stored on the GPU for efficiency, and the scalar values as well as the object-space ordering of the triangles can be streamed to the GPU at each time instance.

The proposed algorithm extends the HAVS algorithm with time-varying data with virtually no overhead by taking advantage of the HAVS architecture. Since work performed on the CPU can be performed simultaneously to work on the GPU, this parallelization is leveraged to prefetch the time-varying data. During the GPU rendering stage of the current time instance, the CPU is used to decompress the time-varying field of the next time-step and prepare it for rendering. User provided viewing transformations that affect visibility order are also distinguished from those that do not and perform visibility ordering only when necessary. Therefore, the object-space centroid sort only occurs on the CPU during frames that have a change in the rotation transformation. This avoids unnecessary computation when viewing time-varying data.

To manage the time-stepping of the time-varying data, the algorithm automatically increments the time instance at each frame. To allow more control from the user, a slider is also provided for interactive exploration of the time instances.

6.1.3.1 Direct Volume Rendering

The HAVS algorithm performs direct volume rendering with the use of pre-integration [41]. In a preprocess, a three-dimensional lookup table is computed that contains the color and opacity for every set of scalars and distances between them (s_f, s_b, d). Then, as fragments are rasterized, the k-buffer is used to retrieve the closest two fragments and the front scalar s_f, back scalar s_b, and distance d between them is calculated. The color and opacity for the gap between the fragments is determined with a texture lookup, then composited into the framebuffer.

6.1.3.2 Isosurfaces

The HAVS algorithm is extended to perform isosurfacing in the time-varying framework. The fragments are sorted using the k-buffer as described above. However, instead of compositing the integral contribution for the gap between the first and second fragments, a simple test is performed to determine if the isosurface value lies between them. If so, the isosurface depth is determined by interpolating between the depths of the two fragments. The result is a texture that contains the depth for the isosurface at each pixel (i.e., a depth buffer), which can be displayed directly as a color buffer or postprocessed to include shading.

There are several differences between this isosurfacing using HAVS and previous hardware-assisted approaches. First, with the advent of predicates in the fragment shader, a direct isosurface comparison can be performed efficiently, without the need of texture lookups as in previous work by Röttger et al. [115]. Second, the k-buffer naturally provides the means to handle multiple transparent isosurfaces by compositing the isosurface fragments in the order that they are extracted. Third, to handle lighting of the isosurfaces, keeping normals is avoided in the k-buffer by using an extra shading pass. One option is a simple depth-based shading [133], which may not give sufficient detail of the true nature of the surface. Another option is to use a gradient-free shading approach similar to work by Desgranges et al. [33], which uses a preliminary pass over the geometry to compute a shadow buffer. Instead, this extra geometry pass is avoided by using screen-space shading of the computed isosurface through central differencing on the depth buffer [77]. This results in fast and high quality shading. The isosurface figures shown in this chapter were generated with the latter shading model. Finally, since the isosurfacing algorithm is based on the direct volume rendering algorithm, the same level-of-detail strategies can be used to increase performance. However, level-of-detail rendering may introduce discontinuities in the depths that define the isosurface, adversely affecting the quality of the image-space shading. Thus, an additional smoothing pass is performed on the depths using a convolution filter, which removes sharp discontinuities, before the final shading. This extra pass is inexpensive and typically only necessary at low levels-of-detail.

6.1.4 Time-Varying Level-of-Detail

Recent work by Callahan et al. [21] introduces a new dynamic level-of-detail (LOD) approach that works by using a *sample-based simplification* of the geometry. This algorithm operates by assigning an importance to each triangle in the mesh in a preprocessing step based on properties of the original geometry. Then, for each pass of the volume renderer, a subset of the original geometry is selected for rendering based on the frame rate of the previous pass. This recent LOD strategy was incorporated into the original HAVS algorithm to provide a more interactive user experience.

An important consideration for visualizing time-varying data is the rate at which the data are progressing through the time instances. To address this problem, the proposed algorithm uses this LOD approach to allow the user to control the speed and quality of the animation. Since this work assumes time-varying scalar fields, heuristics that attempt to optimize the quality of the mesh based on the scalar field are ideal. However, approaches that are based on a static mesh can be poor approximations when considering a dynamically changing scalar field.

Callahan et al. introduce a heuristic based on the scalar field of a static mesh for assigning an importance to the triangles. The idea is to create a scalar histogram and use stratified sampling to stochastically select the triangles that cover the entire range of scalars. This approach works well for static geometry, but may miss important regions if applied to a time-step that does not represent the whole time-series well. Recent work by Akiba et al. [1] classifies time-varying datasets as either *statistically dynamic* or *statistically static* depending on the behavior of the time histogram of the scalars. A statistically dynamic dataset may have scalars that change dramatically in some regions of time, but remain constant during others. In contrast, the scalars in statistically static datasets change consistently throughout the time-series. Because datasets vary differently, two sampling strategies were developed for dynamic LOD: a local approach for statistically dynamic datasets and a global approach for statistically static datasets.

To incorporate these LOD strategies into the time-varying system, two types of interactions are allowed based on user preference. The first is to keep the animation at a desired frame-rate independent of the data size or viewing interaction. This dynamic approach adjusts the LOD on the fly to maintain interactivity. The second type of interaction allows the user to use a slider to control the LOD. This slider dynamically changes the speed of the animation by setting the LOD manually. Since visibility ordering-dependent viewing transformations occur on the CPU in parallel to the GPU rendering, they do not change the LOD or speed of the animation. Figure 6.2 shows the interaction of the LOD algorithm with the time-varying data.

6.1.4.1 Local Sampling

Datasets with scalars that vary substantially in some time regions, but very little in others, benefit from a local sampling approach. The general idea is to apply methods for static datasets to multiple time-steps of a time-varying dataset and change the sampling strategy to correspond to the current region. Local sampling is performed by selecting time-steps at regular intervals in the time-series and importance sampling the triangles in those time-steps using the previously described stochastic sampling of scalars. Since the LOD selects a subset of triangles ordered by importance, a separate list of triangles is created for each partition of the time sequence. Then, during each rendering step, the region is determined and the corresponding list is used for rendering. Since changing the list also requires the triangles to be resorted for rendering, this operation occurs in the sorting thread to minimize delays between frames.

Statistically dynamic datasets benefit from this approach because the triangles are locally optimized for rendering. The disadvantage of this approach is that because the active set of renderable triangles may change between time intervals, flickering may occur. However, this is a minor issue when using dynamic LOD because the number of triangles drawn at each frame may be changing anyway, to maintain interactivity.

Certain datasets may contain regions in time where none of the scalars are changing and other regions where many scalars are changing. These datasets would benefit from a nonlinear partitioning of the time sequence (i.e., logarithmic). A more automatic approach to partitioning the time-steps is to greedily select areas with the highest scalar variance to ensure that important changes are not missed. In the experiments described in this chapter, a regular interval is used because the experimental datasets do not fall into this category.

6.1.4.2 Global Sampling

A global strategy is desirable in datasets that are statistically static due to its simplicity and efficiency. For global sampling, one ordering is determined that attempts to optimize all time-steps. This has the advantage that it does not require changing the triangle ordering between frames and thus gives a smoother appearance during an animation. Figure 6.4 shows an example of a dataset rendered using global sampling.

A sampling can be obtained globally using a statistical measure of the scalars. For n time-steps, consider the n scalar values s for one vertex as an independent random variable X, then the expectation at that position can be expressed as

$$E[X] \quad = \quad \sum_{1}^{n} s(Pr\{X = s\}),$$

where $Pr\{X = s\} = 1/n$. The dispersion of the probability distribution of the scalars at a vertex can then be expressed as the variance of the expectation:

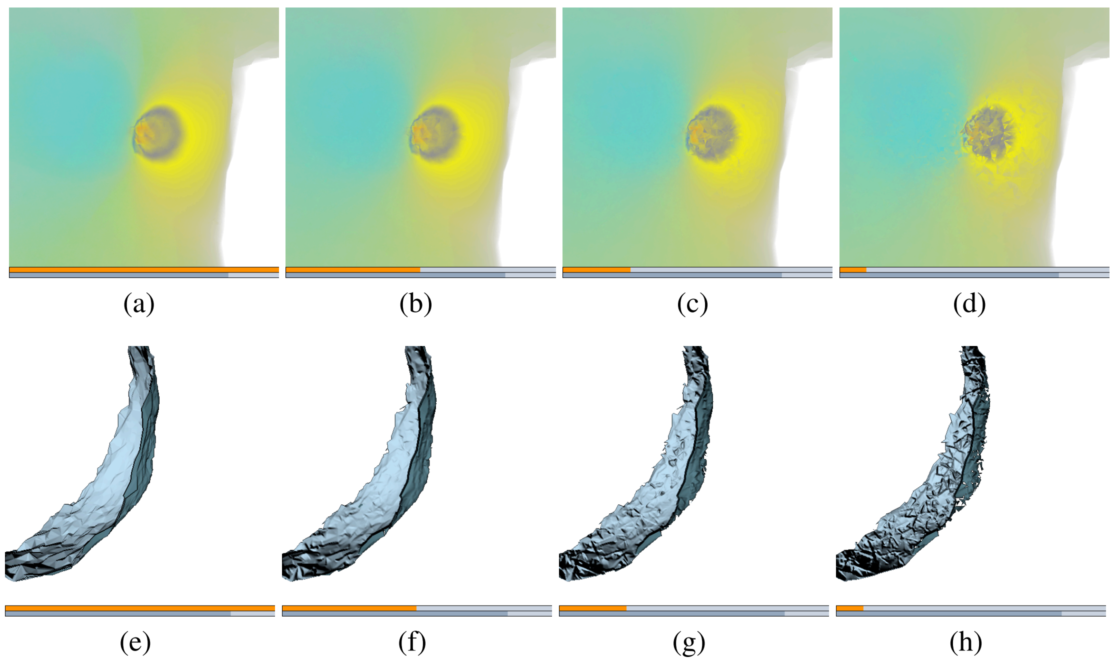


Figure 6.4. The time-varying level-of-detail (LOD) strategy using the coefficient of variance for the Torso dataset (50K tetrahedra and 360 time steps). For a close-up view using direct volume rendering: (a) 100% LOD at 18 fps, (b) 50% LOD at 40 fps, (c) 25% LOD at 63 fps, and (d) 10% LOD at 125 fps. For the same view using isosurfacing: (e) 100% LOD at 33 fps, (f) 50% LOD at 63 fps, (g) 25% LOD at 125 fps, and (h) 10% LOD at 250 fps. Front isosurface fragments are shown in light blue and back fragments are shown in dark blue.

$$\begin{aligned} Var[X] &= E[X^2] - E^2[X] \\ &= \sum_1^n (\frac{s^2}{n}) - (\sum_1^n \frac{s}{n})^2 \end{aligned}$$

In essence, this gives a *spread* of the scalars from their expectation. To measure dispersion of probability distributions with widely differing means, it is common to use the coefficient of variation C_v, which is the ratio of the standard deviation to the expectation. This metric has been used in related research for transfer function specification on time-varying data [58, 1] and as a measurement for spatial and temporal error in a time-varying structured grids [122]. Thus, for each triangle t, the importance can be assigned by calculating the sum of the C_v for each vertex as follows:

$$C_v(t) = \sum_{i=1}^{3} \frac{\sqrt{Var[X_{t(i)}]}}{E[X_{t(i)}]}$$

This results in a dimensionless quantity that can be used for assigning importance to each face by directly comparing the amount of change that occurs at each triangle over time.

The algorithm provides good quality visualizations even at lower levels-of-detail because the regions of interest (those that are changing) are given a higher importance (see Figure 6.4). The described heuristic works especially well in statistically static datasets if the mesh has regions that change very little over time since they are usually assigned a lower opacity and their removal introduces very little visual difference.

6.2 Results

In this section, the results that are reported were obtained using a PC with Pentium D 3.2 GHz Processors, 2 GB of RAM, and an NVidia 7800 GTX GPU with 256 MB RAM. All images were generated at 512×512 resolution.

6.2.1 Datasets

The datasets used in the tests are diverse in size and number of time instances. The time-varying scalars on the SPX1, SPX2 and Blunt Fin datasets were procedurally generated by linear interpolating the original scalars to zero over time. The Torso dataset shows the result of a simulation of a rotating dipole in the mesh. The SPX-Force (SPXF) dataset represents the magnitude of reaction forces obtained when a vertical force is applied to a mass-spring model that has as particles the mesh vertices and as springs the edges between mesh vertices. Finally, the Turbulent Jet (TJet) dataset represents a regular time-varying dataset that was tetrahedralized and simplified to a reduced representation of the original. The meshes used in the tests, with their respective sizes, are listed in Table 6.1.

6.2.2 Compression

The compression of Time-Varying Scalar Field (TVSF) data use an adaption of the vector quantization code written by Schneider et al. [119], as described in Section 6.1.1. The original code works with structured grids with building blocks of $4 \times 4 \times 4$ (for a total of 64 values per block). To adapt its use for unstructured grids it is necessary to group TVSF data into basic blocks with the same amount of values. For each vertex in the unstructured grid, the scalar values corresponding to 64 contiguous instances of time are grouped into a basic block and sent to the VQ code.

The VQ code produced two codebooks containing difference vectors for the first and second level in the multiresolution representation, each with 256 entries (64×256 and 8×256 codebooks). For the synthetic datasets this configuration led to acceptable compression results as seen in Table 6.1. However, for the TJet and SPXF datasets the number of entries in the codebook was increased due to the compression error obtained. Both datasets were compressed using codebooks with 1024 entries.

The size of TVSF data without compression is given by $size_u = v \times t \times 4B$, where v is the number of mesh vertices, t is the number of time instances in each dataset, and each scalar uses four bytes (float). The compressed size using VQ is equal to $size_{vq} = v \times c \times 3 \times 4B\ +\ c \times size\_codebook$, where c is the number of codebooks used ($c = t/64$), s is the number of entries in the codebook (256 or 1024), each vertex requires 3 values per codebook (mean plus codebook indices i_8 and i_{64}), and each codebook size corresponds to $s \times 64 \times 4B\ +\ s \times 8 \times 4B$.

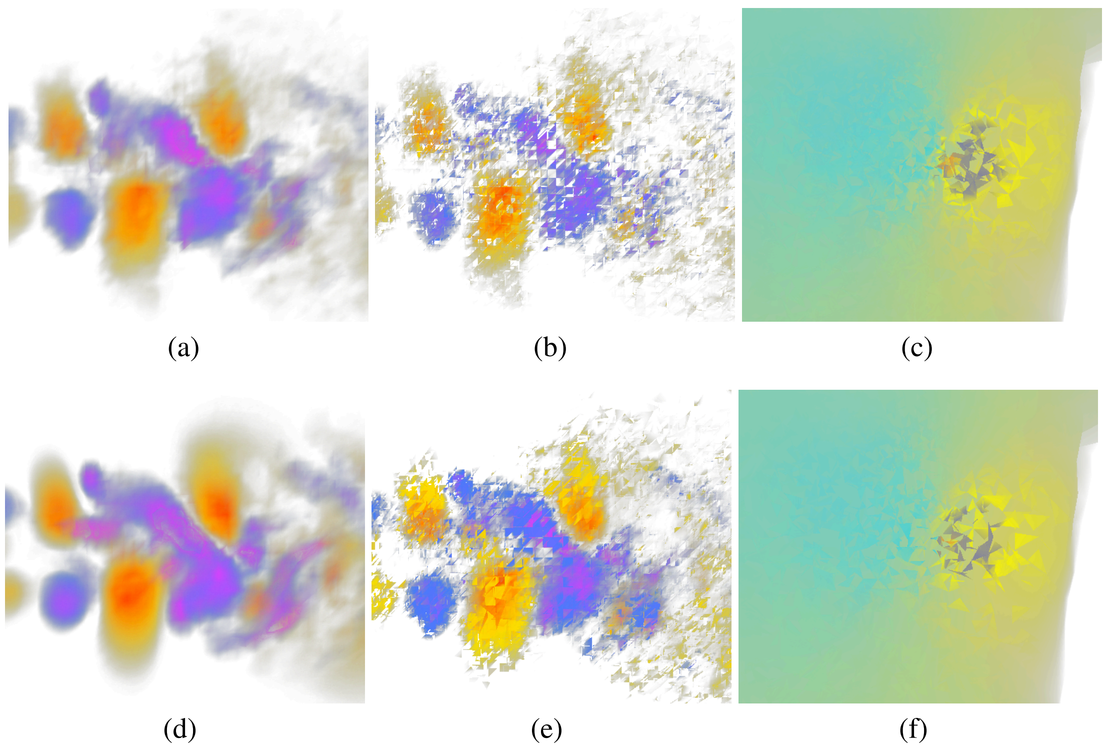
In Table 6.1 the compression results obtained are summarized. In addition to the signal-to-noise ration (SNR) given by the VQ code, the minimum and maximum discrepancy between the

Table 6.1. Results of compression sizes, ratios, and error for time-varying volumes.

Mesh	**Num Verts**	**Num Tets**	**Time Instances**	**Size TVSF**	**Size VQ**	**Comp. Ratio**	**SNR Min**	**SNR Max**	**Max Error**
SPX1	36K	101K	64	9.0M	504K	18.3	39.5	42.0	0.005
SPX2	162K	808K	64	40.5M	2.0M	20.6	39.2	42.0	0.009
SPXF	19K	12K	192	14.7M	2.0M	7.1	20.8	30.2	0.014
Blunt	40K	183K	64	10.0M	552K	18.6	41.7	44.4	0.005
Torso	8K	50K	360	11.2M	1.0M	11.4	20.5	28.1	0.002
TJet	160K	1M	150	93.8 M	2.7M	34.7	5.3	17.9	0.204

original and quantized values are measured. Results show that procedurally generated datasets have a higher SNR and smaller discrepancy, since they have a smoother variation in their scalars over time. The TJet dataset has smaller SNR values because it represents a real fluid simulation, but it also led to higher compression ratios due to its fixed codebook sizes. In general, the quality of the compression corresponds with the variance in the scalars between steps. Thus, datasets with smoother transitions result in less compression error. A limitation of the current approach is that because it exploits temporal coherence, it may not be suitable for datasets with abrupt changes between time instances. In this case, compression methods that focus more on spatial coherence may be necessary.

In addition to the numerical compression results described above, the image quality for all datasets was evaluated by comparing them against the rendering from uncompressed data. For most datasets the difference in image quality was minimal. However, for the TJet dataset (the one with the smaller SNR values), there are some small differences that can be observed in close-up views of the simulation (see Figure 6.5).


(a) (b) (c)

(d) (e) (f)

Figure 6.5. Comparison of direct volume rendering using (a) uncompressed and (d) compressed scalars on the TJet dataset(1M tetrahedra, 150 time steps). Level-of-Detail is compared at 5% for the TJet and 3% for the Torso dataset(50K tetrahedra, 360 time steps) using (b)(c) a local approach and (e)(f) a global approach.

6.2.3 Rendering

The rendering system allows the user to interactively inspect the time-varying data, continuously play all time instances, and pause or even manually select a given time instance by dragging a slider. Level-of-detail changes dynamically to achieve interactive frame rates, or can be manually set using a slider. Changing from direct volume rendering to isosurfacing is accomplished by pressing a key. Similarly, the isovalue can be interactively changed with the press of a key and incurs no performance overhead.

Rendering time statistics were produced using a fixed number of viewpoints. In Table 6.2 timing results for the experimental datasets are shown and in Figure 6.6 quality results for different time steps of two datasets are shown. To compare the overhead of the system with the original HAVS system that handles only static data, the rendering rates are also measured for static scalar fields without multithreading. The dynamic overhead is minimal even for the larger datasets. In fact, for some datasets, the multithreading approach is faster with dynamic data than single threading with static data. Note that for the smaller datasets, frame-rates greater than 60 frames per second are not reported since it is difficult to accurately measure higher rates.

6.2.4 Level-of-Detail

The sample-based level-of-detail for time-varying scalar fields computes the importance of the faces in a preprocessing step that takes less than two seconds for the global strategy or for

Table 6.2. Performance measures for static and time-varying (TV) scalar fields for direct volume rendering (DVR) and isosurfacing (Iso). Static measurements were taken without multithreading. Performance is reported for each dataset with object-space sorting (during rotations) and without object-space sorting (otherwise).

Mesh	**Sort**	**DVR Static FPS**	**DVR TV FPS**	**DVR TV Tets/s**	**Iso Static FPS**	**Iso TV FPS**	**Iso TV Tets/s**
SPX1	Y	24.2	32	3.3M	26.4	28.2	2.9M
SPX1	N	42.6	41.7	4.3M	51.1	43.5	4.5M
SPX2	Y	2.8	2.9	2.4M	2.9	2.9	2.4M
SPX2	N	7.6	7.5	6.2M	8.2	8.2	6.8
SPXF	Y	>60	>60	0.7M	>60	>60	0.7M
SPXF	N	>60	>60	0.7M	>60	>60	0.7M
Blunt	Y	16.1	20.4	3.8M	15.9	19.5	3.6M
Blunt	N	25.6	27.5	5.2M	31.2	31.1	5.8M
Torso	Y	40.6	31.2	1.6M	44.8	31.2	1.6M
Torso	N	>60	>60	3.1M	>60	>60	3.1M
TJet	Y	2.3	2.1	2.1M	2.2	2.4	2.4M
TJet	N	6.1	5.8	5.8M	5.7	6	6.0M

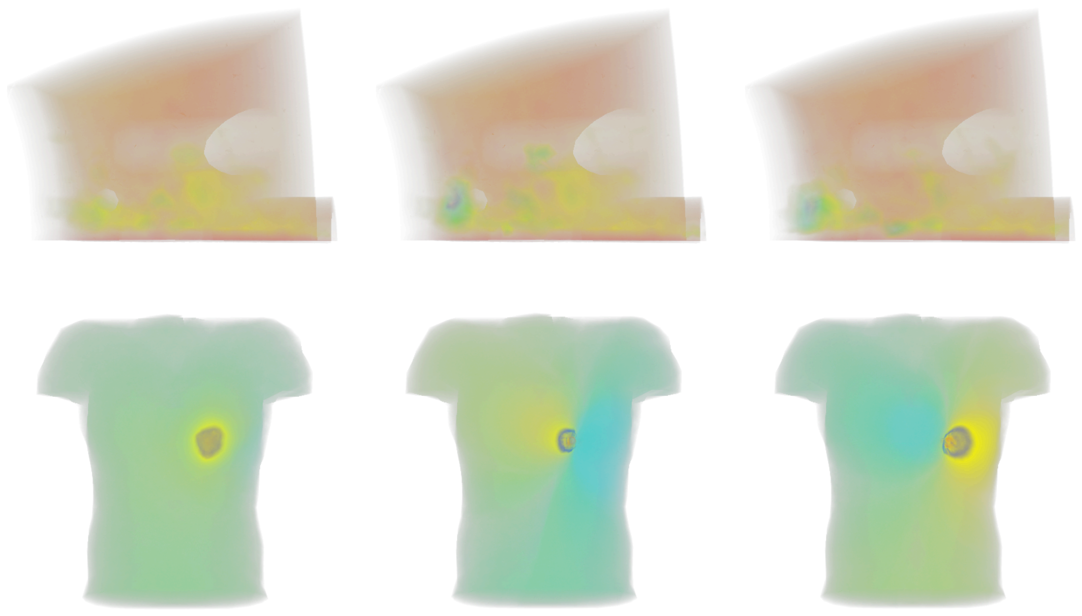


Figure 6.6. Different time instances of the SPXF (above, 12K tetrahedra and 192 time steps) and Torso (below, 50K tetrahedra and 360 time steps) datasets volume rendered at full quality.

the local strategy using six intervals, even for the largest dataset in the experiments. In addition, there is no noticeable overhead in adjusting the level-of-detail at a per frame basis because only the number of triangles in the current frame is computed [21]. Figure 6.4 shows the results of the global level-of-detail strategy on the Torso dataset at decreasing levels-of-detail. Figure 6.5 shows a comparison of the global and local sampling strategies. To capture an accurate comparison, the local sampling results are shown in the middle of an interval, thus showing an average quality. In the experiments, the frame-rates increase at the same rate as the level-of-detail decreases (see Figure 6.4) for both strategies.

6.3 Discussion

An important consideration for the framework presented is the scalability of the solution on current and future hardware configurations. Development of faster processors is reaching physical limits that are expensive and difficult to overcome. This is leading the industry to shift from the production of faster single-core processors to multicore machines. On the other hand, graphics hardware has not yet reached these same physical limitations. Even so, hardware vendors are already providing multiple GPUs along with multiple CPUs. The use of parallel technology ensures that the processing power of commodity computers keeps growing independently of some physical limitations, but new applications must be developed considering this new reality to take full advantage of the new features. In particular, for data-intensive graphics applications, an efficient load balance needs to be maintained between resources to provide interactivity. In this chapter, the focus was this multicore configuration that is becoming increasingly popular on commodity machines instead of focusing on traditional CPU clusters.

To take advantage of multicore machines with programmable graphics hardware, the computation was separated into three components controlled by different threads. For the largest dataset in the experiments, the computation time for the three components is distributed as follows: 2% decompression, 55% sorting, and 45% rendering. For the smaller datasets, the processing time shifts slightly more towards rendering. With more available resources, the rendering and the sorting phases could benefit from additional parallelization.

To further parallelize the sorting thread, the computation could be split amongst multiple cores on the CPU. This could be done in two ways. The first is to use a sort-first approach that performs parallel sorting on the geometry in screen space (see Gasarch et al. [2]), then pushes the entire geometry to the graphics hardware. The second is a sort-last approach that breaks the geometry into chunks that are sorted separately, and sent to the graphics hardware for rendering and compositing (see Vo et al. [137]).

Because of the parallel nature of modern GPUs, the vertex and fragment processing automatically occurs in parallel. Even so, multiple-GPU machines are becoming increasingly common. The effective use of this technology, however, is a complex task, especially for scientific computing. Some tentative tests of the framework have been performed on a computer with an NVidia SLI configuration. SLI, or Scalable Link Interface, is an NVidia technology developed to synchronize multiple GPUs (currently two or four) inside one computer. This technology was developed to automatically enhance the performance of graphics applications, and offer two new operation modes: Split Frame Rendering (SFR) and Alternate Frame Rendering (AFR). SFR splits the screen into multiple regions and assigns each region to a GPU. AFR renders every frame on a different GPU, cycling between all available resources. The experiments with both SFR and AFR on a dual SLI machine did not improve performance with this volume rendering algorithm. The SFR method must perform *k*-buffer synchronization on the cards so frequently that no performance benefit is achieved. With AFR, the GPU selection is controlled by the driver and changes at the end of each rendering pass. With a multipass rendering algorithm, this forces synchronization between GPUs and results in the same problem as SFR. In the future, as more control of these features becomes available, multiple-GPU machines should be easier to take advantage of to improve performance.

6.4 Summary

Rendering dynamic data is a challenging problem in volume visualization. In this chapter, a description was provided of how time-varying scalar fields on unstructured grids can be efficiently rendered using multiple visualization techniques with virtually no penalty in performance. In fact, for the larger datasets in the experiments, time-varying rendering only incurred a performance penalty of 6% or less. Vector quantization can be employed with minimal error to mitigate the data transfer bottleneck while leveraging a GPU-assisted volume rendering system to achieve interactive rendering rates. The algorithm exploits both the CPU and GPU concurrently to balance the computation load and avoid idle resources. In addition, new time-varying approaches were introduced for dynamic level-of-detail that improve upon existing techniques for static data and allows the user to control the interactivity of the animation. The proposed algorithm is simple, easily implemented, and most importantly, it closes the gap between rendering time-varying data on structured and unstructured grids. This is the first system for handling time-varying data on unstructured grids in an interactive manner.

CHAPTER 7

APPLICATION: A FRAMEWORK FOR TRANSFER FUNCTION DESIGN

Creating insightful visualizations from both simulated and measured data is an important problem for the visualization community. For scalar volumes, direct volume rendering has proved to be a useful tool for data exploration. With the use of a transfer function, scalar values can be mapped to colors and opacities to identify and enhance important features. Though some automatic techniques have been developed for transfer function specification [61, 48], the exploration process still involves tuning the parameters manually until the desired visualization is produced. A great deal of research has recently been performed to assist the user in this specification task with interactive widgets [62, 64]. These tools generally assist the user by allowing them to create and manipulate widgets over one or more dimensions of histogram information of the data.

Even with current tools, the specification of transfer functions is not a trivial task [104]. The primary obstacle is the diversity of datasets to be rendered. A tool that excels at extracting features from a structured grid of scanned medical data may have difficulty finding relevant features in an unstructured grid of simulation data. The use of multidimensional transfer functions may be helpful for some datasets, but may just complicate the specification in others. This problem is further compounded when different features in a dataset are enhanced by different histograms. Another difficulty is that the datasets may contain a high dynamic range of floating point scalar values that vary over time. General tools to efficiently handle the diversity of data that can be encountered in both medical and scientific domains are not currently available.

As an example, consider the ubiquitous Utah Torso dataset—an unstructured grid containing simulated electric potential in the human torso. In this example, a version of the dataset is used consisting of about 50,000 tetrahedra and 360 floating point time steps of a rotating dipole that emphasizes the simulation results. Figure 7.1 illustrates several visualizations of these data through direct volume rendering. There are several issues that are encountered when attempting to explore this data using transfer functions. First, for each time step of the data, the scalars in

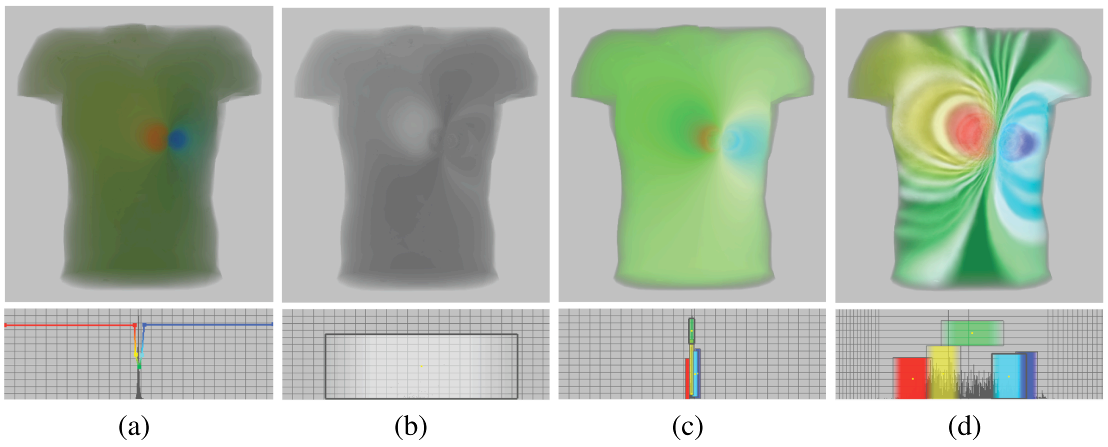


Figure 7.1. An example demonstrating some of the capabilities of the proposed system for transfer function design using the Utah Torso dataset. (a) Traditional transfer function specification using a simple polyline widget over the scalar histogram shows positive and negative potentials in the simulation. (b) Other features such as the variation of the scalars over time can only be expressed using more advanced histograms such as this 2D histogram of the magnitude of the variation. (c) Transfer function *ensembles* provide user-specified blending operators between histograms defined on in both 1D and 2D, in this case a 1D scalar histogram and a 2D time histogram. (d) By remapping the scalar range from the previous ensemble, high dynamic range details can easily be explored, resulting in a more insightful visualization.

the volume are concentrated in one peak in the histogram (i.e., 83% of the scalars fall into 1% of the scalar range). Thus, with traditional specification tools such as a polyline defined over the histogram, feature finding may be difficult because much of the data maps to few entries in the corresponding color and opacity lookup table (see Figure 7.1(a)). A zooming interface on the color map (e.g., Yuan et al. [153]) will facilitate the placement of the specification widget, but will not show additional features due to the static resolution of the lookup table. The second issue that occurs in the specification is that some features of the data can only be found using one type of histogram. On the Utah Torso, positive and negative potentials can be determined with a polyline or rectangle widgets on the scalar histogram (see Figure 7.1(a)), but the amount of change in the scalars over time requires a time-sensitive 2D histogram (see Figure 7.1(b)). Available techniques that merge transfer functions (e.g., Wu et al. [148]) are not designed to provide user control of the blending of 1D and 2D transfer functions. The final issue that is encountered during transfer function specification is that limited control is currently available for specifying transfer functions for time-varying data. The ability to transition between user-defined transfer functions temporally is essential for tracking features through time or to provide temporal focus and context animations.

The goal of this research is to create a tool that facilitates feature extraction through transfer function specification for disparate data types by addressing each of these issues. The proposed system builds on more than a decade of previous work by unifying existing methods as well as introducing new techniques for specifications that can be used independently or in unison for volume visualization of complex data. In this chapter, the focus is on transfer function generation for unstructured volumes due to their complexity and disparity, but the algorithms described are general enough to be applied to structured data as well. In particular, the contributions include:

- An interactive system is described that facilitates data exploration of diverse data types by combining existing techniques with new ones;
- An algorithm is introduced for simplifying feature extraction in high dynamic range datasets by allowing the interactive, nonlinear remapping of the scalar range in the histogram;
- The notion of transfer function *ensembles* is introduced to allow a user-controlled blending of multiple transfer functions defined on different histograms—each representing different features;
- Transfer function specification is discussed across multiple steps of time-varying data and a tool is introduced for blending ensembles of transfer functions over time through user-controlled keyframing;
- The results of a user study of the system are provided in the form of an expert review.

Figure 7.1(c) and (d) demonstrate these contributions on the example dataset. Figure 7.1(c) provides the results of blending transfer functions defined on different histograms. In this case, it uses an additive blending function to combine a 1D scalar histogram defined using rectangle widgets (Figure 7.1(c) bottom) with a 2D time histogram that uses the variation of scalars over time to emphasize the changing regions in the volume. Thus multiple, distinct features are visible in a single visualization. Figure 7.1(d) shows the results of distributing the scalars more evenly through the color and opacity lookup table by dynamically remapping them. Previously unseen features (such as the abrupt changes potentials) become visible by taking advantage of a nonlinear mapping of the scalars. Finally, using a keyframing interface, ensembles of transfer functions (such as the one shown in Figure 7.1(d)) can be assigned to specific time steps and smoothly transitioned during animations using user-defined step functions. The overall result is a tool that allows insight into the data that is not otherwise available with established tools and techniques.

The rest of the chapter is outlined as follows. Section 7.1 provides an outline of the system and tools for transfer function specification. In Section 7.2 range mapping for high dynamic range volumes is introduced. In Section 7.3 the merging of multiple transfer functions is discussed. Section 7.4 outlines an algorithm for blending transfer function ensembles over time. An evaluation of the methods is described in Section 7.5 followed by a discussion in Section 7.6. Finally, a summary of this work is provided in Section 7.7.

7.1 System Overview

In this section a summary of the proposed system for transfer function specification is given. It comprises several tools that aim to provide useful insights about the underlying data and help the user during the creation of transfer functions. An overview of the visualization process, as provided by the proposed system, is shown in Figure 7.2. First, the volumetric data are loaded and the appropriate visualization algorithm is selected depending on the data type. Currently, unstructured time-varying grids are supported. Next, the task of data exploration is performed by finding features in the data with the assistance of histograms and specifying color and opacity using interactive widgets. A variety of specialized 1D and 2D histograms and corresponding color and opacity widgets are available to highlight regions of interest in the volume. In addition, fine details in high-dynamic range volumes can be further explored by directly manipulating the spacing of the scalars in the histograms through range-mapping. This process of data exploration through transfer function specification continues until a desired set of features is distinguished in the volume through one or more transfer functions (potentially defined on different histograms). The next step of the visualization process allows the user to combine these transfer functions into

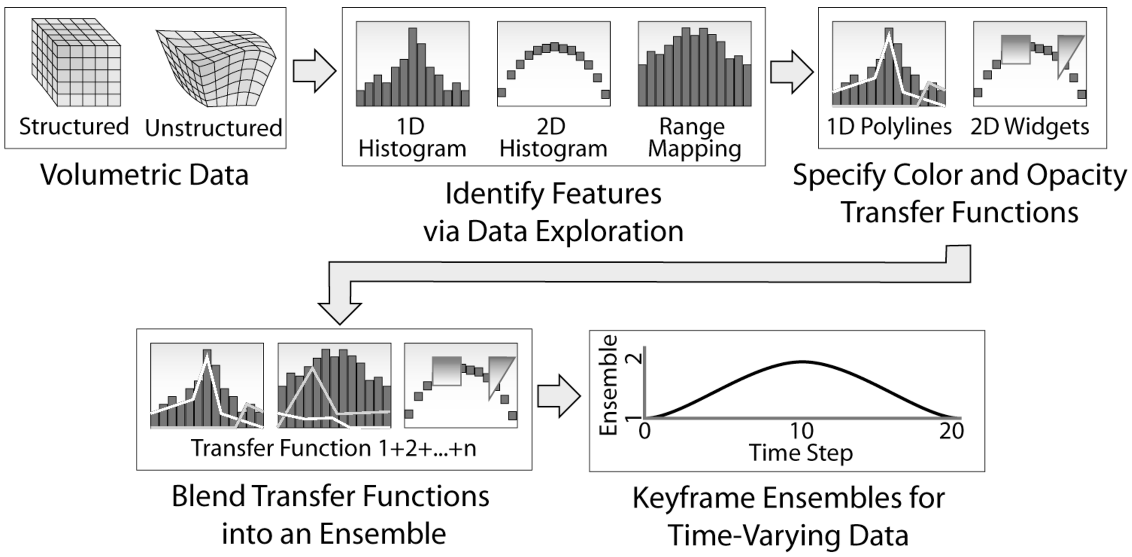
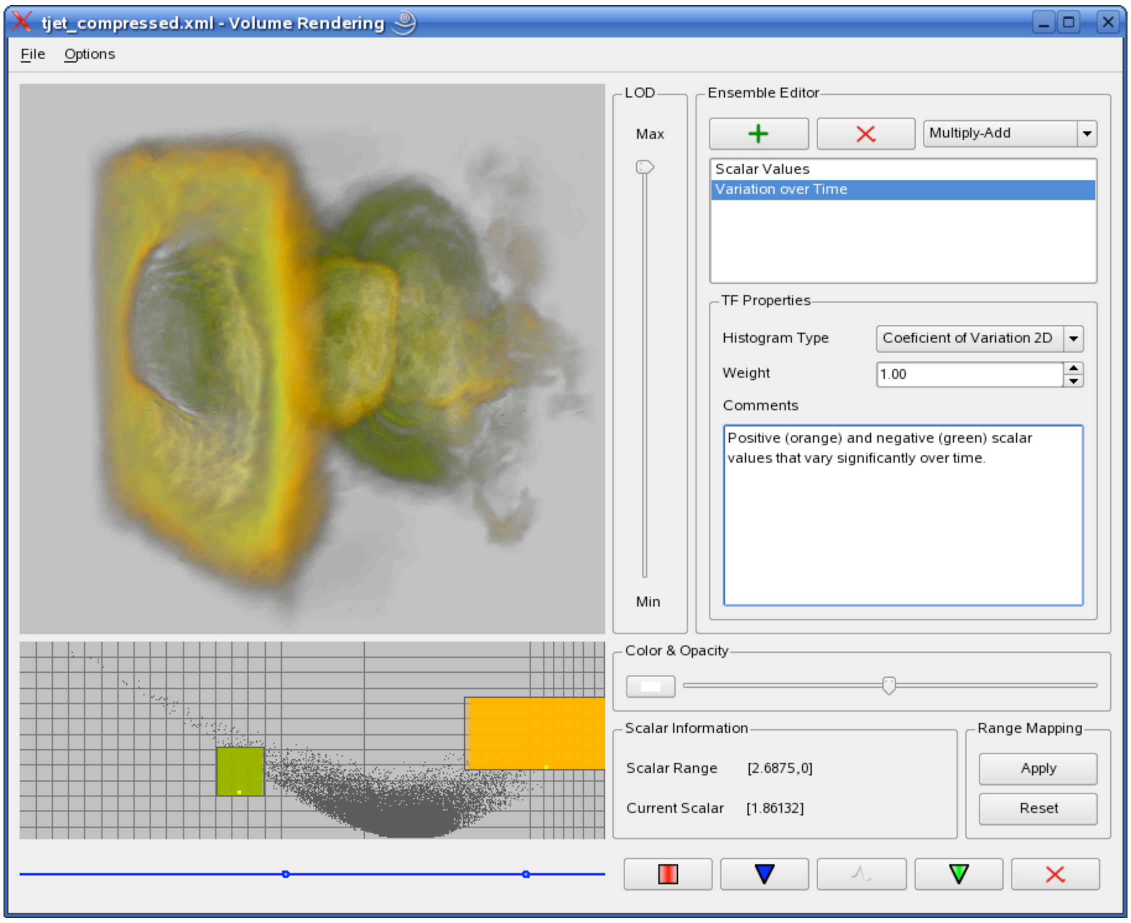

Structured
Unstructured
Volumetric Data
1D Histogram
2D Histogram
Range Mapping
Identify Features via Data Exploration
1D Polylines
2D Widgets
Specify Color and Opacity Transfer Functions
Transfer Function 1+2+...+n
Blend Transfer Functions into an Ensemble
Ensemble
1
2
0
10
20
Time Step
Keyframe Ensembles for Time-Varying Data

Figure 7.2. The visualization process provided by the proposed system.

ensembles using user-defined blending operators. Finally, these ensembles can then be keyframed with user-specified transitions for time-varying data.

The user interface for the system is shown in Figure 7.3. The upper left window of the interface displays the volume rendering described by the current transfer function specification. Though the transfer function tools described are general enough for any data type, these tools were developed primarily for unstructured grids. The volume rendering is performed using the Hardware-Assisted Visibility Sorting (HAVS) algorithm for unstructured grids [22] with dynamic level-of-detail [21] and support for time-varying data [6, 7], as described in Chapter 6. Partial preintegration is used to reduce the overhead of dynamic updates to the transfer function [94].

Statistical analysis is important to capture hidden aspects of the data. Thus the system offers several histogram options. In addition to a basic 1D scalar histogram, also provided are a 2D histogram that includes the gradient magnitude. For time-varying data, 1D and 2D histograms



Multiply-Add

Histogram Type: Coeficient of Variation 2D

Weight: 1.00

Comments: Positive (orange) and negative (green) scalar values that vary significantly over time.

Scalar Range [2.6875,0]

Current Scalar [1.86132]

Figure 7.3. The user interface for interactive transfer function specification is shown for the time-varying Turbulent Jet dataset using a 2D time histogram.

are also provided that are based on the coefficient of variation calculated from the scalar time steps [58]. Additional histograms can also be incorporated into the framework. The active histogram is displayed in the lower left window of the interface as shown in Figure 7.3.

Similar to Kniss et al. [62], interaction with the scalar histograms is performed by defining widgets that represent color and opacity on the histogram. Four types of widgets are exposed to the user, some of which are shown in Figures 7.1 and 7.3. The first widget is a rectangle defining a single color and opacity ramp and is used in both 1D and 2D histograms. In 1D, the vertical position does not change the transfer function but is useful for manipulating overlapping widgets. The second widget is a triangle defining a single color used for 2D histograms. The third widget is a triangle widget with a fall-off, similar to the one described by Kniss et al. [62], to emphasize boundaries. This widget is primarily used in 2D gradient magnitude histograms. Finally, the last widget is a polyline, where the control points denote colors and the vertical placement of the points controls opacity. This final widget is used only in 1D histograms.

7.2 Scalar Range Mapping

Volume data produced from scientific simulation typically contain a high dynamic range (HDR) of floating point scalar values. In addition, a high percentage of the scalar values are often contained in a small range of the histogram (see Figure 7.4(a)). Consequently, to expose details that may be contained in these small regions, a large number of control points and a high resolution lookup table are required. There are two main issues with traditional transfer function design when dealing with HDR data. First, the narrow range of values makes specification difficult due to the low resolution of the features on the histogram interface. Second, the limited resolution of the color and opacity lookup table in graphics hardware is not sufficient to fully represent all the unique scalar values in the data.

To overcome the resolution limitations of the histogram interface, tools like ParaView [99] incorporate user-controlled zooming widgets to assist with transfer function specification over small regions of the data. Yuan et al. [153] recently introduced a 1D fish-eye visualization of the histogram based on a *focus and context* concept, that allows simultaneous representation of global (context) and detail (focus) information on the same histogram display. These approaches, based on magnifying the range of interest in the user interface, greatly assist the user with HDR transfer function design. However, the second issue has received very little attention even though it is of equal, if not greater, importance for feature extraction in HDR volumes. Ideally, the number of entries in the color and opacity lookup table should correspond to the number of unique scalar values in the volume. Yuan et al. [153] leverage tone mapping and specialized high-precision graphics hardware to handle the high precision of texture based volume rendering. With limits in texture size, this is not always sufficient and may result in many scalar values being assigned to one entry in the table (see Figure 7.4(b)). Instead, a *range mapping* is proposed that redistributes the scalar range nonlinearly to spread the regions of interest more evenly across the lookup table (see Figure 7.4(c)). Range mapping is related to histogram equalization, a common approach in image processing for handling low contrast images. This feature facilitates the design process by allowing focus and context zooming effects, while avoiding resolution issues of a fixed-size lookup table. The result is a tool naturally capable of extracting detailed features in the data, as shown in Figure 7.5.

Based on the observation that the transfer function design difficulties of HDR data are mainly due to the nonuniform distribution of scalar data, the solution is to redistribute the scalar range. This can be done automatically by performing histogram equalization, as was proposed by Chourasia and Shulze [26], which spreads out the clustered regions. Mathematically, histogram equalization is performed by introducing a cumulative density function (CDF) as a sum of probability

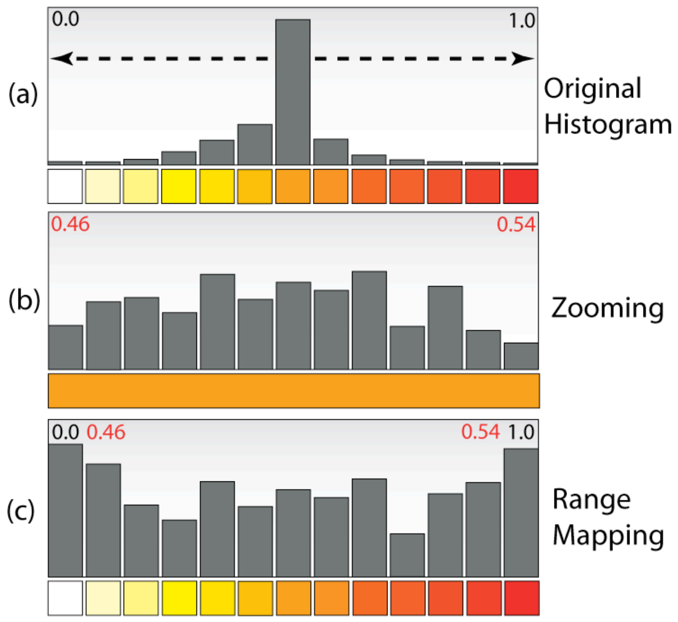


Figure 7.4. A description of how high-dynamic range data affect a lookup table. (a) A high-dynamic range histogram (above) and corresponding lookup table (below) are shown. (b) Zooming into the dense region of the histogram does not change the resulting image due to the static resolution of the lookup table. (c) By range mapping the scalar values, the high-dynamic range elements of the mesh can be spread more evenly across the static lookup table, enhancing hidden features in the data.

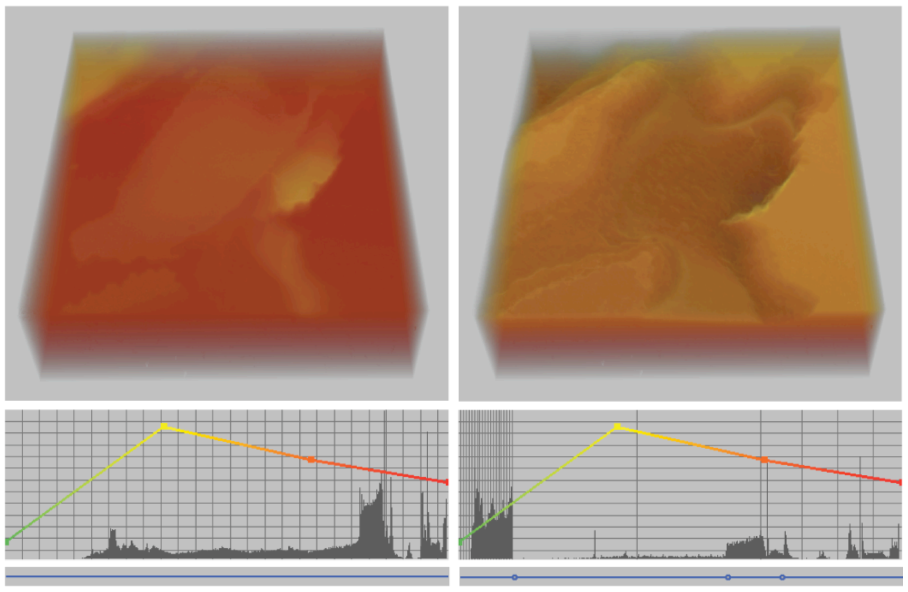


Figure 7.5. Volume exploration of the San Fernando earthquake simulation through range mapping. A predefined transfer function (left) is used to explore the data by remapping the scalars (right). Only a nonlinear remapping can enhance features that are hidden in multiple spikes of the data.

density functions (PDFs) over normalized scalar inputs:

$$CDF(x_i) = \sum_{x_j < x_i} PDF(x_j). \tag{7.1}$$

Then, a simple mapping is performed on the normalized scalar input value x that yields a new uniformly distributed normalized output y:

$$y = CDF(x). \tag{7.2}$$

Due to its speed and simplicity, it is common to use a discrete histogram equalization and perform this mapping with a lookup table. This approach is automatic, but gives the user very little control over the redistribution process and tends to break the continuities of the scalar range. Range mapping, a generalization of histogram equalization, is based on piecewise linear mapping functions and provides more control while maintaining the continuity of the scalar range. The range mapping functions that map the input scalars $[x_0 \ldots x_n]$ to a new scalar range $[y_0 \ldots y_n]$ are a class of piecewise continuous functions f over the input range that satisfy the following conditions: f is a monotonically increasing function, $f(x_0) = y_0$, and $f(x_n) = y_n$. Similar to histogram equalization, the new scalar value y is computed as:

$$y = f(x). \tag{7.3}$$

Given this definition, the function can arbitrarily redistribute the scalar range while maintaining the order and continuity.

In practice, the linear range mapping functions that combine many line segments are used, each of which performs mapping from a specific range $[x_i \ldots x_{i+1}]$ to $[y_i \ldots y_{i+1}]$ by applying the linear mapping equation:

$$y = \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i) + y_i. \tag{7.4}$$

These linear functions are sufficient to represent all range mappings since any function can be approximated using many piecewise linear functions.

Because the cost of the linear interpolation is relatively low, range mapping can be performed interactively while the user is manipulating control points for the range. The remapping process is performed in hardware by storing a 1D texture that contains one entry for every control point of the remapping. When the mapping changes, to minimize CPU to GPU transfer, only the new mapping texture is sent to the GPU. During rendering, the normalized scalar values can then be remapped to normalized scalar values using a single texture lookup with linear interpolation enabled. Thus, this extra remapping step minimally impacts the rendering performance and is flexible enough to be used in a variety of volume rendering algorithms.

As illustrated in Figure 7.4, range mapping yields a magnification that is different from a normal zooming effect, since the actual shape of the histogram changes nonlinearly. This helps the user exploit the real data distribution in narrow clusters of the scalar range. Even without transfer function widgets, range mapping can be a powerful exploration tool. Figure 7.5 shows how range mapping can be used to explore the data using a simple, predefined transfer function.

Creating a user interface that can fully exploit the power of range mapping is a challenge. The solution is a simple, intuitive interface that allows the user to choose the range by adding control points and extend the range by dragging two control points away from each other. This allows the user to continue adding control points between the previous points to further probe important regions. The histogram and volume rendering change interactively during this control point manipulation to provide visual feedback of the remapping. In addition, the range and scalar value under the cursor are displayed to the user to facilitate specification when there is *a priori* knowledge of the data. Figures 7.3 and 7.5 show snapshots of this interface.

7.3 Transfer Function Ensembles

Designing transfer functions that effectively enhance volume characteristics is a difficult task because the exploration of the transfer function space can be unintuitive[62]. Volume statistics may provide meaningful insights about data and aid users during the specification process, but a single statistical measure may not reveal all the important features in the data. The use of multiple histograms for transfer function specification that highlight different features in the data is advocated. These transfer functions are then merged using weighted blending operations to produce a single, complex transfer function that are called an *ensemble*. Ensembles provide the ability to derive completely new transfer functions from a collection of existing ones.

The idea of combining transfer functions is not new [83, 64, 148, 146]. Previous work in the area has concentrated on merging transfer functions defined on the same histogram space using both linear and nonlinear combinations of transfer functions. The proposed approach with ensembles is to allow more flexibility in the combination process by providing boolean-like blending operations on transfer functions defined on multiple spaces of a single volume. The only limitation to these transfer functions is that they share one common space (i.e., scalar value). Thus, an ensemble represents a new transfer function that is created by aggregating a collection of transfer functions using different blending operations.

For 1D histograms that define a transfer function on the scalar value blending is straightforward. The transfer function widgets directly map to an RGBA lookup table that is used for rendering. Blending multiple transfer functions is then just a texture compositing of RGBA values. Extending this to other spaces (i.e., gradient magnitude or coefficient of variation) requires an additional dimension for each space. Unfortunately, it is not feasible to extend beyond three dimensions due to constraints on textures in hardware. Thus, these combinations are limited to histograms that have three unique dimensions that do not share a common space. This has been found to be adequate for the volumes used in the experiments described in this chapter.

Another approach for combining multiple transfer functions defined in multiple spaces is to map them to a single common space. The extra dimension in the 2D case is then used to modulate the intensity of the color. This is done by determining the number of scalar values within a 2D widget and the number outside and using this fraction to reduce the intensity accordingly. Though not nearly as powerful as true multidimensional transfer functions, this heuristic tends to produce acceptable results in practice and allows all the histograms to be reduced to a common space. For instance, the combination of transfer functions in 1D and 2D shown in Figure 7.1 were performed using this heuristic.

Through the use of ensembles, users can intuitively explore features enhanced by different transfer functions. Complex mappings can be generated by aggregating several simpler transfer functions. Another application is the contextualization of the entire volume through one common transfer function, while different specific transfer functions focus only on important features.

Different strategies can be interactively swapped, providing a fast tool for exploring volume datasets. The system also allows users to load custom transfer functions and combine them with new transfer functions designed with the tool. In this framework, the ability is provided to specify multiple ensembles of transfer functions and manage different visualizations for a volume.

7.3.1 Blending Strategies

Several weighted blending strategies are provided for combining transfer functions. These blending functions are illustrated in Figure 7.6. Three types of operations are available, though others are easily incorporated. Each transfer function is assigned a weight that corresponds to its intensity contribution in the resulting ensemble. This weight is user-controlled and can be used to provide emphasis in the most important regions. Each transfer function is multiplied by its weight prior to blending with other transfer functions. The examples given below describe the blending process for two weighted transfer functions. If more than two transfer functions are provided then the compositions are performed sequentially. The following principle blending operations are utilized:

1. **ADD**: This is the most common blending operation in which transfer functions are summed together. The result r for color C and opacity α of a lookup table entry i when combining transfer function 1 and 2 can be expressed as the following:

$$\begin{aligned} C_r(i) &= C_1(i) + C_2(i) \\ \alpha_r(i) &= \alpha_1(i) + \alpha_2(i) \end{aligned}$$

   Optionally, the color contributions can be weighted by the opacity before the addition. This mode is useful for combining features that are unique to two transfer functions, producing a single transfer function that emphasizes all features of both transfer functions. Figure 7.6(c) shows the result of adding two transfer functions to combine their enhanced characteristics into a single image.

2. **AND**: This blending operation enhances features that are shared by two transfer functions. The result of this combination of transfer functions can be expressed using the same notation as above:

$$\begin{aligned} C_r(i) &= Max(C_1(i), C_2(i)) \\ \alpha_r(i) &= Min(\alpha_1(i), \alpha_2(i)) \end{aligned}$$

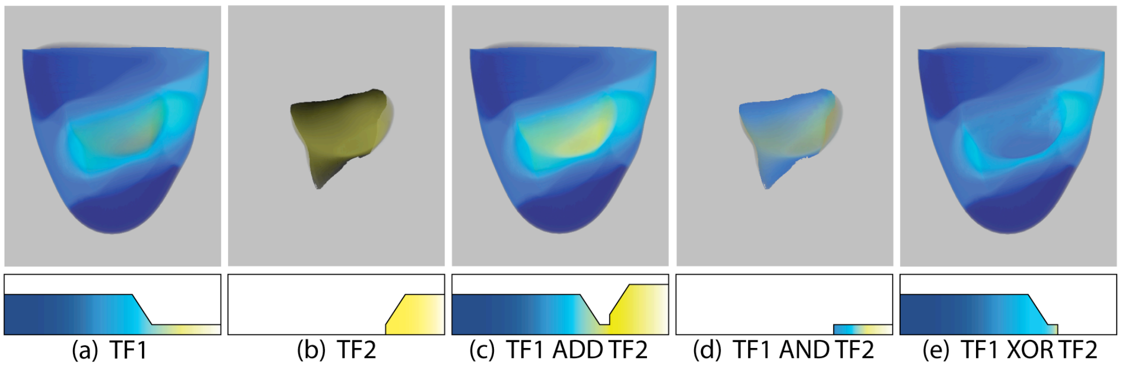


Figure 7.6. An ensemble is used to combine multiple transfer functions with different blending strategies for an unstructured dataset of the heart (top: volume rendering result; bottom: illustration of the operation in transfer function space, where the scalar value is mapped to the horizontal axis, color is defined inside the function, and opacity is defined by the height of the function). (a) A single transfer function shows all features of the volume. (b) A more specific transfer function focuses on one detail. (c) Adding both transfer functions combines the results. (d) The *AND* operation emphasizes common features of both transfer functions. (e) The *XOR* operation removes common features from the two transfer functions.

The maximum of the RGB channels (independently) is used to maintain the color intensity and the minimum alpha to remove regions that are not common. As before, the color contribution can optionally be weighted by the opacity. Figure 7.6(d) shows the result of an *AND* operation in the heart dataset. This feature is useful for determining common properties between regions of interest, in this case it emphasizes the region that contains a sensor.

3. **XOR**: This blending operation removes common areas from two transfer functions. The result of the *XOR* operation on two transfer functions can be expressed using the same notation provided above:

$$C_r(i) = (C_1(i) \wedge \overline{C_2(i)}) \vee (\overline{C_1(i)} \wedge C_2(i))$$
$$\alpha_r(i) = (\alpha_1(i) \wedge \overline{\alpha_2(i)}) \vee (\overline{\alpha_1(i)} \wedge \alpha_2(i))$$

In other words, a color and opacity contribution defined by a transfer function is preserved only if no other transfer function assigns a color and opacity value to it. This is a good strategy to remove features from existing transfer functions. Figure 7.6(e) shows an example of removing the region of the heart that contains a sensor.

Combining different transfer functions is a straightforward procedure for creating complex visualizations. For instance, in medical imaging data, one can create individual transfer functions that enhance different organs (heart, lung, liver, etc.) and combine them all in different ways using *ADD* to provide a series of visualizations. The subtraction capabilities of *AND* and *XOR* also allow the user to create unique transfer functions from existing ones.

7.4 Time Varying Datasets

Transfer function design and generation for time-varying datasets has received little attention in the research community. Some of the available literature focuses on defining transfer functions that are applied globally to all time steps [84, 1]. However, these global approaches may not be sufficient for all types of volumes. Datasets like the Utah Torso, shown in Figure 7.1, contain regular time-varying patterns that can be readily inspected with the use of a single ensemble of transfer functions. This regularity is demonstrated by the relatively static appearance of the histogram as the time steps progress. Unfortunately, all volumes do not demonstrate this same temporal regularity or periodicity. For example, the Five Jets dataset, shown in Figure 7.7, has a large variance in the histogram as the time steps progress. This results in moving features that cannot be enhanced with a single transfer function. To address this problem, several methods have been introduced to change the transfer function over time to track these moving features. One approach is to keyframe derived transfer functions and linearly interpolate between them [58]. More recent work suggests that using machine learning to build adaptive transfer functions between keyframes is superior than a simple linear interpolation [132].

The proposed work concentrates on easing the burden of specifying transfer functions that change across time by building on previous keyframing concepts. The system allows manipulation of raw or compressed datasets and provides controls to play, pause, and stop animations (see Figure 7.8). As with previous techniques temporal histograms are provided for capturing

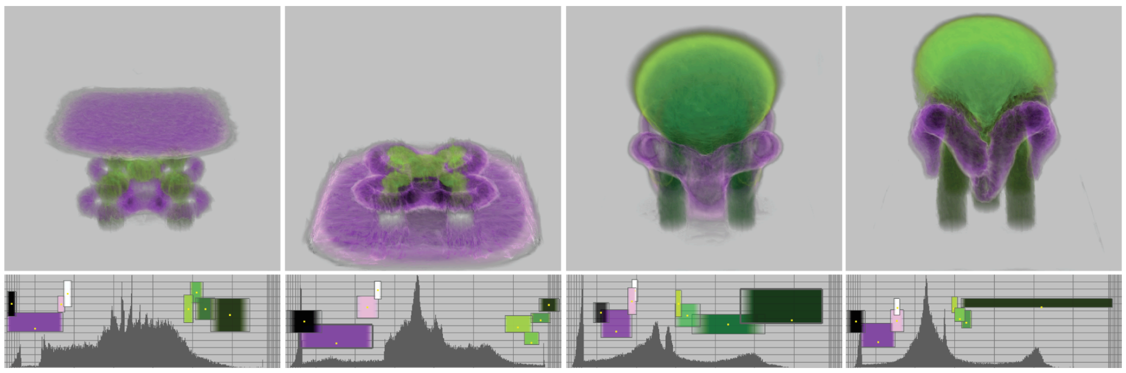


Figure 7.7. Multiple 1D scalar transfer functions are used during different time steps of a time-varying simulation of five jets. Keyframing can be used for temporal focus and context visualization as well as a means of highlighting features that may move through the scalar space over time (as shown above).

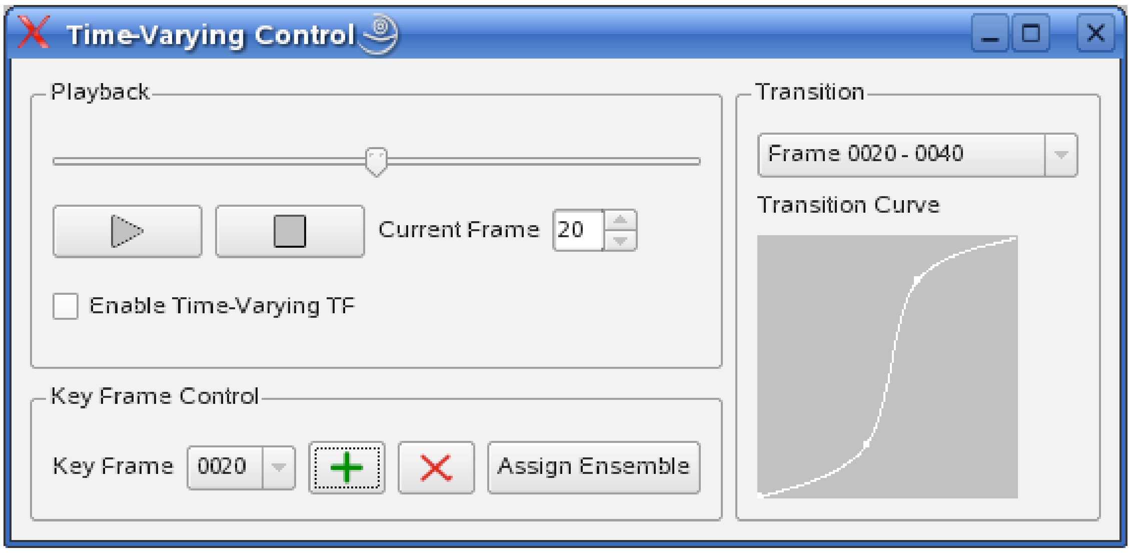


Figure 7.8. The time-varying interface: playback control (upper left), keyframes control (lower left) and interpolation control (right).

global information about the changing data. In addition, the histograms in other spaces change interactively to reflect the current time step. This feedback gives the user greater understanding of the underlying structure of the temporal data. It also allows the user to specify transfer function ensembles that are unique to specific time steps.

To handle temporally irregular volumes, a user-controlled keyframing approach is used for transitioning between ensembles of transfer functions. The interface provides the user with the ability to identify frames in the animation that require changes, specify a new ensemble of transfer functions, and assign the ensemble to the frame. To avoid the visual discontinuities that occur when changing from one keyframe to the next, an editable transition curve is provided that specifies the blending between adjacent keyframes. An ensemble is represented in hardware as a color and opacity lookup table that was composed using one or more transfer functions (see Section 7.3). Blending between multiple ensembles for time-varying data is performed in a similar manner. For every color C and opacity α entry i in the new lookup table, the resulting interpolation r between two ensembles (1 and 2) can be determined from the transition curve $f(t)$ as follows:

$$C_r(i) = C_1(i)(1 - f(t)) + C_2(i)f(t)$$
$$\alpha_r(i) = \alpha_1(i)(1 - f(t)) + \alpha_2(i)f(t)$$

The horizontal axis of the transition curve represents the time interval between the two keyframes. The weight of the interpolation between ensembles is described by the vertical axis. For improved feature tracking, adaptive transfer functions [132] could used with the transition curves by replacing interpolation.

Combining keyframed ensembles with transition curves provides an effective solution for capturing moving features in dynamic datasets while maintaining smooth animations. Keyframing can also be used as a temporal focus and context method for exploring different aspects of the data during different time intervals, similar to an existing approach for static volumes [147]. The detailed control of the temporal aspect of transfer function specification can greatly benefit researchers who have been performing time-varying visualization by providing more control over the animation than is available with current tools.

7.5 Evaluation

An important consideration for a transfer function specification tool is that it does not introduce additional computational overhead and thus adversely impact interactivity. There is no measurable performance penalty when using the transfer function specification tools. Thus, the interactivity of the rendering remains the same as the original volume renderer.

To evaluate the usefulness of the proposed techniques, an informal expert evaluation of the system was performed. Expert reviews have been shown to be a useful means of evaluation, and require fewer reviewers than standard user studies [130]. Obviously, completely automatic techniques for creating images from volumes would be ideal. Unfortunately, the algorithms to completely automate this task are not available. Because of this, the focus of this work has been to give more control to the user in the specification process to enable better visualizations of irregular data types. Thus, for the evaluation, four experts were selected that were familiar with existing transfer function specification tools in their respective research, development, or end user settings. The first expert develops open source visualization software. The second performs research in the area of volume visualization. The third expert has used existing volume visualization software in a medical setting. Finally, the fourth expert is a specialist in bioengineering and concentrates mostly on biomedical computing. These experts were given a demonstration of the system along with ParaView [99], a freely available system that has some basic transfer function specification abilities such as a zooming interface. The experts were then given the opportunity to perform their own explorations using both systems and asked a series of questions about their experience with the system compared to ParaView and other systems that they have used in the past.

Overall, the feedback was very positive and the reviewers feel that the system is useful for fast data exploration and that existing visualization systems would benefit from some of the components that were introduce in the system. The reviewers also provided many suggestions that will be incorporated into the system. The following is a summary of some of the main advantages and disadvantages that the reviewers pointed out.

Advantages:

- The system provides fine-grain control of the data due to the resolution control that range mapping provides.
- Interactive histogram information significantly improves the ability to place widgets and to explore the volume.
- The histograms that update over time provide more information about the volume than other systems provide.
- The ability to interpolate between transfer functions over time is very useful for contextualizing the data.

Disadvantages:

- The interface could use some work to consolidate the concept of ensembles and make it more intuitive.
- Though some of the features are more powerful, they may require longer to learn to use if the user is unfamiliar with transfer function specification.
- Currently there is no undo for operations.

Beyond these general comments, some interesting comments were received about the usefulness of the system from the reviewers' unique perspectives. The first reviewer mentioned that when the application he develops moved from 8-bit data to higher precision, he noticed problems associated with limits in the lookup table precision, though he had not found a reasonable solution for the problem. After the evaluation, he planned to add range mapping to his system. He was also impressed with the idea of creating transfer functions using combinations of other transfer functions and is evaluating this addition to his system as well. The third reviewer worked extensively with time-varying data that changes substantially over time. He commented that the ability to define different transfer functions for time steps and interpolate between them through keyframing would have saved him enormous amounts of time. He also mentioned that the ability to keyframe the range mapping would be a useful feature for these datasets. This feature will be added to the system in the future. The fourth reviewer stated that he would like to see the features presented in the system incorporated into the biomedical simulation software that he and his collaborators use because it would facilitate the process of analysis for time-varying volumes.

7.6 Discussion

Many automatic methods for transfer function specification proposed in the literature focus on identifying and enhancing single features, such as boundary transitions with a 2D gradient magnitude histogram. The transfer function ensembles proposed in this work provide a way for combining different features of several transfer functions. For high dynamic range data, the range mapping proves very efficient for finding features that otherwise are impossible to distinguish. A current limitation of the proposed method is that the range mapping effects all transfer functions in an ensemble. In addition, as pointed out in the evaluation, range mappings effect all time steps. These issues could be resolved with multiple or multidimensional range mapping tables, respectively. An interface similar to the transition curves for time-varying transfer functions could then be used to give the user temporal control over the range mappings.

The transfer function ensembles are blended in texture space due to the simplicity of the approach as well as the performance implications. The disadvantage of performing the blending there is that the number of dimensions that can be combined is limited to three. Another approach to overcome this limitation is to perform the blending at the fragment level as in multivolume approaches (e.g., [88]). This type of blending is more robust for combining transfer functions in different spaces, but is more complex and will effect the rendering performance because it requires multiple lookup tables.

Although this tool was designed with the goal of specifying transfer functions for unstructured data, there are no limitations that prevent it from being applied to structured grids. For example, the datasets shown in Figures 7.3 and 7.7 originate from simulations over structured grids. In addition, the framework can easily be extended with new histograms, widgets, and blending methods. The new features described in this chapter can be easily integrated into existing visualization systems because of their independence from the rendering algorithm.

7.7 Summary

In this chapter, a new framework was proposed for transfer function specification based on several new features. First, statistical information is extracted using multiple histogram types, including time-varying data, and used to facilitate the specification of color and opacity with widgets placed on these histograms. For histograms with high dynamic range, a range mapping equalization was proposed that focuses the analysis into smaller regions of the histogram, providing a more powerful tool than existing zooming techniques. The concept of ensembles was introduced for combining multiple transfer functions with user-controlled blending functions.

Finally, a technique was introduced for transfer function specification on time-varying data that uses keyframing and transition curves to change ensembles over time. The resulting system has been demonstrated to introduce new insights in data by facilitating transfer function specification for disparate data types.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This dissertation has introduced several new techniques for interactive visualization of unstructured grids through volume rendering. In particular, the methods introduced allow the visualization of disparate data types that could not be handled with existing methods. A framework for exploring data that combines many of these techniques was described and evaluated to demonstrate a practical application of the tools developed in this dissertation.

Chapter 3 introduced a new method for accelerating volume rendering of unstructured volumes that works in object-space. By representing the tetrahedral cells as carefully re-shaped points, the amount of data sorted and rasterized is greatly reduced. The result is an algorithm that is many times faster than existing approaches, yet still retains high quality visualizations. For future work, it would be interesting to experiment with programmable geometry shaders to obtain better approximations of tetrahedra, while minimizing the amount of excess fragments that are generated. Furthermore, adding lighting and shadows would be useful to improve the visualizations. A final avenue of future research for this point-based technique would be to adapt it to handle extremely large datasets through parallelization using multiple CPUs and GPUs [137].

An alternative approach for accelerating volume rendering was introduced in Chapter 4. The algorithm operates as a post-process by upsampling an image computed at a low resolution using a joint bilateral filter. The result is a high-quality approximation of the visualization that has been measured to be up to 30 times faster than rendering the full size image. The benefit of the approach is that it is very simple and can be applied to existing algorithms with very little overhead. In the future, it would be useful to explore using additional information to improve the image quality in the upsampling. This could include isosurfaces within the volume or regions of high gradient magnitude (internal boundaries). Exploring the use of the algorithm on structured data would also be beneficial.

While the first part of this dissertation dealt with accelerating volume rendering, Chapter 5 introduced a method for handling static data too large to handle previously. An approach was introduced for performing progressive volume rendering that allows for remote visualization from a data server to a thin client (such as a laptop). Using a bounded amount of memory, the client is capable of accumulating a full quality image by rendering portions of the data incrementally. The advantage of this approach is that it gives high-quality approximations and continuous feedback while remaining interactive, but still allows the user to achieve full resolution visualizations by waiting for the progression to finish. In the future, better techniques for compressing the stream of data and alleviating bandwidth constraints would be beneficial to explore [11]. A more advanced depth culling algorithm and user-controlled cutting planes would also be useful to improve the server performance. In addition, extending the system to handle isosurfacing and time-varying data would also be important.

Another type of data that have been too large to render until now are volumes with a dynamically changing scalar field. Chapter 6 described an algorithm that efficiently handles the problems associated with time-varying scalar fields. Data compression is introduced for handling the storage and data transfer issues, while parallelization amortizes the cost of decompression, sorting, level-of-detail management, and rendering. The resulting framework is capable of handling dynamic data with only a measured performance penalty of 6% over a static approach. For future work, it would be useful to revisit the decompression stage of the algorithm and explore the use of new programmable features in GPUs to perform it more efficiently. It would also be useful to adapt the compression method to automatically select parameters based on dataset characteristics. A major challenge will be to explore solutions for the more complex cases of time-varying geometry and topology.

Finally, in Chapter 7 an application is presented for interactively exploring large unstructured time-varying volumes using direct volume rendering. The application introduces three new components to assist in the specification process for this type of data. The first is a range mapping that facilitates the placement of specification widgets on high dynamic range histograms. The second is an approach for combining multiple transfer functions in ensembles. The third is a method for keyframing tranfer functions to handle time-varying data. The resulting system combines many of the features described in the dissertation into a single framework that facilitates data exploration of unstructured volumes. This was confirmed with an expert user evaluation of the tool. In the future, it would be beneficial to incorporate additional features for structured volumes and prepare this final application for release as an open-source tool for researchers.

REFERENCES

[1] AKIBA, H., FOUT, N., AND MA, K.-L. Simultaneous classification of time-varying volume data based on the time histogram. In *Eurographics/IEEE VGTC Symposium on Visualization* (2006), 171–178.

[2] AKL, S. G. *Parallel Sorting Algorithms*. Academic Press, Inc., 1990.

[3] ANDERSON, E. W., CALLAHAN, S. P., SCHEIDEGGER, C. E., SCHREINER, J., AND SILVA, C. T. Hardware-assisted point-based volume rendering of tetrahedral meshes. In *Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)* (2007), 163–170.

[4] BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. The contour spectrum. In *IEEE Visualization* (1997), 167–ff.

[5] BAVOIL, L., AND MYERS, K. Deferred rendering using a stencil routed k-buffer. In *Shader X6 - Advanced Rendering Techniques*, W. Engel, Ed. Charles River Media, 2008.

[6] BERNARDON, F. F., CALLAHAN, S. P., COMBA, J. L. D., AND SILVA, C. T. Volume rendering of unstructured grids with time-varying scalar fields. In *Eurographics Symposium on Parallel Graphics and Visualization* (2006), 51–58.

[7] BERNARDON, F. F., CALLAHAN, S. P., COMBA, J. L. D., AND SILVA, C. T. An adaptive framework for visualizing unstructured grids with time-varying scalar fields. *Parallel Computing 33*, 6 (2007), 391–405.

[8] BERNARDON, F. F., HA, L. K., CALLAHAN, S. P., COMBA, J. L. D., AND SILVA, C. T. Interactive transfer function specification for direct volume rendering of disparate volumes. Tech. Rep. UUSCI-2007-007, Scientific Computing and Imaging Institute, University of Utah, 2007.

[9] BERNARDON, F. F., PAGOT, C. A., COMBA, J. L. D., AND SILVA, C. T. GPU-based tiled ray casting using depth peeling. *Journal of Graphics Tools 11*, 3 (2006), 23–29.

[10] BETHEL, W., TIERNEY, B., LEE, J., GUNTER, D., AND LAU, S. Using high-speed WANs and network data caches to enable remote and distributed visualization. In *IEEE/ACM Supercomputing* (2000), 28.

[11] BISCHOFF, U., AND ROSSIGNAC, J. Tetstreamer: Compressed back-to-front transmission of delauney tetrahedra meshes. In *Proceedings of Data Compression Conference* (2005), 93–102.

[12] BLINN, J. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics 16*, 3 (1982), 21–29.

[13] BOTSCH, M., HORNUNG, A., ZWICKER, M., AND KOBBELT, L. High-quality surface splatting on today's GPUs. *Eurographics Symposium on Point-Based Graphics* (2005), 17–24.

[14] BRUCKNER, S., AND GRÖLLER, E. Volume shop: An interactive system for direct volume rendering. In *IEEE Visualization* (2005), 671–678.

[15] BUNYK, P., KAUFMAN, A., AND SILVA, C. T. Simple, fast, and robust ray casting of irregular grids. In *Proc. of Dagstuhl, Scientific Visualization* (1997), 30–36.

[16] CABRAL, B., CAM, N., AND FORAN, J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *IEEE/ACM Symposium on Volume Visualization* (1994), 91–98.

[17] CALLAHAN, S. P. The K-buffer and its applications to volume rendering. Master's thesis, University of Utah, 2005.

[18] CALLAHAN, S. P., BAVOIL, L., PASCUCCI, V., AND SILVA, C. T. Progressive volume rendering of large unstructured grids. *IEEE Transactions on Visualization and Computer Graphics (Proc. Visualization) 12*, 5 (Sept/Oct 2006), 1307–1314.

[19] CALLAHAN, S. P., BAVOIL, L., PASCUCCI, V., AND SILVA, C. T. Progressive volume rendering of unstructured grids on modern GPUs. In *ACM SIGGRAPH Sketches* (2006).

[20] CALLAHAN, S. P., CALLAHAN, J. H., SCHEIDEGGER, C. E., AND SILVA, C. T. Direct volume rendering: A 3D plotting technique for scientific data. *Computing in Science & Engineering 10*, 1 (2008), 88–92.

[21] CALLAHAN, S. P., COMBA, J. L. D., SHIRLEY, P., AND SILVA, C. T. Interactive rendering of large unstructured grids using dynamic level-of-detail. In *IEEE Visualization* (2005), 199–206.

[22] CALLAHAN, S. P., IKITS, M., COMBA, J. L., AND SILVA, C. T. Hardware-assisted visibility sorting for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics 11*, 3 (2005), 285–295.

[23] CALLAHAN, S. P., AND SILVA, C. T. Image-space acceleration for direct volume rendering of unstructured grids using joint bilateral upsampling. submitted, 2008.

[24] CARPENTER, L. The A-buffer, an antialiased hidden surface method. *Computer Graphics (Proc. ACM SIGGRAPH) 18*, 3 (1984), 103–108.

[25] CHEN, W., REN, L., ZWICKER, M., AND PFISTER, H. Hardware-accelerated adaptive EWA volume splatting. *IEEE Visualization* (2004), 67–74.

[26] CHOURASIA, A., AND SHULZE, J. Data centric transfer functions for high dynamic range volume data. In *International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision* (2007), 9–16.

[27] CIGNONI, P., FLORIANI, L. D., MAGILLO, P., PUPPO, E., AND SCOPIGNO, R. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE*

Transactions on Visualization and Computer Graphics 10, 1 (2004), 29–45.

[28] COMBA, J., KLOSOWSKI, J. T., MAX, N., MITCHELL, J. S. B., SILVA, C. T., AND WILLIAMS, P. L. Fast polyhedral cell sorting for interactive rendering of unstructured grids. *Computer Graphics Forum 18*, 3 (Sept. 1999), 369–376.

[29] COOK, R., HALSTEAD, J., PLANCK, M., AND RYU, D. Stochastic simplification of aggregate detail. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3 (2007), 79.

[30] COOK, R., MAX, N., SILVA, C. T., AND WILLIAMS, P. Efficient, exact visibility ordering of unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics 10(6)* (2004), 695–707.

[31] CORRÊA, W. T., KLOSOWSKI, J. T., AND SILVA, C. T. iWalk: Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University, 2002.

[32] DANSKIN, J., AND HANRAHAN, P. Fast algorithms for volume ray tracing. In *Workshop on Volume Visualization* (1992), 91–98.

[33] DESGRANGES, P., ENGEL, K., AND PALADINI, G. Gradient-free shading: A new method for realistic interactive volume rendering. In *Vision, Modeling, and Visualization* (2005), 209–216.

[34] DOI, A., AND KOIDE, A. An efficient method of triangulating equivalued surfaces by using tetrahedral cells. *IEICE Transactions Communication, Elec. Info. Syst. E74*, 1 (1991), 214–224.

[35] DOLEISCH, H., MAYER, M., GASSER, M., PRIESCHING, P., AND HAUSER, H. Interactive feature specification for simulation data on time-varying grids. In *Conference on Simulation and Visualization (SimVis)* (2005), 291–304.

[36] DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. Volume rendering. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1988), 65–74.

[37] DURAND, F., AND DORSEY, J. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics (Proc. SIGGRAPH) 21*, 3 (2002), 257–266.

[38] DURAND, F., HOLZSCHUCH, N., SOLER, C., CHAN, E., AND SILLION, F. X. A frequency analysis of light transport. *ACM Transactions on Graphics (Proc. SIGGRAPH) 24*, 3 (2005), 1115–1126.

[39] EISEMANN, E., AND DURAND, F. Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics (Proc. SIGGRAPH) 23*, 3 (2004), 673–678.

[40] ELLSWORTH, D., CHIANG, L.-J., AND SHEN, H.-W. Accelerating time-varying hardware volume rendering using TSP trees and color-based error metrics. In *Volume Visualization Symposium* (2000), 119–128.

[41] ENGEL, K., KRAUS, M., AND ERTL, T. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. *ACM SIGGRAPH/EG workshop on Graphics*

Hardware (2001), 9–16.

[42] ENGEL, K., SOMMER, O., ERNST, C., AND ERTL, T. Progressive isosurfaces on the web. Late Breaking Hot Topics, *IEEE Visualization*, 1998.

[43] ENGEL, K., SOMMER, O., AND ERTL, T. A framework for interactive hardware accelerated remote 3D-visualization. In *EG/IEEE TCVG Symposium on Visualization (VisSym)* (2000).

[44] EVERITT, C. Interactive order-independent transparency. White paper, NVidia Corporation, 1999.

[45] FARIAS, R., MITCHELL, J., AND SILVA, C. T. ZSWEEP: An efficient and exact projection algorithm for unstructured volume rendering. In *IEEE Volume Visualization and Graphics Symposium* (2000), 91–99.

[46] FLEISHMAN, S., DRORI, I., AND COHEN-OR, D. Bilateral mesh denoising. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (2003), 950–953.

[47] FOUT, AKIBA, H., MA, K.-L., LEFOHN, A., AND KNISS, J. M. High-quality rendering of compressed volume data formats. In *EuroVis* (2005), 77–84.

[48] FUJISHIRO, I., AZUMA, T., AND TAKESHIMA, Y. Automating transfer function design for comprehensible volume rendering based on 3D field topology analysis. In *IEEE Visualization* (1999), 467–470.

[49] GARLAND, M., AND ZHOU, Y. Quadric-based simplification in any dimension. *ACM Transactions on Graphics 24*, 2 (Apr. 2005), 209–239.

[50] GARRITY, M. P. Raytracing irregular volume data. *Computer Graphics (San Diego Workshop on Volume Visualization) 24*, 5 (Nov. 1990), 35–40.

[51] GEORGII, J., AND WESTERMANN, R. A generic and scalable pipeline for GPU tetrahedral grid rendering. *IEEE Transactions on Visualization and Computer Graphics (Proc. Visualization) 12*, 5 (2006), 1345–1352.

[52] GUTHE, S., AND STRASSER, W. Real-time decompression and visualization of animated volume data. In *IEEE Visualization* (2001), 349–356.

[53] GUTHE, S., WAND, M., GONSER, J., AND STRASSER, W. Interactive rendering of large volume data sets. In *IEEE Visualization '02* (2002), 53–60.

[54] HADWIGER, M., BERGER, C., AND HAUSER, H. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *IEEE Visualization* (2003), 301–308.

[55] HANSEN, C., AND JOHNSON, C. *The Visualization Handbook*. Academic Press, 2004.

[56] HAVS. http://havs.sourceforge.net.

[57] HUMPHREYS, G., HOUSTON, M., NG, R., FRANK, R., AHERN, S., KIRCHNER, P. D., AND KLOSOWSKI, J. T. Chromium: a stream-processing framework for interactive

rendering on clusters. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2002), 693–702.

[58] JANKUN-KELLY, T., AND MA, K.-L. A study of transfer function generation for time-varying volume data. In *Volume Graphics Workshop* (2001), 51–65.

[59] KAEHLER, R., PROHASKA, S., HUTANU, A., AND HEGE, H.-C. Visualization of time-dependent remote adaptive mesh refinement data. In *IEEE Visualization* (2005), 175–182.

[60] KAJIYA, J., AND HERZEN, B. V. Ray tracing volume densities. *Computer Graphics 18*, 3 (1984), 165–174.

[61] KINDLMANN, G., AND DURKIN, J. W. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization* (1998), 79–86.

[62] KNISS, J., KINDLMANN, G., AND HANSEN, C. Multi-dimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (July 2002), 270–285.

[63] KNISS, J., PREMOZE, S., IKITS, M., LEFOHN, A., HANSEN, C., AND PRAUN, E. Gaussian transfer functions for multi-field volume visualization. In *IEEE Visualization* (2003), 497–504.

[64] KÖNIG, A., AND GRÖLLER, E. Mastering transfer function specification by using VolumePro technology. *Spring Confererence on Computer Graphics 17* (2001), 279–286.

[65] KOPF, J., COHEN, M., LISCHINSKI, D., AND UYTTENDAELE, M. Joint bilateral upsampling. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3 (2007).

[66] KRAUS, M., AND ERTL, T. Cell-projection of cyclic meshes. In *IEEE Visualization* (2001), 215–222.

[67] KRAUS, M., QIAO, W., AND EBERT, D. S. Projecting tetrahedra without rendering artifacts. In *IEEE Visualization* (2004), 27–34.

[68] KRÜGER, J., AND WESTERMANN, R. Acceleration techniques for GPU-based volume rendering. *IEEE Visualization* (2003), 38–45.

[69] LAMAR, E., HAMANN, B., AND JOY, K. Multiresolution techniques for interative texture-based volume visualization. In *IEEE Visualization* (1999), 355–361.

[70] LAUR, D., AND HANRAHAN, P. Hierarchical splatting: a progressive refinement algorithm for volume rendering. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1991), 285–288.

[71] LEVEN, J., CORSO, J., COHEN, J. D., AND KUMAR, S. Interactive visualization of unstructured grids using hierarchical 3D textures. In *IEEE Symposium on Volume Visualization and Graphics* (2002), 37–44.

[72] LEVOY, M. Efficient ray tracing of volume data. *ACM Transactions on Graphics 9*, 3 (1990), 245–261.

[73] LEVOY, M. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications 2*, 4 (1990), 33–40.

[74] LEVOY, M. Volume rendering by adaptive refinement. *The Visual Computer 6*, 1 (1990), 2–7.

[75] LEVOY, M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1998), 29–37.

[76] LIPPERT, L., GROSS, M. H., AND KURMANN, C. Compression domain volume rendering for distributed environments. *Computer Graphics Forum 16*, 3 (1997), C95–C107.

[77] LIVNAT, Y., AND TRICOCHE, X. Interactive point based isosurface extraction. In *IEEE Visualization* (2004), 457–464.

[78] LOOP, C., AND BLINN, J. Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics (Proc. SIGGRAPH) 24*, 3 (2005), 1000–1009.

[79] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (Proc. SIGGRAPH) 21*, 4 (1987), 163–169.

[80] LUM, E., MA, K.-L., AND CLYNE, J. Texture hardware assisted rendering of time-varying volume data. In *IEEE Visualization* (2001), 263–270.

[81] LUM, E., MA, K.-L., AND CLYNE, J. A hardware-assisted scalable solution of interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 286–301.

[82] LUM, E. B., AND MA, K.-L. Lighting transfer functions using gradient aligned sampling. In *IEEE Visualization* (2004), 289–296.

[83] MA, K.-L. Image graphs-a novel approach to visual data exploration. In *IEEE Visualization* (1999), 81–88.

[84] MA, K.-L. Visualizing time-varying volume data. *Computing in Science & Engineering 5*, 2 (2003), 34–42.

[85] MA, K.-L., BLONDIN, J., CHEN, J. H., RAST, M., AND SAMTANEY, R. Meet the scientists. In *IEEE Visualization Panels* (2007).

[86] MA, K.-L., AND LUM, E. Techniques for visualizing time-varying volume data. In *Visualization Handbook*, C. D. Hansen and C. Johnson, Eds. Academic Press, 2004.

[87] MA, K.-L., AND SHEN, H.-W. Compression and accelerated rendering of time-varying volume data. In *International Computer Symposium Workshop on Computer Graphics and Virtual Reality* (2000), 82–89.

[88] MANAGULI, R., YOO, Y. M., AND KIM, Y. Multi-volume rendering for three-dimensional power doppler imaging. *IEEE Ultrasonics Symposium 4* (2005), 2046–2049.

[89] MAO, X., HONG, L., AND KAUFMAN, A. Splatting of curvilinear volumes. *IEEE*

Visualization (1995), 61–68.

[90] Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J., Kang, T., Mirtich, B., Pfister, H., Ruml, W., Ryall, K., Seims, J., and Shieber, S. Design galleries: A general approach to setting parameters for computer graphics and animation. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1997), 389–400.

[91] Marroquim, R., Maximo, A., Farias, R., and Esperanca, C. GPU-based cell projection for interactive volume rendering. *Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)* (2006), 147–154.

[92] Max, N. L. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 99–108.

[93] McCormick, P. S., Inman, J., Ahrens, J. P., Hansen, C., and Roth, G. Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In *IEEE Visualization* (2004), 171–178.

[94] Moreland, K., and Angel, E. A fast high accuracy volume renderer for unstructured data. In *IEEE Symposium on Volume Visualization and Graphics* (2004), 9–16.

[95] Moreland, K., Wylie, B., and Pavlakos, C. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *IEEE Symposium on Parallel and Large-data Visualization and Graphics* (2001), 85–92.

[96] Muigg, P., Hadwiger, M., Doleisch, H., and Hauser, H. Scalable hybrid unstructured and structured grid raycasting. *IEEE Transactions on Visualization and Computer Graphics (Proc. Visualization) 13*, 6 (2007), 1592–1599.

[97] Munzner, T., Johnson, C., Moorhead, R., Pfister, H., Rheingans, P., and Yoo, T. S. NIH-NSF visualization research challenges report summary. *IEEE Computer Graphics and Applications 26*, 2 (2006), 20–24.

[98] Museth, K., and Lombeyda, S. TetSplat: Real-time rendering and volume clipping of large unstructured tetrahedral meshes. In *IEEE Visualization* (2004), 433–440.

[99] ParaView. http://www.paraview.org.

[100] Parker, S., Parker, M., Livnat, Y., Sloan, P.-P., Hansen, C., and Shirley, P. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 238–250.

[101] Parker, S., Shirley, P., Livnat, Y., Hansen, C., and Sloan, P.-P. Interactive ray tracing for isosurface rendering. In *IEEE Visualization* (1998), 233–238.

[102] Pascucci, V. Isosurface computation made simple: Hardware acceleration, adaptive refinement, and tetrahedral stripping. In *Eurographics/IEEE VGTC Symposium on Visualization* (2004), 293–300.

[103] Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., and

Toyama, K. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics (Proc. SIGGRAPH) 23*, 3 (2004), 664–672.

[104] Pfister, H., Lorensen, B., Bajaj, C., Kindlmann, G., Schroeder, W., Avila, L. S., Martin, K., Machiraju, R., and Lee, J. The transfer function bake-off. *IEEE Computer Graphics and Applications 21*, 3 (2001), 16–22.

[105] Pfister, H., Zwicker, M., van Baar, J., and Gross, M. Surfels: surface elements as rendering primitives. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2000), 335–342.

[106] Pharr, M., and Humphreys, G. *Physically Based Rendering*. Morgan Kaufmann, 2004, ch. 4.4.

[107] Porter, T., and Duff, T. Compositing digital images. *Computer Graphics (Proc. SIGGRAPH) 18*, 3 (1984), 253–259.

[108] Potts, S., and Möller, T. Transfer functions on a logarithmic scale for volume rendering. In *Graphics Interface* (2004), 57–63.

[109] Preim, B., and Bartz, D. *Visualization in Medicine*. Morgan Kaufmann, 2007.

[110] Qiao, W., Ebert, D. S., Entezari, A., Korkusinski, M., and Klimeck, G. Volqd: Direct volume rendering of multi-million atom quantum dot simulations. In *IEEE Visualization* (2005), 319–326.

[111] Reed, D. M., Yagel, R., Law, A., Shin, P.-W., and Shareef, N. Hardware assisted volume rendering of unstructured grids by incremental slicing. In *Symposium on Volume Visualization* (1996), 55–ff.

[112] Rezk-Salama, C., Keller, M., and Kohlmann, P. High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006).

[113] Roettger, S., Bauer, M., and Stamminger, M. Spatialized transfer functions. In *IEEE VGTC/EG Symposium on Visualization* (2005), 271–278.

[114] Roettger, S., and Ertl, T. A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *IEEE Volume Visualization and Graphics Symposium* (2002), 23–28.

[115] Röettger, S., Kraus, M., and Ertl, T. Hardware-accelerated volume and isosurface rendering based on cell-projection. *IEEE Visualization* (2000), 109–116.

[116] Rusinkiewicz, S., and Levoy, M. QSplat: a multiresolution point rendering system for large meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2000), 343–352.

[117] Sabella, P. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (Proc. SIGGRAPH) 22*, 4 (1988), 51–58.

[118] Sawhney, H. S., Guo, Y., Hanna, K., Kumar, R., Adkins, S., and Zhou, S.

Hybrid stereo camera: an IBR approach for synthesis of very high resolution stereoscopic image sequences. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2001), 451–460.

[119] SCHNEIDER, J., AND WESTERMANN, R. Compression domain volume rendering. In *IEEE Visualization* (2003), 39–48.

[120] SEREDA, P., BARTROLI, A. V., SERLIE, I. W. O., AND GERRITSEN, F. A. Visualization of boundaries in volumetric datasets using LH histograms. *IEEE Transactions on Visualization and Computer Graphics 12*, 2 (March/April 2006), 208–218.

[121] SHAREEF, N., LEE, T.-Y., SHEN, H.-W., AND MUELLER, K. An image-based modelling approach to GPU-based rendering of unstructured grids. In *Volume Graphics* (2006), 31–38.

[122] SHEN, H.-W., CHIANG, L.-J., AND MA, K.-L. A fast volume rendering algorithm for time-varying field using a time-space partitioning (TSP) tree. In *IEEE Visualization* (1999), 371–377.

[123] SHIRLEY, P., AND TUCHMAN, A. A polygonal approximation to direct scalar volume rendering. *Proc. San Diego Workshop on Volume Visualization 24*, 5 (Nov. 1990), 63–70.

[124] SILVA, C. T., COMBA, J. L. D., CALLAHAN, S. P., AND BERNARDON, F. F. GPU-based volume rendering of unstructured grids. *Brazilian Journal of Theoretic and Applied Computing (RITA) 12*, 2 (2005), 9–29.

[125] SILVA, C. T., MITCHELL, J. S., AND WILLIAMS, P. L. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *IEEE Symposium on Volume Visualization* (1998), 87–94.

[126] STRENGERT, M., MAGALLÓN, M., WEISKOPF, D., GUTHE, S., AND ERTL, T. Hierarchical visualization and compresssion of large volume datasets using GPU clusters. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (2004), 41–48.

[127] SUTHERLAND, I. E., SPROULL, R. F., AND SCHUMACKER, R. A. A characterization of ten hidden surface algorithms. *ACM Computing Surveys 6*, 1 (1974), 1–55.

[128] SVAKHINE, N., EBERT, D. S., AND STREDNEY, D. Illustration motifs for effective medical volume illustration. *IEEE Computer Graphics and Applications 25*, 3 (2005), 31–39.

[129] TOMASI, C., AND MANDUCHI, R. Bilateral filtering for gray and color images. In *International Conference on Computer Vision* (1998), 839–846.

[130] TORY, M., AND MÖLLER, T. Evaluating visualizations: Do expert reviews work? *IEEE Computer Graphics and Applications 25*, 5 (2005), 8–11.

[131] TORY, M., POTTS, S., AND MÖLLER, T. A parallel coordinates style interface for exploratory volume visualization. *IEEE Transactions on Visualization and Computer Graphics 11*, 1 (2005), 71–80.

[132] TZENG, F.-Y., LUM, E. B., AND MA, K.-L. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics 11*, 3 (2005), 273–284.

[133] VANNIER, M., MARSH, J., AND WARREN, J. Three dimensional computer graphics for craniofacial surgical planning and evaluation. In *Computer Graphics and Interactive Techniques* (1983), 263–273.

[134] VIOLA, I., KANITSAR, A., AND GRÖLLER, E. Importance-driven volume rendering. In *IEEE Visualization* (2004), 139–145.

[135] VizServer. http://www.sgi.com/products/software/vizserver.

[136] VO, H. T., CALLAHAN, S. P., LINDSTROM, P., PASCUCCI, V., AND SILVA, C. T. Streaming simplification of tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (Jan/Feb 2007), 145–155.

[137] VO, H. T., CALLAHAN, S. P., SMITH, N., SILVA, C. T., MARTIN, W., OWEN, D., AND WEINSTEIN, D. iRun: Interactive rendering of large unstructured grids. *Eurographics Symposium on Parallel Graphics and Visualization* (2007), 93–100.

[138] WEILER, M., KRAUS, M., MERZ, M., AND ERTL, T. Hardware-based ray casting for tetrahedral meshes. In *IEEE Visualization* (2003), 333–340.

[139] WEILER, M., KRAUS, M., MERZ, M., AND ERTL, T. Hardware-based view-independent cell projection. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 163–175.

[140] WEILER, M., MALLÓN, P. N., KRAUS, M., AND ERTL, T. Texture-encoded tetrahedral strips. In *Symposium on Volume Visualization* (2004), 71–78.

[141] WEILER, M., WESTERMANN, R., HANSEN, C., ZIMMERMAN, K., AND ERTL, T. Level-of-detail volume rendering view 3D textures. In *IEEE Symposium on Volume Visualization* (2000), 7–13.

[142] WESTERMANN. Compression time rendering of time-resolved volume data. In *IEEE Visualization '95* (1995), 168–174.

[143] WILHELMS, J., AND GELDER, A. V. Octrees for faster isosurface generation. *ACM Transaction on Graphics 11*, 3 (1992), 201–227.

[144] WILLIAMS, P. L. Visibility-ordering meshed polyhedra. *ACM Transactions on Graphics 11*, 2 (1992), 103–126.

[145] WOLBERG, G. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[146] WOODRING, J., AND SHEN, H.-W. Multi-variate, time-varying, and comparative visualization with contextual cues. *IEEE Transactions on Visualization and Computer Graphics (Proc. Visualization) 12*, 5 (2006), 909–916.

[147] WU, Y., QU, H., ZHOU, H., AND CHAN, M.-Y. Focus + context visualization with

animations. In *IEEE Pacific-Rim Symposium on Image and Video Technology* (2006), 1293–1302.

[148] WU, Y., QU, H., ZHOU, H., AND CHAN, M.-Y. Fusing features in direct volume rendering images. In *International Symposium on Visual Computing* (2006), 273–282.

[149] WYLIE, B., MORELAND, K., FISK, L. A., AND CROSSNO, P. Tetrahedral projection using vertex shaders. *IEEE Symposium on Volume Visualization and Graphics* (2002), 7–12.

[150] XUE, D., AND CRAWFIS, R. Efficient splatting using modern graphics hardware. *Journal of Graphics Tools 8*, 3 (2004), 1–21.

[151] YOUNESY, H., MÖLLER, T., AND CARR, H. Visualization of time-varying volumetric data using differential time-histogram table. In *IEEE VGTC/Eurographics Workshop on Volume Graphics* (2005), 21–30.

[152] YU, H., MA, K.-L., AND WELLING, J. I/O strategies for parallel rendering of large time-varying volume data. In *Eurographics Symposium on Parallel Graphics and Visualization* (2004), 31–40.

[153] YUAN, X., NGUYEN, M. X., CHEN, B., AND PORTER, D. H. HDR VolVis: High dynamic range volume visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (July/August 2006), 433–455.

[154] zlib. http://www.zlib.net.

[155] ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. EWA volume splatting. *IEEE Visualization* (2001), 29–36.